

Toward More Localized Local Algorithms: Removing Assumptions Concerning Global Knowledge

[Extended Abstract]

Amos Korman^{*}
CNRS & Univ. Paris Diderot
Paris
France

Amos.Korman@liafa.jussieu.fr

Jean-Sébastien Sereni[†]
CNRS
Paris
France

sereni@kam.mff.cuni.cz

Laurent Viennot[‡]
INRIA & Univ. Paris Diderot
Paris
France

Laurent.Viennot@inria.fr

ABSTRACT

Numerous sophisticated local algorithms were suggested in the literature for various fundamental problems. Notable examples are the MIS and $(\Delta + 1)$ -coloring algorithms by Barenboim and Elkin [6], by Kuhn [22], and by Panconesi and Srinivasan [33], as well as the $O(\Delta^2)$ -coloring algorithm by Linial [27]. Unfortunately, most known local algorithms (including, in particular, the aforementioned algorithms) are *non-uniform*, that is, they assume that all nodes know good estimations of one or more global parameters of the network, e.g., the maximum degree Δ or the number of nodes n .

This paper provides a rather general method for transforming a non-uniform local algorithm into a *uniform* one. Furthermore, the resulting algorithm enjoys the same asymptotic running time as the original non-uniform algorithm. Our method applies to a wide family of both deterministic and randomized algorithms. Specifically, it applies to almost all of the state of the art non-uniform algorithms regarding MIS and Maximal Matching, as well as to many results concerning the coloring problem. (In particular, it applies to all aforementioned algorithms.)

To obtain our transformations we introduce a new distributed tool called *pruning algorithms*, which we believe may be of independent interest.

^{*}Supported in part by a France-Israel cooperation grant (“Mutli-Computing” project) from the France and Israel Ministries of Science, by the ANR projects ALADDIN and PROSE, and by the INRIA project-team GANG.

[†]CNRS (LIAFA, Université Denis Diderot), Paris, France and Department of Applied Mathematics (KAM), Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic. Supported in part by the French *Agence Nationale de la Recherche* under reference ANR 10 JCJC 0204 01.

[‡]INRIA project-team GANG, supported by the european STREP project EULER, and the ANR project PROSE.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC’11, June 6–8, 2011, San Jose, California, USA.

Copyright 2011 ACM 978-1-4503-0719-2/11/06 ...\$5.00.

Categories and Subject Descriptors

G.2.2 [Discrete Mathematics]: Graph Theory—*Graph algorithms, Graph labeling, Network problems*

General Terms

Algorithms

Keywords

Distributed algorithm, global knowledge, parameters, MIS, coloring, Maximal matching

1. INTRODUCTION

1.1 Background and motivation

Distributed computing concerns environments in which many processors, located at different sites, must collaborate in order to achieve some global task. One of the main themes in distributed network algorithms concerns the question of how to cope with *locality* constraints, that is, the lack of knowledge about the global structure of the network (cf., [34]). On the one hand, information about the global structure may not always be accessible to individual processors and the cost of computing it from scratch may overshadow the cost of the algorithm using it. On the other hand, global knowledge is not always essential, and many seemingly global tasks can be efficiently achieved by letting processors know more about their immediate neighborhoods and less about the rest of the network.

A standard model for capturing the essence of locality is the *LOCAL* model (cf., [34]). In this model, the network is modeled by a graph $G = (V, E)$, where the nodes of G represent the processors and the edges represent the communication links. To perform a task, nodes are woken up simultaneously, and computation proceeds in fault-free synchronous rounds during which every node exchanges messages with its neighbors, and performs arbitrary computations on its data. Since many tasks cannot be solved distributively in an anonymous network, symmetry breaking must be addressed. The typical way to address this issue is by assuming that a unique identity $\text{Id}(v)$ is initially provided to each node v in the network, and encoded using $O(\log n)$ bits, where n is the number of nodes in the network. A *local algorithm* operating in such a setting must return an output at each node such that the collection of outputs satisfies the required

task. For example, in the Maximal Independent Set (MIS) problem, the output at each node v is a bit $b(v)$ indicating whether v belongs to a selected set $S \subseteq V$ of nodes, and it is required that S forms a MIS of G . The *running time* of a local algorithm is the number of rounds needed for the algorithm to complete its operation at each node, taken in the worst case scenario. This is typically evaluated with respect to some parameters of the underlying graph. The common parameters used are the number of nodes n in the graph and the maximum degree Δ of a node in the graph.

To ease the computation, it is often assumed that some kind of knowledge about the global network (e.g., the number of nodes) is provided to each node *a priori*. Actually, the amount and type of such information may have a profound effect on the design of the distributed algorithm. Obviously, if the whole topology of the network is known to each node in advance, then the distributed algorithm can be reduced to a central one. In fact, the whole area of *computation with advice* [9, 12, 13, 14, 15, 20, 21] is dedicated to studying the amount of information known to nodes and its effect on the performances of the distributed algorithm. For instance, Fraignaud *et al.* [15] showed that if each node is provided with only a constant number of bits then one can locally construct a BFS-tree in constant time, and can locally construct a MST in $O(\log n)$ time, while both tasks require diameter time if no knowledge is assumed. As another example, Cohen *et al.* [9] showed that $O(1)$ bits, chosen judiciously at each node, can allow a finite automata to distributively explore every graph. As a matter of fact, from a radical point of view, for many questions (e.g., MIS, Maximal Matching), additional information may push the question at hand into absurdity: even a constant number of bits of additional information per node is enough to compute a solution—simply let the additional information encode the solution!

When dealing with locality issues, it is desired that the amount of information that is known to nodes regarding the whole network is minimized. A local algorithm that assumes that each node initially knows merely its own identity is often called *uniform*. Unfortunately, there are only few local algorithms in the literature that are uniform (e.g., [11, 25, 28, 29, 36]). In contrast, most known local algorithms assume that all nodes know upper bounds on the values of some global parameters of the network. Moreover, it is often assumed that all nodes agree on their candidates for being these upper bounds. Furthermore, typically, not only the correct operation of the algorithm requires the knowledge of the upper bounds, but also its running time is actually a function of the upper bound estimations and not of the actual value of the parameters. Hence, it is desired that the known upper bounds are not significantly larger than the real values of the parameters.

Some attempts to transform a non-uniform local algorithm into a uniform one were made by examining the details of the algorithm at hand and modifying it appropriately. For example, Barenboim and Elkin [6] first gave a non-uniform MIS algorithm for the family of graphs with arboricity $a = O(\log^{1/2-\delta} n)$, for some constant $\delta \in (0, 1/2)$, running in time $O(\log n / \log \log n)$. At the cost of increasing the running time to $O(\frac{\log n}{\log \log n} \log^* n)$, the authors show how to modify their algorithm to not require the knowledge of a . (Nevertheless, their algorithm still requires nodes to agree on an upper bound on n .)

In this paper, we present a rather general method for

transforming a non-uniform local algorithm into a uniform one without increasing the asymptotic running time of the original algorithm. Our method can apply to a wide family of both deterministic and randomized algorithms. In particular, our method can apply to all of the state of the art non-uniform algorithms regarding MIS and Maximal Matching, as well as to several of the best known results concerning the coloring problem.

Our transformations are obtained using a new type of local algorithms termed *pruning algorithms*. Informally, the basic property of a pruning algorithm is that it allows one to iteratively apply a sequence of local algorithms (whose output may not form a correct global solution) one after the other, in a way that “always progresses” toward a solution. In a sense, a pruning algorithm is a combination of a gluing mechanism and a *local checking* algorithm (cf., [16, 31]). A local checking algorithm for a problem Π runs on graphs with an output value at each node (and possibly an input too), and can locally detect whether the output is “legal” with respect to Π . That is, if the instance is not legal then at least one node detects this, and raises an alarm. (For example, a local checking algorithm for MIS is trivial: each node in the set S , which is suspected to be a MIS, checks that none of its neighbors belongs to S , and each node not in S checks that at least one of its neighbors belongs to S . If the check fails, then the node raises an alarm.) A pruning algorithm needs to satisfy an additional *gluing* property not required by local checking algorithms. Specifically, if the instance is not legal, then the pruning algorithm must carefully choose the nodes raising the alarm (and possibly modify their input too), so that a solution for the subgraph induced by those alarming nodes can be well glued to the previous output of the non-alarming nodes, in a way such that the combined output is a solution to the whole initial graph.

We believe that this new type of algorithms may be of independent interest. Indeed, as we show, pruning algorithms have several types of other applications in the theory of local computation, besides the aforementioned issue of designing uniform algorithms. Specifically, they can be used also to transform a local Monte-Carlo algorithm into a Las Vegas one, as well as to obtain an algorithm that runs in the minimum running time of a given set of uniform algorithms.

1.2 Previous work

MIS and coloring: There is a long line of research concerning the two related classical $(\Delta + 1)$ -coloring and MIS problems [3, 10, 17, 18, 23, 24, 27]. Recently, Barenboim and Elkin [4] and independently Kuhn [22] presented two elegant $(\Delta + 1)$ -coloring and MIS algorithms running in $O(\Delta + \log^* n)$ time on general graphs. This is the current best bound for these problems on low degree graphs. For graphs with a large maximum degree Δ , the best bound is due to Panchon and Srinivasan [33], who devised an algorithm running in $2^{O(\sqrt{\log n})}$ time. The aforementioned algorithms are non-uniform. Specifically, all three algorithms require that all nodes know and agree on an upper bound on n and the first two also require an upper bound on Δ .

