Lower Bounds for Maximal Matchings and Maximal Independent Sets

ALKIDA BALLIU, Aalto University, Finland and University of Freiburg, Germany
SEBASTIAN BRANDT, ETH Zurich, Switzerland
JUHO HIRVONEN, Aalto University, Finland
DENNIS OLIVETTI, Aalto University, Finland and University of Freiburg, Germany
MIKAËL RABIE, Aalto University, Finland and Université de Paris, France
JUKKA SUOMELA, Aalto University, Finland

There are distributed graph algorithms for finding maximal matchings and maximal independent sets in $O(\Delta + \log^* n)$ communication rounds; here n is the number of nodes and Δ is the maximum degree. The lower bound by Linial (1987, 1992) shows that the dependency on n is optimal: these problems cannot be solved in $o(\log^* n)$ rounds even if $\Delta = 2$. However, the dependency on Δ is a long-standing open question, and there is currently an exponential gap between the upper and lower bounds.

We prove that the upper bounds are tight. We show that any algorithm that finds a maximal matching or maximal independent set with probability at least $1 - 1/n$ requires $\Omega(\min\{\Delta, \log\log n/\log\log\log n\})$ rounds in the LOCAL model of distributed computing. As a corollary, it follows that any deterministic algorithm that finds a maximal matching or maximal independent set requires $\Omega(\min\{\Delta, \log n/\log\log n\})$ rounds; this is an improvement over prior lower bounds also as a function of n.

CCS Concepts: • **Theory of computation → Distributed computing models**; *Complexity classes.*

Additional Key Words and Phrases: Maximal matching, maximal independent set, distributed graph algorithms, lower bounds

ACM Reference Format:
Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2021. Lower Bounds for Maximal Matchings and Maximal Independent Sets. 1, 1 (June 2021), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There are four classic problems that have been studied extensively in distributed graph algorithms since the very beginning of the field in the 1980s [22]: *maximal independent set* (MIS), *maximal matching* (MM), *vertex coloring with* $\Delta + 1$ *colors*, and *edge coloring with* $2\Delta - 1$ *colors*; here Δ is the maximum degree of the graph. All of these problems are trivial to solve with a greedy centralized algorithm, but their distributed computational complexity has remained an open question.

In this work, we resolve the distributed complexity of MIS and MM in the region $\Delta \ll \log\log n$. In this region, for the LOCAL model [33, 40] of distributed computing, the fastest known algorithms for these problems are:

Authors' addresses: Alkida Balliu, Aalto University, Helsinki, Finland, University of Freiburg, Freiburg, Germany, alkida.balliu@cs.uni-freiburg.de; Sebastian Brandt, ETH Zurich, Zurich, Switzerland, brandts@ethz.ch; Juho Hirvonen, Aalto University, Helsinki, Finland, juho.hirvonen@aalto.fi; Dennis Olivetti, Aalto University, Helsinki, Finland, University of Freiburg, Freiburg, Germany, dennis.olivetti@cs.uni-freiburg.de; Mikaël Rabie, Aalto University, Helsinki, Finland, Université de Paris, Paris, France, mikael.rabie@irif.fr; Jukka Suomela, Aalto University, Helsinki, Finland, jukka.suomela@aalto.fi.

- MM is possible in $O(\Delta + \log^* n)$ rounds [38].
- MIS is possible in $O(\Delta + \log^* n)$ rounds [9].

Nowadays we know how to find a vertex or edge coloring with $O(\Delta)$ colors in $o(\Delta) + O(\log^* n)$ rounds [7, 19]. Hence the current algorithms for both MIS and MM are conceptually very simple: color the vertices or edges with $O(\Delta)$ colors, and then construct an independent set or matching by going through all color classes one by one (first select all nodes or edges of color 1, then select the nodes or edges of color 2 that are not adjacent to previously selected nodes or edges, then do the same for color 3, etc.). The second part is responsible for the $O(\Delta)$ term in the running time, and previously we had no idea if this is necessary.

1.1 Prior work

Already in the 1990s we had a complete understanding of the term $\log^* n$:

- MM is not possible in $f(\Delta) + o(\log^* n)$ rounds for any f [32, 33, 36].
- MIS is not possible in $f(\Delta) + o(\log^* n)$ rounds for any f [32, 33, 36].

Here the upper bounds are deterministic and the lower bounds hold also for randomized algorithms.

However, we have had no lower bounds that would exclude the existence of algorithms of complexity $o(\Delta) + O(\log^* n)$ for either of these problems [8, 42]. For regular graphs we have not even been able to exclude the possibility of solving both of these problems in time $O(\log^* n)$, while for the general case the best lower bound as a function of Δ was $\Omega(\log \Delta / \log\log \Delta)$ [17, 28–30].

1.2 Contributions

We close the gap and prove that the current upper bounds are tight. There is no algorithm of complexity $o(\Delta) + O(\log^* n)$ for MM or MIS. More precisely, our main result is:

> Any algorithm that solves MM or MIS with probability at least $1 - 1/n$ requires $\Omega\left(\min\left\{\Delta, \frac{\log\log n}{\log\log\log n}\right\}\right)$ rounds in the LOCAL model.
>
> Any deterministic algorithm that solves MM or MIS requires $\Omega\left(\min\left\{\Delta, \frac{\log n}{\log\log n}\right\}\right)$ rounds in the LOCAL model.

As corollaries, we have a new separation and a new equivalence in the $\Delta \ll \log\log n$ region (see Section 2.2):

- MM and MIS are strictly harder than $(\Delta + 1)$-vertex coloring and $(2\Delta - 1)$-edge coloring.
- MM and MIS are exactly as hard as greedy coloring.