For bounded-independence graphs, Schneider and Wattenhofer [36] designed uniform deterministic MIS and $(\Delta + 1)$ -coloring algorithms running in $O(\log^* n)$ time. Barenboim and Elkin devised [6] a deterministic algorithm for the MIS problem on graphs of bounded arboricity that requires time $O(\log n / \log \log n)$. More specifically, for graphs with ar-

| Problem | Parameters | Time | Ref. | This paper (uniform) | Theorem |
|---|-----------------|---------------------------------|---------|---|-------------|
| Det. MIS and $(\Delta+1)$ -coloring | n, Δ | $O(\Delta + \log^* n)$ | [4, 22] | $\min \{O(\Delta + \log^* n), 2^{O(\sqrt{\log n})}\}$ | Th. 1 Th. 2 |
| | n | $2^{O(\sqrt{\log n})}$ | [33] | | |
| Det. MIS (arboricity $a = o(\sqrt{\log n})$) | n, a | $o(\log n)$ | [6] | $o(\log n)$ | Th. 1 |
| Det. MIS (arboricity $a = O(\log^{1/2-\delta} n)$) | n, a | $O(\log n / \log \log n)$ | [6] | $O(\log n / \log \log n)$ | Th. 1 |
| Det. $\lambda(\Delta+1)$ -coloring | n, Δ | $O(\Delta/\lambda + \log^* n)$ | [4, 22] | $O(\Delta/\lambda + \log^* n)$ | Th. 3 |
| Det. $O(\Delta)$ -edge coloring | n, Δ | $O(\Delta^\epsilon + \log^* n)$ | [7] | $O(\Delta^\epsilon + \log^* n)$ | Th. 4 |
| Det. $O(\Delta^{1+\epsilon})$ -edge coloring | n, Δ | $O(\log \Delta + \log^* n)$ | [7] | $O(\log \Delta + \log^* n)$ | Th. 4 |
| Det. Maximal Matching | n or Δ | $O(\log^4 n)$ | [19] | $O(\log^4 n)$ | Th. 5 |
| Rand. MIS | uniform | $O(\log n)$ | [29, 1] | | |
| Rand. $(2, 2(c+1))$ -ruling-set | n | $O(2^c \log^{1/c} n)$ | [35] | $O(2^c \log^{1/c} n)$ | Th. 6 |

Table 1: Comparison of \mathcal{LOCAL} algorithms with respect to global parameter knowledge. “Det.” stands for deterministic, and “Rand.” for randomized.

boricity $a = o(\sqrt{\log n})$, they show that a MIS can be computed deterministically in $o(\log n)$ time, and whenever $a = O(\log^{1/2-\delta} n)$ for some constant $\delta \in (0, 1/2)$, the same algorithm runs in time $O(\log n / \log \log n)$. At the cost of increasing the running time by a multiplicative factor of $O(\log^* n)$, the authors show how to modify their algorithm to not require the knowledge of a . Nevertheless, all their algorithms require all nodes to know and agree on an upper bound on the value of n . Deterministic non-uniform coloring algorithms with number of colors and running time corresponding to the arboricity parameter were given by Barenboim and Elkin [5, 6].

Concerning the problem of coloring with more than $\Delta + 1$ colors, Linial [26, 27], and subsequently Szegedy and Vishwanathan [37], described $O(\Delta^2)$ -coloring algorithms with running time $\Theta(\log^* n)$. Barenboim and Elkin [4] and, independently, Kuhn [22] generalized this by presenting a tradeoff between the running time and the number of colors: they devised a $\lambda(\Delta + 1)$ -coloring algorithm with running time $O(\Delta/\lambda + \log^* n)$, for any $\lambda \geq 1$. All these algorithms require the knowledge of upper bounds on both n and Δ .

Efficient deterministic algorithms for the edge-coloring problem can be obtained from [5, 7, 32]. The state of the art results are due to Barenboim and Elkin [7]. Specifically, they design an $O(\Delta)$ -edge coloring algorithm running in time $O(\Delta^\epsilon) + \log^* n$, for any $\epsilon > 0$, and an $O(\Delta^{1+\epsilon})$ -edge coloring algorithm running in time $O(\log \Delta) + \log^* n$, for any $\epsilon > 0$. Both these algorithms require the knowledge of upper bounds on both n and Δ .

Randomized algorithms for MIS and $(\Delta + 1)$ -coloring running in expected time $O(\log n)$ were initially given by Luby [29] and, independently, by Alon *et al.* [1]. Recently, Schneider and Wattenhofer [35] constructed the best known non-uniform $(\Delta + 1)$ -coloring algorithm, which runs in time $O(\log \Delta + \sqrt{\log n})$. They also provide random algorithms for coloring using more colors. For integral $c > 0$, a randomized algorithm for $(2, 2(c+1))$ -ruling-set in time $O(2^c \log^{1/c} n)$ is also presented. All these algorithms in [35] are non-uniform and require the knowledge of an upper bound on n .

Maximal Matching: Schneider and Wattenhofer [36] designed a uniform deterministic maximal matching algorithm on bounded-independence graphs running in $O(\log^* n)$ time. For general graphs, however, the state of the art maximal matching algorithm is non-uniform: Hanckowiak *et al.* [19] presented a non-uniform deterministic algorithm for maximal matching running in time $O(\log^4 n)$. This algorithm assumes the knowledge of an upper bound for n (for some parts of the algorithm, the nodes can ignore n provided they know Δ).

1.3 Our results

The main conceptual contribution of the paper is the introduction of a new type of algorithms called *pruning algorithms*. Informally, the fundamental property of this type of algorithms is to allow one to iteratively run a sequence of algorithms (whose output may not necessarily be correct everywhere) so that the global output does not deteriorate, and it always progresses toward a solution.

Our main application for pruning algorithm concerns the problem of locally computing a global solution while minimizing the necessary global knowledge known to nodes. Addressing this, we provide a rather general method for transforming a non-uniform local algorithm into a uniform one without increasing the asymptotic running time of the original algorithm. Our method applies to a wide family of both deterministic and randomized algorithms; in particular, it applies to many of the best known results concerning classical problems such as MIS, Coloring, and Maximal Matching. (See table 1.2 for a summary of some of the uniform algorithms we obtain and the corresponding state of the art existing non-uniform algorithms.)

In another application, we show how to transform a Monte-Carlo local algorithm into a Las Vegas one. Finally, given several uniform algorithms for the same problem whose running times depend on different parameters—which are unknown to nodes—we show a general method for constructing a uniform algorithm solving the problem, that on every instance runs asymptotically as fast as the fastest algorithm among those given algorithms. In particular, we obtain the following theorems.

THEOREM 1. *There exists a uniform deterministic algorithm solving MIS on general graphs in time*

$$\min \left\{ O(\Delta + \log^* n), 2^{O(\sqrt{\log n})}, f(a) \right\},$$

where $f(a) := o(\log n)$ for graphs of arboricity $a = o(\sqrt{\log n})$, and $f(a) := O(\log n / \log \log n)$ for arboricity $O(\log^{1/2-\delta} n)$, for some constant $\delta \in (0, 1/2)$ (otherwise, $f(a) := n$).

THEOREM 2. *There exists a uniform deterministic algorithm solving $(\Delta + 1)$ -coloring problem on general graphs in time $\min\{O(\Delta + \log^* n), 2^{O(\sqrt{\log n})}\}$.*

THEOREM 3. *There exists a uniform deterministic algorithm solving the $\lambda(\Delta+1)$ -coloring problem on general graphs and running in time $O(\Delta/\lambda + \log^* n)$, for any $\lambda \geq 1$, such that Δ/λ is either a constant or a moderately increasing function. In particular, there exists a uniform deterministic algorithm solving the $O(\Delta^2)$ -coloring problem in time $O(\log^* n)$.*

THEOREM 4. (1) *There exists a uniform deterministic $O(\Delta)$ -edge coloring algorithm for general graphs running in time $O(\Delta^\epsilon + \log^* n)$, for any $\epsilon > 0$.* (2) *There exists a uniform deterministic $O(\Delta^{1+\epsilon})$ -edge coloring algorithm for general graphs that runs in time $O(\log \Delta + \log^* n)$, for any $\epsilon > 0$.*

THEOREM 5. *There exists a uniform deterministic algorithm solving the maximal matching problem in time $O(\log^4 n)$.*

THEOREM 6. *For a constant integral $c > 0$, there exists a uniform randomized algorithm solving the $(2, 2(c+1))$ -ruling-set problem in time $O(2^c \log^{1/c} n)$.*

2. PRELIMINARIES

General definitions: For two integers a and b , we let $[a, b] := \{a, a+1, \dots, b\}$. Let $G = (V(G), E(G))$ be an undirected and unweighted graph. The *degree* $\deg_G(v)$ of a node $v \in V(G)$ is the number of neighbors of v in G . The *maximum degree* of G is $\Delta_G := \max \{\deg_G(v) \mid v \in V(G)\}$. The *distance* $\text{dist}_G(u, v)$ between two nodes $u, v \in G$ is the number of edges on a shortest path connecting them. Given a node u and an integer r , the *ball* of radius r around u is the subgraph $B_G(u, r)$ of G induced by the collection of nodes at distance at most r from u . The *neighborhood* $N_G(u)$ of u is the set of neighbors of u , i.e., $N_G(u) := B_G(u, 1) \setminus \{u\}$. In what follows, we may omit the subscript G from the previous notations when there is no risk of confusion.