1.3 Plan

We will present a simpler version of our new linear-in-Δ lower bound in Section 3. There we will introduce a restricted setting of distributed computing—deterministic algorithms in the port-numbering model—and explain the key ideas using that. In Section 4 we will then see how to extend the result to randomized and deterministic algorithms in the usual LOCAL model of computing.

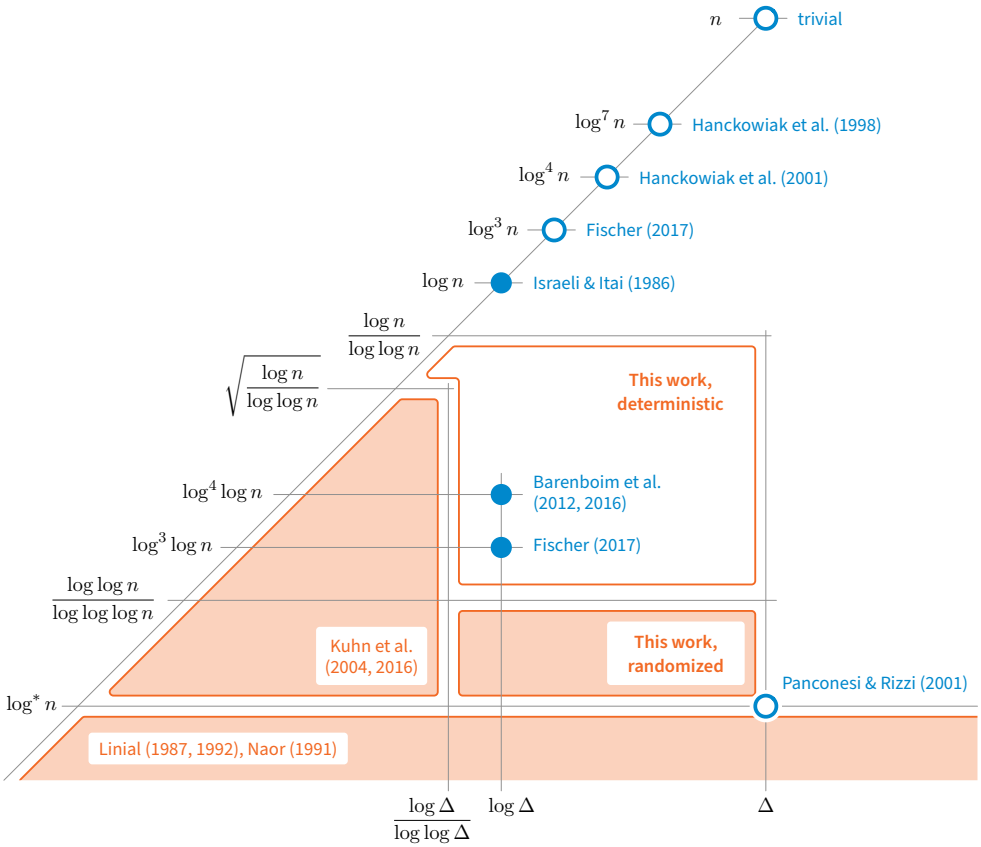


Fig. 1. Distributed algorithms for maximal matching: upper bounds (blue dots) and lower bounds (orange regions). Filled dots are randomized algorithms and filled regions are lower bounds for randomized algorithms; white dots are deterministic algorithms and white regions are lower bounds for deterministic algorithms. The running time is represented here in the form $O(f(\Delta) + g(n))$, the horizontal axis represents the $f(\Delta)$ term, and the vertical axis represents the $g(n)$ term.

2 RELATED WORK

2.1 MIS and MM

For MIS and MM, as well as for other classical symmetry-breaking problems, there are three major families of algorithms:

- Deterministic algorithms suitable for small Δ, with a complexity of the form $f(\Delta) + O(\log^* n)$.
- Deterministic algorithms suitable also for large Δ, with superlogarithmic complexities as a function of n.
- Randomized algorithms suitable also for large Δ, with at most logarithmic complexities as a function of n.

We summarize the state of the art in Table 1 and Figure 1; see e.g. Alon et al. [1], Luby [34, 35], Israeli and Itai [27], Panconesi and Srinivasan [39], Hanckowiak et al. [24, 25], Barenboim et al. [10, 11], and Ghaffari [21] for more prior work on maximal matchings and maximal independent sets. The listed upper bounds for MIS by Rozhoň and Ghaffari [41] were obtained after the preliminary conference version of our work.

Previously, it was not known if any of these algorithms are optimal. In essence, there have been only two lower bound results:

Table 1. Efficient algorithms for MM and MIS.

Problem	Δ	Randomness	Complexity	Reference
MM	small	deterministic	$O(\Delta + \log^* n)$	Panconesi and Rizzi [38]
	large	deterministic	$O(\log^2 \Delta \cdot \log n)$	Fischer [18]
	large	randomized	$O(\log \Delta + \log^3 \log n)$	Barenboim et al. [10, 11], Fischer [18]
MIS	small	deterministic	$O(\Delta + \log^* n)$	Barenboim et al. [9]
	large	deterministic	$O(\log^7 n)$	Rozhoň and Ghaffari [41]
	large	randomized	$O(\log \Delta + \log^7 \log n)$	Rozhoň and Ghaffari [41]

- Linial [32, 33] and Naor [36] show that any deterministic or randomized algorithm for MM or MIS requires $\Omega(\log^* n)$ rounds, even if we have $\Delta = 2$.
- Kuhn et al. [28, 29, 30] show that any deterministic or randomized algorithm for MM or MIS requires

$$\Omega\left(\min\left\{\frac{\log \Delta}{\log \log \Delta}, \sqrt{\frac{\log n}{\log \log n}}\right\}\right)$$

  rounds.