Functions: Fix an integer k . A function $f : \mathbf{R}^k \rightarrow \mathbf{R}$ is *non-decreasing* if $f(x_1, x_2, \dots, x_k) \leq f(y_1, y_2, \dots, y_k)$ for any two sequences (x_1, x_2, \dots, x_k) and (y_1, y_2, \dots, y_k) where $x_i \leq y_i$ for each $i \in [1, k]$. An *ascending* function is a non-decreasing and unbounded function $f : \mathbf{R} \rightarrow \mathbf{R}^+$ (by unbounded, we mean that $\lim_{x \rightarrow \infty} f(x) = \infty$). A function $f : (\mathbf{R}^+)^{\ell} \rightarrow \mathbf{R}^+$ is called *additive* if $f(x_1, \dots, x_{\ell}) = \sum_{i=1}^{\ell} f_i(x_i)$ and f_i is ascending for each $i \in [1, \ell]$.

A function $f : \mathbf{R}^+ \rightarrow \mathbf{R}^+$ is *moderately-increasing* if it is increasing and there exists a positive integer α such that $f(\alpha i) > 2f(i)$ and $\alpha f(i) > f(2i)$ for every integer $i \geq 2$. Note that $f(x) = x^{k_1} \log^{k_2}(x)$ is a moderately-increasing function for every non-negative constants k_1 and k_2 (such that $k_1 + k_2 > 0$).

Problems and instances: Given a set of nodes V , a *vector* for V is an assignment \mathbf{x} of a bit string $\mathbf{x}(v)$ to each $v \in V$, i.e., \mathbf{x} is a function $\mathbf{x} : V \rightarrow \{0, 1\}^*$. A *problem* is defined by a collection of triplets: $\Pi = \{(G, \mathbf{x}, \mathbf{y})\}$, where $G = (V, E)$ is a (not necessarily connected) graph, and \mathbf{x} and \mathbf{y} are *input* and *output* vectors for V , respectively. We consider only problems that are closed under disjoint union, i.e., if G_1 and G_2 are two vertex disjoint graphs and $(G_1, \mathbf{x}_1, \mathbf{y}_1), (G_2, \mathbf{x}_2, \mathbf{y}_2) \in \Pi$ then $(G, \mathbf{x}, \mathbf{y}) \in \Pi$, where $G = G_1 \cup G_2$, $\mathbf{x} = \mathbf{x}_1 \cup \mathbf{x}_2$ and $\mathbf{y} = \mathbf{y}_1 \cup \mathbf{y}_2$. An *instance*, with respect to a given a problem Π , is a pair (G, \mathbf{x}) for which there exists an output vector \mathbf{y} satisfying $(G, \mathbf{x}, \mathbf{y}) \in \Pi$. In what follows, whenever we consider some collection \mathcal{F} of instances, we always assume that \mathcal{F} is closed under inclusion. That is, if $(G, \mathbf{x}) \in \mathcal{F}$ and $(G', \mathbf{x}') \subseteq (G, \mathbf{x})$ (i.e., G' is a subgraph of G and \mathbf{x}' is the input vector \mathbf{x} restricted to $V(G')$) then $(G', \mathbf{x}') \in \mathcal{F}$. Informally, given a problem Π and a collection of instances \mathcal{F} , the goal is to design an efficient distributed algorithm that takes an instance $(G, \mathbf{x}) \in \mathcal{F}$ as input, and produces an output vector \mathbf{y} satisfying $(G, \mathbf{x}, \mathbf{y}) \in \Pi$. The reason why we require Π to be closed

under disjoint union is that a distributed algorithm operating on an instance (G, \mathbf{x}) behaves separately and independently on each connected component of G . Let \mathcal{G} be a family of graphs closed under inclusion. We define $\mathcal{F}(\mathcal{G})$ to be the set of pairs $\{(G, \mathbf{x}) \mid G \in \mathcal{G}, \mathbf{x} \text{ is arbitrary}\}$.

We assume that each node $v \in V$ is provided with a unique integer referred to as the *identity* of v , and denoted $\text{Id}(v)$, encoded using $O(\log |V|)$ bits; by unique identities, we mean that $\text{Id}(u) \neq \text{Id}(v)$ for every two distinct nodes u and v . (In some works on coloring, the assumption that the identities are unique is relaxed and the collection of assigned identities is only required to be a coloring, i.e., every two neighboring nodes have different identities.) For ease of exposition, we consider the identity of a node to be part of its input.

We consider classical problems such as coloring, (α, β) -ruling set (and in particular MIS, which is $(2, 1)$ -ruling set) and maximal matching. Informally, viewing the output of a node as a *color*, the requirement of *coloring* is that the colors of two neighboring nodes must be different. In *MIS* (for Maximal Independent Set), the output at each node is Boolean, and indicates whether the node belongs to a set S that must form a MIS, that is, S must be independent: no two neighboring nodes are in S , and must be maximal: if all neighbors of a node v are not in S then $v \in S$. In (α, β) -ruling set, the set S of selected nodes must satisfy: (1) two nodes that belong to S must be at distance at least α from each other, and (2) if a node does not belong to S , then there is a node in the set at distance at most β from it. (Observe that MIS is precisely $(2, 1)$ -ruling set.) Finally, given a triplet $(G, \mathbf{x}, \mathbf{y})$, two nodes u and v are called *matched* if $(u, v) \in E$, $\mathbf{y}(u) = \mathbf{y}(v)$ and $\mathbf{y}(w) \neq \mathbf{y}(u)$ for every $w \in (N_G(u) \cup N_G(v)) \setminus \{u, v\}$. In *MM* (for Maximal Matching) it is required that each node u is either matched to one of its neighbors or that every neighbor v of u is matched to one of v 's neighbors.

Parameters: Fix a problem Π and let \mathcal{F} be a collection of instances for Π . A *parameter* \mathbf{p} is a non-decreasing positive valued function $\mathbf{p} : \mathcal{F} \rightarrow \mathbf{N}$. By non-decreasing, we mean that if $(G', \mathbf{x}') \in \mathcal{F}$ and $(G', \mathbf{x}') \subseteq (G, \mathbf{x})$ then $\mathbf{p}(G', \mathbf{x}') \leq \mathbf{p}(G, \mathbf{x})$.

Let \mathcal{F} be a collection of instances. A parameter \mathbf{p} for \mathcal{F} is called a *graph-parameter* if \mathbf{p} is oblivious of the input, that is, if $\mathbf{p}(G, \mathbf{x}) = \mathbf{p}(G, \mathbf{x}')$ for every two instances $(G, \mathbf{x}), (G, \mathbf{x}') \in \mathcal{F}$ such that the input assignments \mathbf{x} and \mathbf{x}' preserve the identities, i.e., the inputs $\mathbf{x}(v)$ and $\mathbf{x}'(v)$ contain the same identity $\text{Id}(v)$ for every $v \in V(G)$. For example, in what follows, we will focus on the following graph-parameters: the number n of nodes of the graph G , i.e., $|V(G)|$, the maximum degree $\Delta = \Delta(G)$ of G , i.e., $\max \{\deg_G(u) \mid u \in V(G)\}$, and the arboricity $a = a(G)$ of G , i.e., the least number of edge-disjoint forests whose union is G .

Local algorithms: Consider a problem Π and a collection of instances \mathcal{F} for Π . An algorithm for Π and \mathcal{F} takes as input an instance $(G, \mathbf{x}) \in \mathcal{F}$ and must terminate with an output vector \mathbf{y} such that $(G, \mathbf{x}, \mathbf{y}) \in \Pi$. We consider the *LOCAL* model (cf., [34]). During the execution of a *local* algorithm \mathcal{A} , all processors are woken up simultaneously and computation proceeds in fault-free synchronous rounds, i.e., it occurs in discrete rounds. In each round, every node may send messages of unrestricted size to its neighbors and may perform arbitrary computations on its data. A message that is sent in a round r , arrives to its destination before the next round $r+1$ starts. It must be guaranteed that after a finite number of rounds, each node v terminates with some output

value $\mathbf{y}(v)$. (It is required that a node knows that its output is indeed its final output.) The algorithm \mathcal{A} is *correct* if for every instance $(G, \mathbf{x}) \in \mathcal{F}$, the resulted output vector \mathbf{y} satisfies $(G, \mathbf{x}, \mathbf{y}) \in \Pi$.

Let \mathcal{A} be a local deterministic algorithm for Π and \mathcal{F} . The *running time* of \mathcal{A} over a particular instance $(G, \mathbf{x}) \in \mathcal{F}$, denoted $T_{\mathcal{A}}(G, \mathbf{x})$, is the number of rounds from the beginning of the execution of \mathcal{A} until all nodes terminate. The running time of \mathcal{A} is typically evaluated with respect to a collection of parameters $\Lambda = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_\ell\}$. Specifically, it is compared to a function $f: \mathbf{N}^\ell \rightarrow \mathbf{R}^+$; we say that f is an upper bound for the running time of \mathcal{A} with respect to Λ if $T_{\mathcal{A}}(G, \mathbf{x}) \leq f(\mathbf{q}_1^*, \mathbf{q}_2^*, \dots, \mathbf{q}_\ell^*)$ for every instance $(G, \mathbf{x}) \in \mathcal{F}$ with parameters $\mathbf{p}_i^* := \mathbf{p}_i(G, \mathbf{x})$ for $i \in [1, \ell]$.

A remark about running an algorithm after another: Many *LOCAL* algorithms happen to have different termination times at different nodes. On the other hand, most of the algorithms rely on a simultaneous wake up of all nodes. This becomes a problem when one wants to run an algorithm \mathcal{A} and subsequently an algorithm \mathcal{B} taking the output of \mathcal{A} as input. Indeed, this problem amounts to running \mathcal{B} with non-simultaneous wake up: a node u starts \mathcal{B} when it terminates \mathcal{A} .

As observed (e.g., by Kuhn [22]), one can use a synchronizer [2] to run a synchronous local algorithm in an asynchronous system, with the same asymptotic time complexity. Hence, the synchronicity assumption can actually be removed. Although the standard asynchronous model introduced still assumes simultaneous wake up, it can be easily verified that the technique still applies with non-simultaneous wake up times if a node can buffer messages received before it wakes up, which is the case when running an algorithm after another. However, we have to adapt the notion of running time. We define it as the number of time units elapsed between the last wake up time of a node and the last termination time of a node. We let $\mathcal{A}; \mathcal{B}$ be the process of running \mathcal{B} after \mathcal{A} . Note that the running time of $\mathcal{A}; \mathcal{B}$ is bounded by the sum of the running times of \mathcal{A} and \mathcal{B} . In the sequel we implicitly assume that the simple α synchronizer is used when running a sequence $\mathcal{A}_1; \mathcal{A}_2; \dots; \mathcal{A}_k$ of algorithms.

Local algorithms requiring parameters: Fix a problem Π and let \mathcal{F} be a collection of instances for Π . Let $\Gamma = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r\}$ be a collection of parameters and let \mathcal{A} be a local algorithm. We say that \mathcal{A} *requires* Γ if, in order to execute \mathcal{A} on an instance $(G, \mathbf{x}) \in \mathcal{F}$, all nodes must agree on a value $\tilde{\mathbf{p}}$ for each parameter $\mathbf{p} \in \Gamma$. The value $\tilde{\mathbf{p}}$ is called a *guess* for \mathbf{p} . A collection of guesses for the parameters in Γ is denoted by $\tilde{\Gamma}$. An algorithm \mathcal{A} that requires Γ is denoted by \mathcal{A}^Γ . An algorithm that does not require any parameter is called *uniform*.

Consider an instance $(G, \mathbf{x}) \in \mathcal{F}$, a collection Γ of parameters and a parameter $\mathbf{p} \in \Gamma$. A guess $\tilde{\mathbf{p}}$ for \mathbf{p} is termed *good* if $\tilde{\mathbf{p}} \geq \mathbf{p}(G, \mathbf{x})$, and the guess $\tilde{\mathbf{p}}$ is called *correct* if $\tilde{\mathbf{p}} = \mathbf{p}(G, \mathbf{x})$. We typically write correct guesses and collection of correct guesses with a star exponent, that is \mathbf{p}^* and $\Gamma^*(G, \mathbf{x})$, respectively. When (G, \mathbf{x}) is clear from the context, for simplicity, we may use the notation Γ^* instead of $\Gamma^*(G, \mathbf{x})$.

An algorithm \mathcal{A}^Γ *depends* on Γ if for every instance $(G, \mathbf{x}) \in \mathcal{F}$, the correctness of \mathcal{A}^Γ over (G, \mathbf{x}) is guaranteed when \mathcal{A}^Γ uses a collection $\tilde{\Gamma}$ of good guesses.

Consider an algorithm \mathcal{A}^Γ that depends on a collection of parameters $\Gamma = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r\}$ and fix an instance (G, \mathbf{x}) .

Observe that the running time of \mathcal{A}^Γ over (G, \mathbf{x}) may be different for different collections of guesses $\tilde{\Gamma}$, in other words, the running time over (G, \mathbf{x}) is a function of $\tilde{\Gamma}$. Recall that when we consider an algorithm that does not require parameters, we still typically evaluate its running time with respect to a collection of parameters Λ . We generalize this to the case where the algorithm depends on Γ as follows.

Consider two collections of parameters $\Gamma = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r\}$ and $\Lambda = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_\ell\}$. Some parameters may belong to both Γ and Λ . Without loss of generality, we shall always assume that $\{\mathbf{p}_{r'+1}, \mathbf{p}_{r'+2}, \dots, \mathbf{p}_r\} \cap \{\mathbf{q}_{r'+1}, \mathbf{q}_{r'+2}, \dots, \mathbf{q}_\ell\} = \emptyset$ for some $r' \in [0, \min\{r, \ell\}]$ and $\mathbf{p}_i = \mathbf{q}_i$ for every $i \in [1, r']$. Notice that $\Gamma \setminus (\Gamma \cap \Lambda) = \{\mathbf{p}_{r'+1}, \mathbf{p}_{r'+2}, \dots, \mathbf{p}_r\}$. A function $f: (\mathbf{R}^+)^{\ell} \rightarrow \mathbf{R}^+$ *upper bounds* the running time of \mathcal{A}^Γ with respect to Γ and Λ if the running time $T_{\mathcal{A}^\Gamma}(G, \mathbf{x})$ of \mathcal{A}^Γ for $(G, \mathbf{x}) \in \mathcal{F}$ using a collection of good guesses $\tilde{\Gamma} = \{\tilde{\mathbf{p}}_1, \tilde{\mathbf{p}}_2, \dots, \tilde{\mathbf{p}}_r\}$ is at most $f(\tilde{\mathbf{p}}_1, \tilde{\mathbf{p}}_2, \dots, \tilde{\mathbf{p}}_{r'}, \mathbf{q}_{r'+1}^*, \dots, \mathbf{q}_\ell^*)$, where $\mathbf{q}_i^* = \mathbf{q}_i(G, \mathbf{x})$ for $i \in [r', \ell]$. Note that we do not put any restriction on the running time of \mathcal{A}^Γ over (G, \mathbf{x}) if some of the guesses in $\tilde{\Gamma}$ are not good. In fact, in such a case, the algorithm may not even terminate.

For simplicity of presentation, in this extended abstract, we always assume that the function $f: (\mathbf{R}^+)^{\ell} \rightarrow \mathbf{R}^+$ is additive. In the full paper we show how to obtain similar results for other types of functions; we note that this extension may sometimes require overheads in the running time.

For simplicity of notation, when Γ and Λ are clear from the context, we say that f upper bounds the running time of \mathcal{A}^Γ , without writing that it is with respect to Γ and Λ .

The set Γ is *weakly-dominated* by Λ if for each $j \in [r'+1, r]$, there exists an index $i_j \in [1, \ell]$ and an ascending function g_j such that $g_j(\mathbf{p}_j(G, \mathbf{x})) \leq \mathbf{q}_{i_j}(G, \mathbf{x})$ for every instance $(G, \mathbf{x}) \in \mathcal{F}$. (For example, $\Gamma = \{\Delta\}$ is weakly-dominated by $\{\Lambda\} = n$, since $\Delta(G, \mathbf{x}) \leq n(G, \mathbf{x})$ for any (G, \mathbf{x}) .)

3. PRUNING ALGORITHMS

Consider a problem Π in the centralized setting and an efficient randomized Monte-Carlo algorithm \mathcal{A} for Π . A known method for transforming \mathcal{A} into a Las Vegas algorithm is based on repeatedly doing the following. Execute \mathcal{A} and, subsequently, execute an algorithm that checks the validity of the output. If the checking fails then continue, and otherwise, terminate, i.e., break the loop. This transformation can yield a Las Vegas algorithm whose expected running time is similar to the running time of the Monte-Carlo algorithm provided that the checking mechanism used is efficient.

If we wish to come up with a similar transformation in the context of locality, a first idea would be to consider a local algorithm that checks the validity of a tentative output vector. This concept has been studied in several forms (e.g., [16, 21, 31]). However, such fast local checking procedures can only guarantee that faults are detected by at least one node, whereas to restart the Monte-Carlo algorithm, all nodes should be aware of a fault. This notification can take diameter time and will thus violate the locality constraint.

Instead of using local checking procedures, we introduce the notion of *pruning algorithms*. Informally, this is a mechanism that identifies “valid areas” where the tentative output vector $\hat{\mathbf{y}}$ is valid and *prunes* these areas, i.e., takes them out of further consideration. A pruning algorithm \mathcal{P} must satisfy two properties, specifically, (1) *gluing*: \mathcal{P} must make sure that the current solution on these “pruned areas” can be

extended to a valid solution for the remainder of the graph, and (2) *solution detection*: if $\hat{\mathbf{y}}$ was a valid global solution to begin with then \mathcal{P} should prune all nodes.

Now, given a Monte-Carlo algorithm \mathcal{A} and a pruning algorithm \mathcal{P} for the problem, we can transform \mathcal{A} into a Las Vegas algorithm by iteratively executing the pair of algorithms $(\mathcal{A}; \mathcal{P})$ in iterations, where each iteration i is executed on the graph G_i induced by the set of nodes that were not pruned in previous iterations (G_1 is the initial graph G). If, in some iteration i , Algorithm \mathcal{A} solves the problem on the graph G_i , then the solution detection property guarantees that the subsequent pruning algorithm will prune all nodes in G_i and hence at that time all nodes are pruned and the execution terminates. Furthermore, using induction, it can be shown that the gluing property guarantees that the correct solution to G_i combined with the output of previously pruned nodes forms a global solution for G .