Hence, for example, when we look at the fastest MM algorithms, dependency on n in $O(\Delta + \log^* n)$ is optimal, and dependency on Δ in $O(\log \Delta + \log^3 \log n)$ is near-optimal. However, could we get the best of both worlds and solve MM or MIS in e.g. $O(\log \Delta + \log^* n)$ rounds?

In this work we show that the answer is no. There is no algorithm that runs in time $o(\Delta) + O(\log^* n)$. The current upper bounds for the case of a small Δ are optimal. Moreover, our work shows there is not much room for improvement in the dependency on n in the algorithm by Fischer [18], either.

With this result we resolve Open Problem 11.6 in the manuscript of Barenboim and Elkin [8], and present a proof for the conjecture of Göös et al. [23].

2.2 Coloring

It is interesting to compare MIS and MM with the classical distributed coloring problems: vertex coloring with $\Delta + 1$ colors and edge coloring with $2\Delta - 1$ colors [8]. As recently as in 2014, the fastest algorithms for all of these problems in the "small Δ" region had the same complexity as MIS and MM, $O(\Delta + \log^* n)$ rounds [9]. However, in 2015 the paths diverged: Barenboim [7] and Fraigniaud et al. [19] have presented algorithms for graph coloring in $o(\Delta) + O(\log^* n)$ rounds, and hence we now know that coloring is strictly easier than MM or MIS in the small Δ region.

The only known lower bound for $(\Delta + 1)$-vertex coloring and $(2\Delta - 1)$-edge coloring is $\Omega(\log^* n)$ [32, 33, 36]; better lower bounds are known only for coloring algorithms of a certain form, called locally-iterative algorithms [31, 43].

However, there is a variant of $(\Delta + 1)$-vertex coloring that is closely related to MIS: *greedy coloring* [20]. Greedy coloring is trivially at least as hard as MIS, as color class 1 in any greedy coloring gives an MIS. On the other hand, greedy coloring is possible in time $O(\Delta + \log^* n)$, as we can turn an $O(\Delta)$-vertex coloring into a greedy coloring in $O(\Delta)$ rounds (and this was actually already known to be tight). Now our work shows that greedy coloring is exactly as hard as MIS. In a sense this is counterintuitive: finding just color class 1 of a greedy coloring is already asymptotically as hard as finding the entire greedy coloring.

2.3 Restricted lower bounds

While no linear-in-Δ lower bounds for MM or MIS were known previously for the usual LOCAL model of distributed computing, there was a tight bound for a toy model of distributed computing: deterministic algorithms in the *edge coloring model*. Here we are given a proper edge coloring of the graph with $\Delta + 1$ colors, the nodes are anonymous, and the nodes can use the edge colors to refer to their neighbors. In this setting there is a trivial algorithm that finds an MM in $O(\Delta)$ rounds: go through color classes one by one and greedily add edges that are not adjacent to anything added so far. It turns out there is a matching lower bound: no algorithm solves this problem in $o(\Delta)$ rounds in the same model [26].

The same technique was later used to study *maximal fractional matchings* in the LOCAL model of computing. This is a problem that can be solved in $O(\Delta)$ rounds (independent of n) in the usual LOCAL model [2], and there was a matching lower bound that shows that the same problem cannot be solved in $o(\Delta)$ rounds (independent of n) in the same model [23].

While these lower bounds were seen as a promising indicator that there might be a linear-in-Δ lower bound in a more general setting, the previous techniques turned out to be a dead end. In particular, they did not tell anything nontrivial about the complexity of MM or MIS in the usual LOCAL model. Now we know that an entirely different kind of approach was needed—even though the present work shares some coauthors with [23, 26], the techniques of the present work are entirely unrelated to those.

2.4 Speedup simulation technique

The technique that we use in this work is based on *speedup simulation*, a.k.a., *round elimination*. In essence, the idea is that we assume we have an algorithm A that solves a problem Π in T rounds, and then we construct a new algorithm A' that solves another problem Π' in $T' \le T - 1$ rounds. A node in algorithm A' gathers its radius-T' neighborhood, considers all possible ways of extending it to a radius-T neighborhood, simulates A for each such extension, and then uses the output of A to choose its own output. Now if we can iterate the speedup step for k times (without reaching a trivial problem), we know that the original problem requires at least k rounds to solve.

This approach was first used by Linial [32, 33] and Naor [36] to prove that graph coloring in cycles requires $\Omega(\log^* n)$ rounds. This was more recently used to prove lower bounds for *sinkless orientations*, algorithmic Lovász local lemma, and Δ-coloring [13, 15], as well as to prove lower bounds for *weak coloring* [6, 12].

In principle, the approach can be used with *any* locally checkable graph problem, in a mechanical manner [12]. However, if one starts with a natural problem Π (e.g. MIS, MM, or graph coloring) and applies the speedup simulation in a mechanical manner, the end result is typically a problem Π' that does not have any natural interpretation or simple description, and it gets quickly exponentially worse. The key technical challenge that the present work overcomes is the construction of a sequence of nontrivial problems $\Pi_1, \Pi_2, \ldots$ such that each of them has a relatively simple description and we can nevertheless apply speedup simulation for any consecutive pair of them. Note that the simplicity of the descriptions is crucial in two ways: it allows us to prove the desired relation between the complexities of any two subsequent problems in the sequence, and it enables us to show that the problems indeed cannot be solved in 0 rounds.

The formalism that we use is closely related to [12]—in essence, we generalize the formalism from graphs to hypergraphs, then represent the hypergraph as a bipartite graph, and we arrive at the formalism that we use in the present work to study maximal matchings in bipartite graphs.

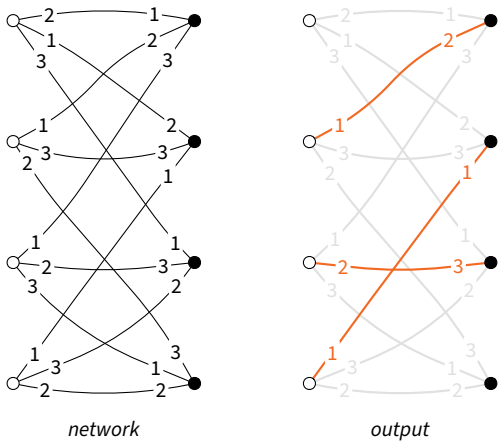


Fig. 2. A 3-regular bipartite port-numbered network and a maximal matching.

3 LOWER BOUND IN THE PORT-NUMBERING MODEL

Consider the following setting: We have a Δ-*regular bipartite graph*; the nodes in one part are white and in the other part black. Each node has a *port numbering* for the incident edges; the endpoints of the edges incident to a node are numbered in some arbitrary order with $1, 2, \ldots, \Delta$. See Figure 2 for an illustration.

The graph represents the topology of a communication network: each node is a computer, and each edge is a communication link. Initially each computer only knows its own color (black or white) and the number of ports (Δ); the computers are otherwise identical. Computation proceeds in *synchronous communication rounds*—in each round, each node can send an arbitrary message to each of its neighbors, then receive a message from each of its neighbors, and update its own state. After some T communication rounds, all nodes have to stop and announce their own part of the solution; here T is the *running time* of the algorithm.

We are interested in algorithms for finding a *maximal matching*; eventually each node has to know whether it is matched and in which port. There is a very simple algorithm that solves this in $T = O(\Delta)$ rounds [24]: In iteration $i = 1, 2, \ldots, \Delta$, unmatched white nodes send a proposal to their port number i, and black nodes accept the first proposal that they receive, breaking ties using port numbers. See Figure 3 for an example.

Hence bipartite maximal matching can be solved in $O(\Delta)$ rounds in Δ-regular two-colored graphs, and the running time is independent of the number of nodes. Surprisingly, nobody has been able to tell if this algorithm is optimal, or anywhere close to optimal. There are no algorithms that break the linear-in-Δ barrier (without introducing some dependency on n in the running time), and there are no nontrivial lower bounds—we have not been able to exclude even the possibility of solving maximal matchings in this setting in e.g. 10 rounds, independently of Δ and n. If we look at a bit more general setting of graphs of degree at most Δ (instead of Δ-regular graphs), there is a lower bound of $\Omega(\log \Delta / \log \log \Delta)$ [28–30], but there is still an exponential gap between the upper and the lower bound.

In this section we show that the trivial proposal algorithm is indeed optimal: there is no algorithm that finds a maximal matching in $o(\Delta)$ rounds in this setting. We will later extend the result to more interesting models of computing, but for now we will stick to the case of deterministic algorithms in the port-numbering model, as it is sufficient to explain all key ideas.

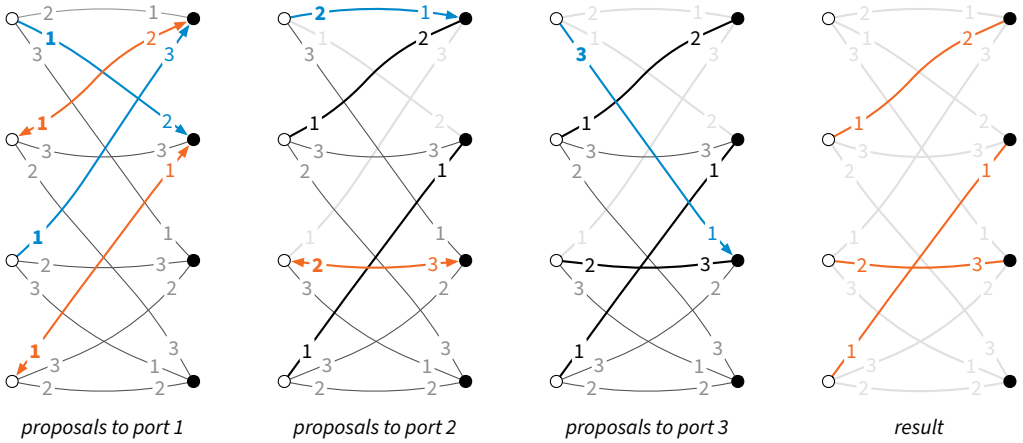


Fig. 3. The proposal algorithm finds a maximal matching in $O(\Delta)$ rounds—orange arrows are accepted proposals and blue arrows are rejected proposals.