We now formally define pruning algorithms. Fix a problem Π and a family of instances \mathcal{F} for Π . A *pruning algorithm* \mathcal{P} for Π and \mathcal{F} takes as input a triplet $(G, \mathbf{x}, \hat{\mathbf{y}})$, where $(G, \mathbf{x}) \in \mathcal{F}$ and $\hat{\mathbf{y}}$ is some tentative output vector, and returns a bit $b(v)$ and an updated input $\mathbf{x}'(v)$ at each node v . The bit $b(v)$ indicates whether v belongs to some selected subset $W \subseteq V(G)$ of nodes to be pruned. (Recall that the idea is to assume that nodes in W have a satisfying tentative output value and that they can be excluded from further computations.) Let G' be the subgraph of G induced by the nodes in $V(G) \setminus W$. The pruning algorithm does not change the input of the nodes in W , i.e., $\mathbf{x}'(v) = \mathbf{x}(v)$ whenever $b(v) = 1$ (that is, $v \in W$). On the other hand, the pruning algorithm may change the input for the rest of the nodes, i.e., those in G' . (Informally, this is because after \mathcal{P} pruned the set W , it may need to adjust the remaining nodes for further use.) Thus, Algorithm \mathcal{P} takes a triplet $(G, \mathbf{x}, \hat{\mathbf{y}})$ as input, where $(G, \mathbf{x}) \in \mathcal{F}$, and returns a pair (G', \mathbf{x}') . The pruning algorithm must guarantee that $(G', \mathbf{x}') \in \mathcal{F}$.

Consider now an output vector \mathbf{y}' for the nodes in $V(G')$. The *combined* output vector \mathbf{y} of the vectors $\hat{\mathbf{y}}$ and \mathbf{y}' is the output vector that is a combination of $\hat{\mathbf{y}}$ restricted to the nodes in W and \mathbf{y}' restricted to the nodes in G' , i.e., $\mathbf{y}(v) := \hat{\mathbf{y}}(v)$ if $v \in W$ and $\mathbf{y}(v) := \mathbf{y}'(v)$ otherwise. A pruning algorithm \mathcal{P} for a problem Π must satisfy the following properties.

- **Solution detection:** $(G, \mathbf{x}, \hat{\mathbf{y}}) \in \Pi \iff \mathcal{P}$ outputs $W = V(G)$.
- **Gluing:** if $\mathcal{P}(G, \mathbf{x}, \hat{\mathbf{y}}) = (G', \mathbf{x}')$ and \mathbf{y}' is a solution for (G', \mathbf{x}') , i.e., $(G', \mathbf{x}', \mathbf{y}') \in \Pi$, then the combined output vector \mathbf{y} is a solution for (G, \mathbf{x}) , i.e., $(G, \mathbf{x}, \mathbf{y}) \in \Pi$.

The pruning algorithm \mathcal{P} is *monotone with respect to a parameter* \mathbf{p} if $\mathbf{p}(G, \mathbf{x}) \geq \mathbf{p}(G', \mathbf{x}')$, for every $(G, \mathbf{x}) \in \mathcal{F}$, where $\mathcal{P}(G, \mathbf{x}, \hat{\mathbf{y}}) = (G', \mathbf{x}')$, for some $\hat{\mathbf{y}}$. The pruning algorithm \mathcal{P} is *monotone with respect to a collection of parameters* Γ if \mathcal{P} is monotone with respect to every parameter $\mathbf{p} \in \Gamma$. In such a case, we may also say that \mathcal{P} is Γ -*monotone*. The following assertion follows from the definition of a parameter.

Observation 3.1 *Let \mathcal{P} be a pruning algorithm. Then (1) Algorithm \mathcal{P} is monotone with respect to any graph-parameter, and (2) If \mathcal{P} does not update the inputs of the unpruned nodes, i.e., $\mathbf{x}'(v) = \mathbf{x}(v)$ for every $v \in V \setminus W$, then \mathcal{P} is monotone with respect to any parameter.*

For simplicity, in this extended abstract, we restrict the running time of a pruning algorithm \mathcal{P} to be constant. In

the full paper, we consider the more general case where \mathcal{P} is a uniform algorithm whose running time may depend on some parameters. We note that this generalization may incur an overhead in the running time of our transformations, as these repeatedly use \mathcal{P} .

We now give examples of (constant time) pruning algorithms for several problems, namely, $(2, \beta)$ -Ruling set for a constant integer β (recall that MIS is precisely $(2, 1)$ -Ruling set), and maximal matching. These pruning algorithms do not change the input at nodes outside W (in fact, the input is ignored in these problems, and can be assumed to be empty). Thus, by Observation 3.1, they are monotone with respect to any parameter.

The $(2, \beta)$ -ruling set pruning algorithm: Let β be a (constant) integer. We define a pruning algorithm $\mathcal{P}_{(2, \beta)}$ for the $(2, \beta)$ -ruling set problem as follows. Given a triplet $(G, \mathbf{x}, \hat{\mathbf{y}})$, let W be the set of nodes u satisfying one of the following two conditions.

- $\hat{\mathbf{y}}(u) = 1$ and $\hat{\mathbf{y}}(v) = 0$ for all $v \in N(u)$, or
- $\hat{\mathbf{y}}(u) = 0$ and $\exists v \in B_G(u, \beta)$ such that $\hat{\mathbf{y}}(v) = 1$ and $\hat{\mathbf{y}}(w) = 0$ for all $w \in N(v)$.

Observation 3.2 *Algorithm $\mathcal{P}_{(2, \beta)}$ is a pruning algorithm for the $(2, \beta)$ -ruling set problem, running in time $1 + \beta$. Furthermore, $\mathcal{P}_{(2, \beta)}$ is monotone with respect to any parameter.*

The interested reader can check that this pruning algorithm is not implementable through simple combinations of the classical local check procedure for $(2, \beta)$ -ruling set even though it has a similar flavor.

The maximal matching problem: We define a pruning algorithm \mathcal{P}_{MM} as follows. Given a tentative output vector $\hat{\mathbf{y}}$, recall that u and v are matched when u and v are neighbors, $\hat{\mathbf{y}}(u) = \hat{\mathbf{y}}(v)$ and $\hat{\mathbf{y}}(w) \neq \hat{\mathbf{y}}(u)$ for every $w \in (N_G(u) \cup N_G(v)) \setminus \{u, v\}$. Set W to be the set of nodes u satisfying one of the following conditions.

- $\exists v \in N(u)$ such that u and v are matched, or
- $\forall v \in N(u)$, $\exists w \neq u$ such that v and w are matched.

Observation 3.3 *Algorithm \mathcal{P}_{MM} is a pruning algorithm for MM whose running time is 3. Furthermore, \mathcal{P}_{MM} is monotone with respect to any parameter.*

4. A GENERAL METHOD

We now turn to the main application of pruning algorithms discussed in this paper, that is, the construction of a transformer taking a non-uniform algorithm A^Γ as a black box and producing a uniform one that enjoys the same (asymptotic) time complexity as the original non-uniform algorithm.

The basic idea is very simple. Consider a problem for which we have a pruning algorithm \mathcal{P} , and a non uniform algorithm \mathcal{A} that requires the knowledge of upper bounds on some parameters. To obtain a uniform algorithm, we execute the pair of algorithms $(\mathcal{A}; \mathcal{P})$ in iterations, where each iteration executes \mathcal{A} using a specific set of guesses for the parameters. Typically, as iterations proceed, the guesses for the parameters grow larger and larger until we reach an iteration i where all guesses are larger than the actual value of the parameters. In this iteration, the operation of \mathcal{A} on G_i using such guesses guarantees a correct solution on G_i . As mentioned before, the properties of the pruning algorithm guarantee that the execution terminates in this iteration and that at that time, the output of all nodes combines to a

global solution on G . To bound the running time, we will make sure that the total running time is dominated by the running time of the last iteration, and that this last iteration is relatively fast.

There are various delicate points when using this general method. For example, in iterations where incorrect guesses are used, we have no control over the behavior of the non-uniform algorithm \mathcal{A} and, in particular, it may run for too many rounds, perhaps even indefinitely. To overcome this obstacle, we allocate a prescribed number of rounds for each iteration; if Algorithm \mathcal{A} reaches this time bound without outputting at some node u , then we force it to terminate with an arbitrary output. Subsequently, we run the pruning algorithm and proceed to the next iteration.

Obviously, this simple approach of running in iterations and increasing the guesses from iteration to iteration is hardly new. It was used, for example, in the context of wireless networks to compute estimates of parameters (cf., e.g., [8, 30]). One of the main contributions of this current paper is the formalization and generalization of this technique, allowing it to be used for a wide varieties of problems and applications. Interestingly, note that we are only concerned with getting rid of the knowledge of some parameters, and not with obtaining estimates for them (in particular, when our algorithms terminate, the vertices have no way of knowing whether they have upper bounds on these parameters).

To illustrate the method, let us consider the non-uniform MIS algorithm of Panconesi and Srinivasan [33]. This algorithm \mathcal{A} assumes the knowledge of an upper bound \tilde{n} on the number of nodes n , and runs in time at most $f(\tilde{n}) = 2^{O(\sqrt{\log \tilde{n}})}$. Consider a pruning algorithm \mathcal{P}_{MIS} for MIS (such an algorithm is given by Observation 3.2). The following sketches our method for obtaining a uniform MIS algorithm. For each integer i , let n_i be the largest integer a such that $f(a) \leq 2^i$. In Iteration i , for $i = 1, 2, \dots$, we first execute Algorithm \mathcal{A} using the guess n_i (as an input serving as an upper bound for the number of nodes) for precisely 2^i rounds. Subsequently, we run the pruning algorithm \mathcal{P}_{MIS} . When the pruning algorithm terminates, we execute the next iteration $i + 1$ on the non-pruned nodes. Let s be the integer such that $2^{s-1} < f(n) \leq 2^s$, where n is the number of nodes of the input graph. By the definition, $n \leq n_s$. Therefore, the application of \mathcal{A} in Iteration s uses a guess n_s that is indeed larger than the number of nodes. Moreover, this execution of \mathcal{A} is completed before the prescribed deadline of 2^s expires because its running time is at most $f(n_s) \leq 2^s$. Hence, we are guaranteed to have a correct solution by the end of Iteration s . The running time is thus at most $\sum_{i=1}^s 2^i = O(f(n))$.

This method can sometimes be extended to simultaneously remove prior knowledge concerning several parameters. For example, consider the MIS algorithm of Barenboim and Elkin [4] (or that of Kuhn [22]), which requires the knowledge of upper bounds \tilde{n} and $\tilde{\Delta}$ on n and Δ , respectively, and runs in time $f(\tilde{n}, \tilde{\Delta}) = f_1(\tilde{n}) + f_2(\tilde{\Delta})$, where $f_1(\tilde{\Delta}) = O(\tilde{\Delta})$ and $f_2(\tilde{n}) = O(\log^* \tilde{n})$. The following sketches our method for obtaining a corresponding uniform MIS algorithm that runs in time $O(f(n, \Delta))$. For each integer i , let n_i (respectively Δ_i) be the largest integer a (respectively, b) such that $f_1(a) \leq 2^i$ (respectively, $f_2(b) \leq 2^i$). In Iteration i , for $i = 1, 2, \dots$, we first execute Algorithm \mathcal{A} using the guesses n_i and Δ_i , but this time the execution lasts for precisely $2 \cdot 2^i$ rounds. (The factor 2 in the running time of an iteration follows from the fact that we consider two parameters here, namely n and

Δ . In general, if we consider r parameters, Iteration i will run for $r \cdot 2^i$ rounds.) Subsequently, we run the pruning algorithm \mathcal{P}_{MIS} , and as before, when the pruning algorithm terminates, we execute the next iteration $i + 1$ on the non-pruned nodes. Now, let s be the integer such that $2^{s-1} < f(n, \Delta) \leq 2^s$. By the definition, $n \leq n_s$ and $\Delta \leq \Delta_s$. Hence, the application of \mathcal{A} in Iteration s uses guesses that are indeed upper bounds on the real values. This execution of \mathcal{A} is completed before the prescribed deadline of 2^{s+1} expires because its running time is at most $f_1(n_s) + f_2(\Delta_s) \leq 2^{s+1}$. Thus, the algorithm consists of at most s iterations. Since the running time of the whole execution is dominated by the running time of the last iteration, the total running time is $O(2^{s+1}) = O(f(n, \Delta))$.

The following theorem formalizes the above discussion. It considers a single set of parameters $\Gamma = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r\}$, and assumes that the given non-uniform algorithm \mathcal{A}^Γ both depends on Γ as well as that its running time is evaluated according to the parameters in Γ . Recall that in such a case, we say that a function $f : \mathbf{N}^r \rightarrow \mathbf{R}^+$ upper bounds the running time of \mathcal{A}^Γ with respect to Γ if the running time $T_{\mathcal{A}^\Gamma}(G, \mathbf{x})$ of \mathcal{A}^Γ for every $(G, \mathbf{x}) \in \mathcal{F}$ using a collection of good guesses $\tilde{\Gamma} = \{\tilde{\mathbf{p}}_1, \tilde{\mathbf{p}}_2, \dots, \tilde{\mathbf{p}}_r\}$ for (G, \mathbf{x}) is at most $f(\tilde{\mathbf{p}}_1, \tilde{\mathbf{p}}_2, \dots, \tilde{\mathbf{q}}_r)$. Recall also that this extended abstract focuses on the case that the function f is additive. The proof of the theorem below follows using similar arguments to the ones discussed in the aforementioned examples and is therefore omitted.

THEOREM 7. *Consider a problem Π and a family of instances \mathcal{F} . Let \mathcal{A}^Γ be a deterministic algorithm for Π and \mathcal{F} depending on $\Gamma := \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_r\}$. Suppose that the running time of \mathcal{A}^Γ is upper bounded by an additive function $f := \sum_{i=1}^r f_i(\mathbf{p}_i)$. Assume there exists a Γ -monotone pruning algorithm \mathcal{P} for Π and \mathcal{F} . Then there exists a uniform deterministic algorithm for Π and \mathcal{F} whose running time is $O(f(\Gamma^*)) = O(\sum_{i=1}^r f_i(\mathbf{p}_i^*))$.*

Theorem 5 follows directly by applying Theorem 7 to the maximal matching algorithm of Hanckowiak *et al.* [19], and using Observation 3.3. In addition, using Observation 3.2, Theorem 7 allows us to transform each of the MIS algorithms in [4, 22, 33] into a uniform one with asymptotically the same time complexity. That is, we obtain the following.

COROLLARY 1. *(1) There exists a uniform deterministic algorithm solving MIS on general graphs in time $O(\Delta + \log^* n)$, and (2) there exists a uniform deterministic algorithm solving MIS on general graphs in time $2^{O(\sqrt{\log n})}$.*

Some complications arise when the correctness of the given non-uniform problem depends on the knowledge of one set of parameters Γ , while its running time is evaluated by another set of parameters Λ . For example, it may be the case that an upper bound on a parameter \mathbf{p} is required for the correct operation of an algorithm, yet the running time of the algorithm does not depend on \mathbf{p} . In this case, it is not clear how to choose the guesses for \mathbf{p} . (This occurs, for example, in the MIS algorithms of Barenboim and Elkin [6], where the knowledge of n and the arboricity a are required, yet the running time f is a function of n only.) Such complications can be solved when there is some relation between the parameters in Γ and those in Λ ; specifically, when Γ is weakly-dominated by Λ . (See the definition of weakly-dominated in Section 2.)

This issue is handled in the following theorem. Due to the lack of space, the proof is deferred to the full paper.

THEOREM 8. *Consider a problem Π , a family of instances \mathcal{F} and two sets of parameters Γ and $\Lambda = \{q_1, q_2, \dots, q_\ell\}$, where Γ is weakly-dominated by Λ . Let \mathcal{A}^Γ be a deterministic algorithm depending on Γ whose running time is upper bounded by some additive function $f := \sum_{i=1}^\ell f_i(q_i)$. Let \mathcal{P} be a $\Lambda \cup \Gamma$ -monotone pruning algorithm for Π and \mathcal{F} . Then there exists a uniform deterministic algorithm for Π and \mathcal{F} whose running time is $O(f(\Lambda^*)) = O(\sum_{i=1}^\ell f_i(q_i^*))$.*

The following corollary follows by applying the theorem above to the work of [6], by setting $\Gamma = \{a, n\}$ and $\Lambda = \{n\}$.

COROLLARY 2. *There exists a uniform deterministic algorithm solving MIS on general graphs in time $O(f(a))$, where $f(a) = o(\log n)$ for graphs with arboricity $a = o(\sqrt{\log n})$, and $f(a) = O(\log n / \log \log n)$ for graphs with arboricity $a = O(\log^{1/2-\delta} n)$, for some constant $\delta \in (0, 1/2)$ (otherwise, $f(a) = n$).*

To illustrate the topic of the next theorem, consider the best known non-uniform algorithms for MIS, namely the algorithms of Barenboim and Elkin [4] and that of Kuhn [22], which run in time $O(\Delta + \log^* n)$ and require the knowledge of n and Δ , and the algorithm of Panconesi and Srinivasan [33], which runs in time $2^{O(\sqrt{\log n})}$ and requires the knowledge of n . Furthermore, consider the MIS algorithm of Barenboim and Elkin [6], which is very efficient for graphs with bounded arboricity a (let $f'(a)$ be the running time of their algorithm). If n , Δ and a are known to all nodes, then one can compare the running times of these algorithms and use the fastest one. That is, there exists a non-uniform algorithm $\mathcal{A}^{\{n, \Delta, a\}}$ that runs in time $T'(n, \Delta, a) := \min\{2^{O(\sqrt{\log n})}, O(\Delta + \log^* n), f'(a)\}$.

Unfortunately, Theorem 8 does not allow us to transform $\mathcal{A}^{\{n, \Delta, a\}}$ into a uniform one—the reason being that the running time $T(n, \Delta, a)$ does not satisfy the running time requirements as specified in Theorem 8. On the other hand, as mentioned in Corollary 1, Theorem 7 does allow us to transform each of the algorithms in [4, 22, 33] into a uniform one—with asymptotically the same time complexity. Moreover, as mentioned in Corollary 2, we can also transform the algorithm in [6] into a uniform one running in time $f(a)$. Nevertheless, since n , Δ and a are unknown to the nodes, it is not clear how to obtain from these transformed algorithms a uniform algorithm running in time $T(n, \Delta, a) := \min\{2^{O(\sqrt{\log n})}, O(\Delta + \log^* n), f(a)\}$. The following theorem solves this problem.

THEOREM 9. *Consider a problem Π , a family of instances \mathcal{F} . Let Λ_1 and Λ_2 be two sets of parameters. Suppose that there exists a $\Lambda_1 \cup \Lambda_2$ -monotone pruning algorithm \mathcal{P} for Π and \mathcal{F} . Consider two uniform algorithms \mathcal{U}_1 and \mathcal{U}_2 whose running times are bounded by non-decreasing functions $f_1(\Lambda_1^*)$ and $f_2(\Lambda_2^*)$, respectively. Then there is a uniform algorithm with running time $O(f_{\min})$, where $f_{\min} := \min\{f_1(\Lambda_1^*), f_2(\Lambda_2^*)\}$.*

The basic idea behind the proof of theorem above is to run in iterations, such that each iteration i consists of running the quadruple $(\mathcal{U}_1; \mathcal{P}; \mathcal{U}_2; \mathcal{P})$, where \mathcal{U}_1 and \mathcal{U}_2 are executed for precisely 2^i rounds each. Hence, a correct solution will be produced in Iteration $s := \lceil \log f_{\min} \rceil$ or before. Since

each iteration i lasts for roughly 2^{i+1} rounds (recall that the running time of \mathcal{P} is constant), the running time is $O(f_{\min})$. Due to the lack of space we defer the detailed proof to the full paper.