3.1 Lower bound idea

Our plan is to prove a lower bound using a *speedup simulation argument* [4, 12, 13, 15, 32, 33, 36]. The idea is to define a sequence of graph problems $\Pi_1, \Pi_2, \ldots, \Pi_k$ such that if we have an algorithm A_i that solves Π_i in T_i rounds, we can construct an algorithm A_{i+1} that solves Π_{i+1} strictly faster, in $T_{i+1} \leq T_i - 1$ rounds (unless $T_i = 0$, in which case we would get that $T_{i+1} = 0$). Put otherwise, we show that solving Π_i takes at least one round more than solving Π_{i+1}. Then if we can additionally show that Π_k is still a nontrivial problem that cannot be solved in zero rounds, we know that the complexity of Π_1 is at least k rounds.

Now we would like to let Π_1 be the maximal matching problem, and identify a suitable sequence of relaxations of the maximal matching problem. A promising candidate might be, e.g., the following problem that we call here a *k-matching* for brevity.

Definition 3.1 (k-matching)**.** Given a graph $G = (V, E)$, a set of edges $M \subseteq E$ is a k-matching if

(1) every node is incident to at most k edges of M,
(2) if a node is not incident to any edge of M, then all of its neighbors are incident to at least one edge of M.

Note that with this definition, a 1-matching is exactly the same thing as a maximal matching. Also it seems that finding a k-matching is easier for larger values of k. For example, we could modify the proposal algorithm so that in each iteration white nodes send k proposals in parallel, and stop as soon as at least one is accepted, and this way find a k-matching in $O(\Delta/k)$ rounds.

We could try to define Π_i as the problem of finding an i-matching. Then, we could try to prove that, given an algorithm for finding an i-matching, we can construct a strictly faster algorithm for finding an $(i + 1)$-matching. Unfortunately, a direct attack along these lines does not seem to work, but this serves nevertheless as a useful guidance that will point us in the right direction.

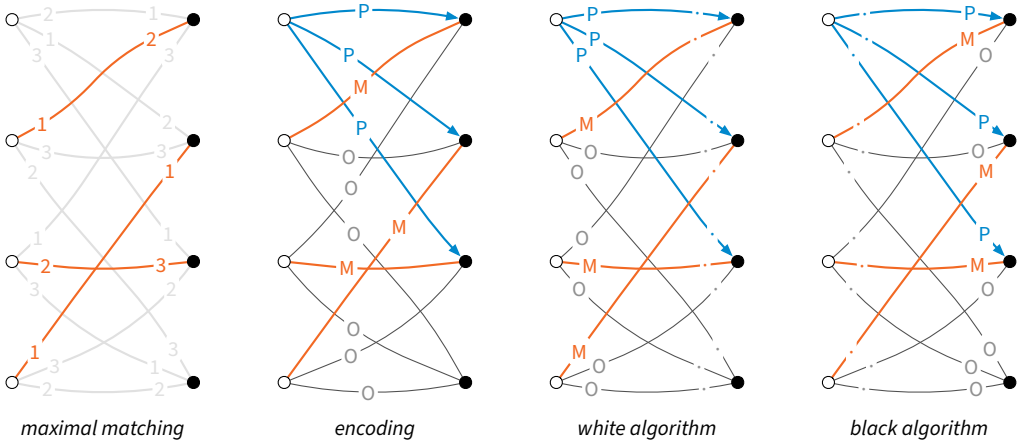


Fig. 4. Encoding maximal matchings with $\Sigma = \{\mathsf{M}, \mathsf{P}, \mathsf{O}\}$.

3.2 Formalism and notation

We will first make the setting as simple and localized as possible. It will be convenient to study graph problems that are of the following form—we call these *edge labeling problems*:

(1) The task is to label the edges of the bipartite graph with symbols from some *alphabet* Σ.
(2) A *problem specification* is a pair $\Pi = (W, B)$, where W is the set of feasible labelings of the edges incident to a white node, and B is the set of feasible labelings for the edges incident to a black node.

In these problems, the feasibility of a solution does not depend on the port numbering. Hence each member of W and B is a *multiset* that contains Δ elements from alphabet Σ. For example, if we have $\Sigma = \{0, 1\}$ and $\Delta = 3$, then $W = \big\{\{0, 0, 0\}, \{0, 0, 1\}\big\}$ indicates that a white node is happy (that is, the labeling of its incident edges is correct according to W) if it is incident to exactly 0 or 1 edges with label 1.

However, for brevity we will here represent multisets as *words*, and write e.g. $W = \big\{000, 001\big\}$. We emphasize that the order of the elements does not matter here, and we could equally well write e.g. $W = \big\{000, 010\big\}$. Now that W and B are languages over alphabet Σ, we can conveniently use regular expressions to represent them. When $x_1, x_2, \ldots, x_k \in \Sigma$ are symbols of the alphabet, we use the shorthand notation $[x_1 x_2 \ldots x_k] = (x_1 | x_2 | \ldots | x_k)$. With this notation, we can represent the above example concisely as $W = 000 \mid 001$, or $W = 00[01]$, or even $W = 0^2[01]$.

3.2.1 Example: encoding maximal matchings. The most natural way to encode maximal matchings would be to use e.g. labels 0 and 1 on the edges, with 1 to indicate an edge in the matching. However, this is not compatible with the above formalism: we would have to have $0^\Delta \in W$ and $0^\Delta \in B$ to allow for unmatched nodes, but then we would also permit a trivial all-0 solution. To correctly capture the notion of maximality, we will use three labels, $\Sigma = \{\mathsf{M}, \mathsf{P}, \mathsf{O}\}$, with the following rules:

$$\begin{aligned} W &= \mathsf{M}\mathsf{O}^{\Delta-1} \mid \mathsf{P}^\Delta, \\ B &= \mathsf{M}[\mathsf{PO}]^{\Delta-1} \mid \mathsf{O}^\Delta. \end{aligned} \tag{1}$$

For a matched white node, one edge is labeled with an M (matched) and all other edges are labeled with an O (other). However, for an unmatched white node, all incident edges have to be labeled with a P (pointer); the intuition is that P

points to a matched black neighbor. The rules for the black nodes ensure that pointers do not point to unmatched black nodes (a P implies exactly one M), and that black nodes are unmatched only if all white neighbors are matched (all incident edges labeled with Os). See Figure 4 for an illustration.