Theorem 1 follows as a direct corollary of Theorem 9, using Corollaries 1 and 2. A standard trick (cf., [27, 29]) allows us to transform an efficient MIS algorithm for general graphs into one for $(\Delta + 1)$ -coloring. This is based on the observation that $(\Delta + 1)$ -colorings of G and MISs of $G' = G \times K_{\Delta+1}$ are in one-to-one correspondence. This known transformation, however, uses the knowledge of Δ . Nevertheless, it is straightforward to check that a similar correspondence holds when G' is defined as follows. For each node $u \in V(G)$, take a clique of size $\deg_G(u) + 1$ with vertices $u_1, u_2, \dots, u_{\deg_G(u)+1}$. Now, for each $(u, v) \in E(G)$ and each $i \in [1, 1 + \min\{\deg_G(u), \deg_G(v)\}]$, let $(u_i, v_i) \in E(G')$. The graph G' can be constructed locally without any global knowledge, hence we obtain Theorem 2 as a corollary of Theorem 1.

In the full paper we show how to extend Theorem 8 to the randomized setting. More specifically, we replace the given deterministic Algorithm \mathcal{A}^Γ in Theorem 8 by a non-uniform Monte-Carlo algorithm A^Γ and produce a uniform Las Vegas one running in the same asymptotic running time as A^Γ . This transformer is more sophisticated than the one given in Theorem 8, and requires the use of sub-iterations for bounding the expected running time and probability of success of the resulting Las-Vegas algorithm. Theorem 6 follows by applying this extended theorem to the ruling-set algorithm of Schneider and Wattenhofer [35], and using the pruning algorithm given by Observation 3.2.

5. MORE COLORING ALGORITHMS

As mentioned, some uniform $(\Delta + 1)$ -coloring algorithms can be obtained from our uniform MIS algorithms using a known technique. In general, however, we could not apply directly the aforementioned transformers to obtain other uniform coloring algorithms. The main reason is that we could not find an efficient pruning algorithm for the coloring problem. Another difficulty in coloring is that, often, the number of colors to be used depends on unknown parameters.

In this section, we present a method for transforming non-uniform algorithms into uniform ones, tailored particularly to the coloring problem. We begin with the following definitions. An *instance for the coloring problem* is a pair (G, \mathbf{x}) where G is a graph and $\mathbf{x}(v)$ contains a color $c(v)$ such that the collection $\{c(v) \mid v \in V(G)\}$ forms a coloring of G . (The color $c(v)$ can be the identity $\text{Id}(v)$.) For a given family of graphs \mathcal{G} , we define $\mathcal{F}(\mathcal{G})$ to be the collection of instances (G, \mathbf{x}) for the coloring problem, where $G \in \mathcal{G}$.

Recall that many coloring algorithms consider the identities as colors, and relax the assumption that the identities are unique by replacing it with the weaker requirement that the set of initial colors forms a coloring. Given an instance (G, \mathbf{x}) , let $m = m(G, \mathbf{x})$ be the maximum integer i such that all identities (initial colors) are taken from $[1, i]$. Without loss of generality, we may assume that $m > \Delta$. Note that m is a graph-parameter.

Recall the $\lambda(\Delta + 1)$ -coloring algorithm of [4, 22] (which generalizes the $O(\Delta^2)$ -coloring algorithm in [27]). We would like to point out that, in fact, everything works similarly in these algorithms if one replaces n with m . That is, the $\lambda(\Delta + 1)$ -coloring algorithms in [4, 22] can be viewed as requiring

m and Δ and running in time $O(\tilde{\Delta}/\lambda + \log^* \tilde{m})$. The same is true for the edge-coloring algorithms of Barenboim and Elkin [7].

The following theorem implies that these algorithms can be transformed into uniform ones. In the theorem, we consider two sets of graph-parameters Γ and Λ such that (1) Γ is weakly-dominated by Λ , and (2) $\Gamma \subseteq \{\Delta, m\}$. Such a pair of sets of parameters is said to be *related*. Also, the function $g(x)$ governing the number of colors is assumed to satisfy (1) $g(x) > x$, (2) $g(x)$ is moderately-increasing, and (3) $g(x)$ is upper bounded by a polynomial in x . Such a function is called *moderately-fast*.

THEOREM 10. *Let \mathcal{A}^Γ be a $g(\tilde{\Delta})$ -coloring algorithm with running time bounded by $f := \sum_{i=1}^{|\Lambda|} f_i$, where Γ and Λ are related collections of graph parameters. If*

1. *$g(x)$ is a moderately-fast function,*
2. *the dependence of f on m is bounded by a polylog, and*
3. *the dependence of f on Δ is moderately-increasing,*

then there exists a uniform $O(g(\Delta))$ -coloring algorithm whose running time is $O(\sum_{i=1}^{|\Lambda|} f_i(q_i^))$.*

Proof sketch: Our first goal is to transform \mathcal{A}^Γ into a uniform algorithm that solves the following problem.

The *strong list-coloring* (SLC) problem: A pair $(G, \mathbf{x}) \in \mathcal{F}(\mathcal{G})$ is a configuration for the SLC problem if the following holds. For every $v \in V(G)$, the input $\mathbf{x}(v)$ contains (in addition to $\text{Id}(v)$) a *degree-estimation* $\hat{\Delta}$, such that $\hat{\Delta} \geq \Delta$ (the estimate $\hat{\Delta}$ being the same for all nodes) and a list $L(v) \subseteq [1, g(\hat{\Delta})] \times [1, \hat{\Delta} + 1]$ of colors, such that

$$\forall k \in [1, g(\hat{\Delta})], \quad |\{j \mid (k, j) \in L(v)\}| \geq \deg_G(v) + 1.$$

Given a configuration $(G, \mathbf{x}) \in \mathcal{F}(\mathcal{G})$, an output vector \mathbf{y} is a *solution* for SLC if it forms a coloring and if $\mathbf{y}(v) \in L(v)$ for every node $v \in V(G)$.

We first transform \mathcal{A}^Γ into an algorithm $B^{\Gamma'}$ that depends on $\Gamma' = \Gamma \setminus \{\Delta\}$ and solves SLC. Specifically, Algorithm $B^{\Gamma'}$ consists of executing \mathcal{A}^Γ using a good guess $\hat{\Delta} = \hat{\Delta}$ for the parameter Δ . Furthermore, if \mathcal{A}^Γ outputs at v a color c , then $B^{\Gamma'}$ outputs the color (c, j) where $j := \min \{s \mid (c, s) \in L(v)\}$. Observe that Algorithm $B^{\Gamma'}$ depends on Γ' and solves SLC.

If $\Delta \in \Lambda$ then set $\Lambda' := \Lambda \setminus \Delta \cup \hat{\Delta}$, and otherwise, $\Lambda' := \Lambda$. ($\hat{\Delta}$ is viewed here as a parameter). It can be shown that the running time of $B^{\Gamma'}$ is bounded by $f(\Lambda')$ (the dependency of f on Δ is replaced by the dependency on $\hat{\Delta}$).

For SLC we can design the following pruning algorithm \mathcal{P} . Consider a triplet $(G, \mathbf{x}, \hat{\mathbf{y}})$, where (G, \mathbf{x}) is a configuration for SLC and $\hat{\mathbf{y}}$ is some tentative assignment of colors. The set W to be pruned is the set of nodes u satisfying $\hat{\mathbf{y}}(u) \in L(u)$ and $\hat{\mathbf{y}}(u) \neq \hat{\mathbf{y}}(v)$ for all $v \in N_G(u)$. Algorithm \mathcal{P} modifies the input for the nodes outside W as follows: the input at a node $u \in V \setminus W$ is changed so that the new list of available colors $L'(u)$ contains precisely $L(u)$ minus the colors assigned to those neighbors of u in G that belong to W . Let (G', \mathbf{x}') be the output of \mathcal{P} . Note that if we start with a configuration (G, \mathbf{x}) for SLC then the output (G', \mathbf{x}') of the pruning algorithm \mathcal{P} is also a configuration for SLC. This is because, for every node v and every k , at most $\deg_W(v)$ pairs (k, j) were removed from the list $L(v)$ of v , where $\deg_W(v)$ is the number of neighbors of v that belong to W . On the other hand, the degree of v in G' is reduced by $\deg_W(v)$.

Assume without loss of generality that f_1 is the function corresponding to Δ in f (i.e., that $q_1 = \Delta$). Having the sets of parameters Γ' and Λ' in mind, Algorithm $B^{\Gamma'}$, and the aforementioned pruning algorithm \mathcal{P} for SLC, we apply Theorem 8 and obtain a uniform algorithm \mathcal{B} for SLC and $\mathcal{F}(\mathcal{G})$ whose running time is $O(f_1(\hat{\Delta}) + \sum_{i=2}^{\ell} f_i(q_i^*))$ if $\Delta \in \Lambda$, and $O(\sum_{i=1}^{\ell} f_i(q_i^*))$ otherwise.

We are now ready to specify a uniform $O(g(\Delta))$ -coloring algorithm. We inductively define integers D_i for $i \in \mathbf{N}$ as follows. $D_1 := 1$, and $D_{i+1} := \min \{\ell \mid g(\ell) \geq 2g(D_i)\}$ for $i \geq 1$. Given an initial configuration (G, \mathbf{x}) , we partition it by node degrees as follows. For $i \in \mathbf{N}$, let G_i be the subgraph of G induced by the set of nodes $v \in G$ with $\deg_G(v) \in [D_i, D_{i+1} - 1]$. Let \mathbf{x}_i be the input \mathbf{x} restricted to the nodes in G_i . The configuration $(G_i, \mathbf{x}_i) \in \mathcal{F}(\mathcal{G})$ is referred to as *layer i* . Note that nodes can figure out locally which layer they belong to. Observe also, that $D_{i+1} - 1$ is an upper bound on node degrees in layer i .

The algorithm proceeds in two phases. In the first phase, each node in layer i is assigned the list of colors $C'_i := [1, g(D_{i+1})] \times [1, D_{i+1} + 1]$, and the degree estimation $\hat{\Delta}_i := D_{i+1}$. Each layer is now an instance of SLC and we execute Algorithm \mathcal{B} in parallel on all layers. If Algorithm \mathcal{B} assigns a color (c, j) to a node v in layer i then we change this color to $(g(D_{i+1}) + c, j)$. Hence, each layer i is colored with colors taken from $C'_i := [g(D_{i+1}) + 1, 2g(D_{i+1})] \times [1, D_{i+1} + 1]$.

Note that the number of colors in C'_i is $O(D_{i+1}g(D_{i+1}))$. Note also that nodes in different layers have disjoint colors. Hence, we obtain a global coloring. Since g and f_1 are moderately increasing it can be shown that this first phase of the algorithm terminates by time $O(\sum_{j=1}^{|\Lambda|} f_j(q_j^*))$, and that the total number of colors used in this phase is $O(\Delta g(\Delta))$.

The second phase consists of running a second algorithm to change the set of possible colors of nodes in layer i from C'_i to $C_i := [g(D_{i+1}) + 1, 2g(D_{i+1})]$. Specifically, on layer i , we execute \mathcal{A}^Γ using the guess $\hat{\Delta} = D_{i+1}$ for the parameter Δ and the guess $\tilde{m} = O(D_{i+1}g(D_{i+1}))$ for the parameter m (recall that $\Gamma \subseteq \{\Delta, m\}$). This colors each layer with colors taken from the range $[1, g(D_{i+1})]$. Let v be in layer i and let $c(v)$ be the color assigned to v by \mathcal{A}^Γ . The final color of v given by our desired algorithm \mathcal{A} is $g(D_{i+1}) + c(v)$. Thus, the colors assigned to the nodes in layer i belong to $[g(D_{i+1}) + 1, 2g(D_{i+1})]$, and are therefore disjoint on different layers. The algorithm is executed on each layer independently, all in parallel, and hence, we obtain a coloring. Moreover, since g is moderately-increasing, the total number of colors used is $O(g(\Delta))$. Finally, using the assumptions on g , and on the dependencies of f on Δ and m , it can be shown that the running time of the second phase is $O(\sum_{i=1}^{\ell} f_i(q_i^*))$. Combining this with the running time of the first phase, we obtain the theorem. \square

Theorem 3 now follows as a direct corollary of the Theorem 10. Regarding edge-coloring, observe that Barenboim and Elkin [7] obtain their edge-coloring algorithm on general graphs by running a vertex-coloring algorithm on the line-graph of the given graph. This vertex-coloring algorithm assumes the knowledge of m and Δ and uses the same number of color and time complexity as the resulted edge-coloring algorithm. Using Theorem 10, we can transform that vertex-coloring algorithm [7] designed for the family of line graphs

into a uniform one, having the same asymptotic number of colors and running time. Hence, Theorem 4 follows.

6. REFERENCES

- [1] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem.*, J. Algorithms, 7 (1986), pp. 567–583.
- [2] B. AWERBUCH, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.
- [3] B. AWERBUCH, M. LUBY, A. V. GOLDBERG, AND S. A. PLOTKIN, *Network decomposition and locality in distributed computation*, in FOCS, 1989, pp. 364–369.
- [4] L. BARENBOIM AND M. ELKIN, *Distributed $(\Delta + 1)$ -coloring in linear (in Δ) time*, in STOC, 2009, pp. 111–120.
- [5] ———, *Deterministic distributed vertex coloring in polylogarithmic time*, in PODC, 2010, pp. 410–419.
- [6] ———, *Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition*, Distrib. Comput., 22 (2010), pp. 363–379.
- [7] ———, *Distributed deterministic edge coloring using bounded neighborhood independence*, in PODC, 2011. Available at <http://arxiv.org/abs/1010.2454>.
- [8] J. L. BENTLEY AND A. C.-C. YAO, *An almost optimal algorithm for unbounded searching*, Inf. Process. Lett., 5 (1976), pp. 82–87.
- [9] R. COHEN, P. FRAIGNAUD, D. ILCINKAS, A. KORMAN, AND D. PELEG, *Label-guided graph exploration by a finite automaton*, ACM Trans. Algorithms, 4 (2008), pp. 42:1–42:18.
- [10] R. COLE AND U. VISHKIN, *Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms*, in STOC, 1986, pp. 206–219.
- [11] B. DERBEL, C. GAVOILLE, D. PELEG, AND L. VIENNOT, *On the locality of distributed sparse spanner construction*, in PODC, 2008, pp. 273–282.
- [12] D. DERENIOWSKI AND A. PELC, *Drawing maps with advice*, in DISC, Springer-Verlag, 2010, pp. 328–342.
- [13] P. FRAIGNAUD, C. GAVOILLE, D. ILCINKAS, AND A. PELC, *Distributed computing with advice: information sensitivity of graph coloring*, Distrib. Comput., 21 (2009), pp. 395–403.
- [14] P. FRAIGNAUD, D. ILCINKAS, AND A. PELC, *Communication algorithms with advice*, J. Comput. Syst. Sci., 76 (2010), pp. 222–232.
- [15] P. FRAIGNAUD, A. KORMAN, AND E. LEBHAR, *Local mst computation with short advice*, in SPAA, 2007, pp. 154–160.
- [16] P. FRAIGNAUD, A. KORMAN, AND D. PELEG, *Local distributed decision*. Submitted for Publication.
- [17] A. V. GOLDBERG AND S. A. PLOTKIN, *Efficient parallel algorithms for $(\Delta + 1)$ -coloring and maximal independent set problem*, in STOC, 1987, pp. 315–324.
- [18] A. V. GOLDBERG, S. A. PLOTKIN, AND G. E. SHANNON, *Parallel symmetry-breaking in sparse graphs*, SIAM J. Discrete Math., 1 (1988), pp. 434–446.
- [19] M. HAŃCOWIAK, M. KAROŃSKI, AND A. PANCONESI, *On the distributed complexity of computing maximal matchings*, SIAM J. Discrete Math., 15 (2001/02), pp. 41–57.
- [20] A. KORMAN AND S. KUTTEN, *Distributed verification of minimum spanning trees*, Distrib. Comput., 20 (2007), pp. 253–266.
- [21] A. KORMAN, S. KUTTEN, AND D. PELEG, *Proof labeling schemes*, Distrib. Comput., 22 (2010), pp. 215–233.
- [22] F. KUHN, *Weak graph colorings: distributed algorithms and applications*, in SPAA, 2009, pp. 138–144.
- [23] F. KUHN, T. MOSCIBRODA, AND R. WATTENHOFER, *What cannot be computed locally!*, in PODC, 2004, pp. 300–309.
- [24] F. KUHN AND R. WATTENHOFER, *On the complexity of distributed graph coloring*, in PODC, 2006, pp. 7–15.
- [25] C. LENZEN, Y. OSWALD, AND R. WATTENHOFER, *What can be approximated locally?: case study: dominating sets in planar graphs*, in SPAA, 2008, pp. 46–54.
- [26] N. LINIAL, *Distributive graph algorithms global solutions from local data*, in FOCS, 1987, pp. 331–335.
- [27] ———, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), p. 193.
- [28] Z. LOTKER, B. PATT-SHAMIR, AND A. ROSÉN, *Distributed approximate matching*, SIAM J. Comput., 39 (2009), pp. 445–460.
- [29] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.
- [30] K. NAKANO AND S. OLARIU, *Uniform leader election protocols for radio networks*, IEEE Trans. Parallel Distrib. Syst., 13 (2002), pp. 516–526.
- [31] M. NAOR AND L. STOCKMEYER, *What can be computed locally?*, SIAM J. Comput., 24 (1995), pp. 1259–1277.
- [32] A. PANCONESI AND R. RIZZI, *Some simple distributed algorithms for sparse networks*, Distrib. Comput., 14 (2001), pp. 97–100.
- [33] A. PANCONESI AND A. SRINIVASAN, *On the complexity of distributed network decomposition*, J. Algorithms, 20 (1996), pp. 356–374.
- [34] D. PELEG, *Distributed computing. A locality-sensitive approach.*, SIAM Monographs on Discrete Mathematics and Applications, SIAM, 343 p., 2000.
- [35] J. SCHNEIDER AND R. WATTENHOFER, *A new technique for distributed symmetry breaking*, in PODC, 2010, pp. 257–266.
- [36] ———, *An optimal maximal independent set algorithm for bounded-independence graphs*, Distrib. Comput., 22 (2010), pp. 1–13.
- [37] M. SZEGEDY AND S. VISHWANATHAN, *Locality based graph coloring*, in STOC, 1993, pp. 201–207.