3.2.2 White and black algorithms. Let $\Pi = (W, B)$ be an edge labeling problem. We say that A is a *white algorithm* that solves Π if in A each white node outputs a labeling of its incident edges, and such a labeling forms a feasible solution to Π. Black nodes produce an empty output.

Conversely, in a *black algorithm*, each black node outputs the labels of its incident edges, and white nodes produce an empty output. See Figure 4 for illustrations. Note that a black algorithm can be easily turned into a white algorithm if we use one additional communication round, and vice versa.

3.2.3 Infinite trees vs. finite regular graphs. It will be convenient to first present the proof for the case of infinite Δ-regular trees. In essence, we will show that any algorithm A that finds a maximal matching in $T = o(\Delta)$ rounds will fail around some node u in some infinite Δ-regular tree G (for some specific port numbering), where failing around node u means that the labels assigned to the edges incident to u form a configuration that is not allowed by the problem. Then it is also easy to construct a finite Δ-regular graph G' such that the radius-$(T+1)$ neighborhood of u in G (including the port numbering) is isomorphic to the radius-$(T+1)$ neighborhood of some node u' in G', and therefore A will also fail around u' in G'.

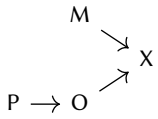
3.3 Parametrized problem family

We will now introduce a parametrized family of problems $\Pi_\Delta(x, y)$, where $x + y \le \Delta$. The problem is defined so that $\Pi_\Delta(0, 0)$ is equivalent to maximal matchings (1) and the problem becomes easier when we increase x or y. We will use the alphabet $\Sigma = \{\mathsf{M}, \mathsf{P}, \mathsf{O}, \mathsf{X}\}$, where M, P, and O have a role similar to maximal matchings and X acts as a *wildcard*. We define $\Pi_\Delta(x, y) = \big(W_\Delta(x, y), B_\Delta(x, y)\big)$, where

$$\begin{aligned} W_\Delta(x, y) &= \left(\mathsf{M}\mathsf{O}^{d-1} \;\middle|\; \mathsf{P}^d\right)\mathsf{O}^y\mathsf{X}^x, \\ B_\Delta(x, y) &= \left([\mathsf{M}\mathsf{X}]\,[\mathsf{P}\mathsf{O}\mathsf{X}]^{d-1} \;\middle|\; [\mathsf{O}\mathsf{X}]^d\right)[\mathsf{P}\mathsf{O}\mathsf{X}]^y\,[\mathsf{M}\mathsf{P}\mathsf{O}\mathsf{X}]^x, \end{aligned} \tag{2}$$

where $d = \Delta - x - y$.

The following partial order represents the "strength" of the symbols from the perspective of black nodes:



(3)

The interpretation is that from the perspective of $B_\Delta(x, y)$, symbol X is feasible wherever M or O is feasible, and O is feasible wherever P is feasible. Hence, we say that X is stronger than M, O, and P, and that O is stronger than P. Furthermore, all relations are strict in the sense that e.g. replacing an X with an M may lead to a word not in $B_\Delta(x, y)$.

Here are three examples of problems in family $\Pi_\Delta(\cdot, \cdot)$, with some intuition (from the perspective of a white algorithm):

- $\Pi_\Delta(0, 0)$: Maximal matching. Note that we cannot use symbol X at all, as it does not appear in $W_\Delta(0, 0)$.
- $\Pi_\Delta(0, 1)$: Unmatched white nodes will use O instead of P once—note that by (3) this is always feasible for the black node at the other end of the edge. Unmatched black nodes can accept P instead of O once.

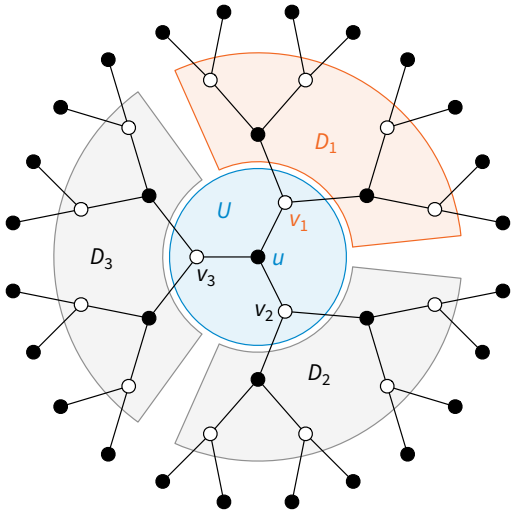


Fig. 5. Speedup simulation, for $T = 2$ and $\Delta = 3$. The radius-$(T-1)$ neighborhood of u is U, and the radius-T neighborhood of v_i is $V_i = U \cup D_i$. The key observation is that D_i and D_j are disjoint for $i \neq j$.

- $\Pi_\Delta(1, 0)$: All white nodes will use X instead of P or O once—again, this is always feasible. All black nodes can accept anything from one port.

In essence, $\Pi_\Delta(0, y)$ resembles a problem in which we can violate maximality, while in $\Pi_\Delta(x, 0)$ we can violate the packing constraints.

3.4 Speedup simulation

Assume that A is a white algorithm that solves $\Pi_\Delta(x, y)$ in $T \geq 1$ rounds in trees, for a sufficiently large Δ. Throughout this section, let $d = \Delta - x - y$, and assume that $\Delta \geq 2x + y + 1$. We are going to proceed as follows. First, we construct a black algorithm A_1 that runs in $T - 1$ rounds and works as follows. Given a $T - 1$-radius neighborhood U, it extends U into a T-radius neighborhood around a white node in all possible ways, then it simulates A in all the obtained neighborhoods, and finally, for each incident edge, it outputs the set of the outputs given by A. On each edge, the output of algorithm A_1 is a set, and there are 15 possible sets that could appear (namely all non-empty subsets of the output label set for $\Pi_\Delta(x, y)$). As our goal is to obtain a problem sequence where each problem is from the family $\Pi_\Delta(\cdot, \cdot)$, we need to reduce the number of labels back to 4. We start by reducing the number of possible outputs to 6, by defining a new algorithm A_2 that first executes A_1 and then maps the aforementioned sets to labels, such that different sets could be mapped to the same label. We could try to exactly characterize the problem solved by A_2, by listing the constraints defined over a set of 6 labels that an output of A_2 satisfies. Instead, we are going to *simplify* the output of A_2, and reduce the number of labels from 6 to 4 by *identifying* some labels, which yields some algorithm A_3. Finally, we are going to define algorithm A_4, that does nothing else than executing A_3 and then renaming the obtained labels, and we are going to show that A_4 solves $\Pi_\Delta(x + 1, y + x)$, but in a graph where the role of black and white nodes is reversed. For a more intuitive explanation of this approach, please see Section 3.8.

3.4.1 Algorithm A_1. We will first construct a black algorithm A_1 that runs in time $T - 1$, as follows:

Each black node u gathers its radius-$(T-1)$ neighborhood U; see Figure 5. Let the white neighbors of u be $v_1, v_2, \ldots, v_\Delta$. Let V_i be the radius-T neighborhood of v_i, and let $D_i = V_i \setminus U$ be the part of V_i that u does not see.

For each i, go through all possible inputs that one can assign to D_i; here the only unknown part is the port numbering that we have in the region D_i. Then simulate A for each possible input and see how A labels the edge $e_i = \{u, v_i\}$. Let S_i be the set of labels that A assigns to edge e_i for some input D_i.

Algorithm A_1 labels edge e_i with set S_i.

3.4.2 Algorithm A_2. Now since the output alphabet of A is $\Sigma = \{\mathsf{M}, \mathsf{P}, \mathsf{O}, \mathsf{X}\}$, the new output alphabet of A_1 consists of its 15 nonempty subsets. We construct another black algorithm A_2 with alphabet $\{\boxed{\mathsf{X}}, \boxed{\mathsf{OX}}, \boxed{\mathsf{POX}}, \boxed{\mathsf{MX}}, \boxed{\mathsf{MOX}}, \boxed{\mathsf{MPOX}}\}$ that simulates A_1 and then maps the output of A_1 as follows (see Figure 6 for an example):

$$\begin{aligned}
\{\mathsf{X}\} &\mapsto \boxed{\mathsf{X}},\\
\{\mathsf{M}\}, \{\mathsf{M},\mathsf{X}\} &\mapsto \boxed{\mathsf{MX}},\\
\{\mathsf{O}\}, \{\mathsf{O},\mathsf{X}\} &\mapsto \boxed{\mathsf{OX}},\\
\{\mathsf{M},\mathsf{O}\}, \{\mathsf{M},\mathsf{O},\mathsf{X}\} &\mapsto \boxed{\mathsf{MOX}},\\
\{\mathsf{P}\}, \{\mathsf{P},\mathsf{O}\}, \{\mathsf{P},\mathsf{X}\}, \{\mathsf{P},\mathsf{O},\mathsf{X}\} &\mapsto \boxed{\mathsf{POX}},\\
\{\mathsf{M},\mathsf{P}\}, \{\mathsf{M},\mathsf{P},\mathsf{O}\}, \{\mathsf{M},\mathsf{P},\mathsf{X}\}, \{\mathsf{M},\mathsf{P},\mathsf{O},\mathsf{X}\} &\mapsto \boxed{\mathsf{MPOX}}.
\end{aligned}$$

Here the intuition is that we first make each set maximal w.r.t. (3): for example, whenever we have a set with a P, we also add an O, and whenever we have a set with an O, we also add an X. This results in only six maximal sets, and then we replace e.g. the maximal set $\{\mathsf{M},\mathsf{O},\mathsf{X}\}$ with the label $\boxed{\mathsf{MOX}}$.

3.4.3 Output of A_2. Let us analyze the output of A_2 for a black node. Fix a black node u and its neighborhood U. The key property is that regions $D_1, D_2, \ldots$ in Figure 5 do not overlap—hence if there is some input for D_1 that makes v_1 output some label L_1, and another input for D_2 that makes v_2 output some label L_2, we can also construct an input in which v_1 outputs L_1 and at the same time v_2 outputs L_2. We make the following observations:

(1) There can be at most $x+1$ edges incident to u with a label in $\{\boxed{\mathsf{MX}}, \boxed{\mathsf{MOX}}, \boxed{\mathsf{MPOX}}\}$. If there were $x+2$ such edges, say $e_1, e_2, \ldots, e_{x+2}$, then it means we could fix D_1 such that A outputs M for e_1, and simultaneously fix D_2 such that A outputs M for e_2, etc. But this would violate the property that A solves $\Pi_\Delta(x, y)$, as all words of $B_\Delta(x, y)$ contain at most $x+1$ copies of M.

(2) If there are at least $x+y+1$ edges with a label in $\{\boxed{\mathsf{POX}}, \boxed{\mathsf{MPOX}}\}$, then there has to be at least one edge with a label in $\{\boxed{\mathsf{X}}, \boxed{\mathsf{MX}}\}$. Otherwise we could choose D_i so that A outputs P on $x+y+1$ edges, and there is no M or X. But all words of $B_\Delta(x, y)$ with at least $x+y+1$ copies of P contain also at least one M or X.

3.4.4 Algorithm A_3. We construct yet another black algorithm A_3 that modifies the output of A_2 so that we replace labels only with larger labels according to the following partial order, which represents subset inclusion:

$$\begin{array}{ccccc}
\boxed{\mathsf{X}} & \longrightarrow & \boxed{\mathsf{OX}} & \longrightarrow & \boxed{\mathsf{POX}} \\
\downarrow & & \downarrow & & \downarrow \\
\boxed{\mathsf{MX}} & \longrightarrow & \boxed{\mathsf{MOX}} & \longrightarrow & \boxed{\mathsf{MPOX}}
\end{array} \tag{4}$$

Edge	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Part of $B_7(1,1)$	[MX]	[POX]	[POX]	[POX]	[POX]	[POX]	[MPOX]
White algorithm A	M	X	O	X	P	O	M
Black algorithm A_1	{M, O}	{X}	{O}	{P, X}	{P}	{P, O}	{M, P}
Black algorithm A_2	MOX	X	OX	POX	POX	POX	MPOX
Black algorithm A_3	MPOX	MX	POX	POX	POX	POX	MPOX
Black algorithm A_4	X	M	O	O	O	O	X
Part of $W_7(2,2)$	X	M	O	O	O	O	X

Fig. 6. Speedup simulation for $\Pi_7(1,1)$: an example of some possible outputs around a black node.

There are two cases:

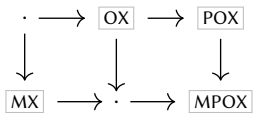
(1) There are at most $x+y$ copies of POX on edges incident to u. We also know that there are at most $x+1$ copies of labels {MX, MOX, MPOX}. Hence there have to be at least $\Delta-(x+y)-(x+1) = d-x-1$ copies of labels {X, OX}. We proceed as follows:
- Replace all of {MX, MOX} with MPOX.
- Replace some of {X, OX, POX} with MPOX so that the total number of MPOX is exactly $x+1$.
- Replace some of the remaining {X, OX} with POX so that the total number of POX is exactly $x+y$.
- Replace all of the remaining X with OX.

We are now left with exactly $x+1$ copies of MPOX, exactly $x+y$ copies of POX, and exactly $d-x-1$ copies of OX.

(2) There are more than $x+y$ copies of POX. Then we know that there is at least one copy of {X, MX}. We first proceed as follows:
- Replace all OX with POX and all MOX with MPOX.
- If needed, replace one X with MX so that there is at least one copy of MX.
- Replace all remaining copies of X with POX.

At this point we have at least one MX and all other labels are in {MPOX, POX}. Furthermore, as we originally had at most $x+1$ copies of {MX, MOX, MPOX}, and we created at most one new label in this set, we have now got at most $x+2$ copies of {MX, MPOX}. Hence we have got at least $d+y-2$ copies of POX. We continue:
- Replace some of MX with MPOX so that the total number of MX is exactly 1.
- Replace some of POX with MPOX so that the total number of POX is exactly $d+y-2$ and hence the total number of MPOX is $x+1$.

Note that we have completely eliminated labels X and MOX and we are left with the following four labels:



(5)

3.4.5 Algorithm A_4. Finally, we construct an algorithm A_4 that maps the output of A_3 as follows:

$$\text{MX} \mapsto \text{M}, \quad \text{OX} \mapsto \text{P}, \quad \text{POX} \mapsto \text{O}, \quad \text{MPOX} \mapsto \text{X}. \tag{6}$$

Note that after this mapping, poset (5) restricted to $\{M, P, O, X\}$ is the same as poset (3). One may have expected $\boxed{OX}$ to be renamed to O and $\boxed{POX}$ to be renamed to P, but we did the opposite. Let us discuss the intuition behind this choice by taking maximal matching as an example. In this problem, P is used by white notes (that are active) to certify that all black neighbors (that are passive) are matched. Similarly, we can see label O as a label that can be used by *black* nodes to certify that they have white matched neighbors. Hence, we can see *both* P and O as pointers. The renaming that we use allows us to see P as the pointer of the *active* side and O as the pointer of the *passive* side, even if the role of active and passive swapped.

3.4.6 Output of A_4. Let us analyze the output of A_4 for a white node v. The key observation is that the output of A is contained in the sets that A_1 outputs. More precisely, algorithm A_1 outputs sets based on its $T-1$-radius neighborhood, and if there is an extension of this neighborhood such that algorithm A, executed on the extended neighborhood, outputs some label on the edge corresponding to this extension, then this output label is contained in the set given by A_1 on this edge. Since all the possible outputs of A are in $(MO^{d-1}|P^d)O^yX^x$, there are two cases:

(1) We have a neighborhood in which the output of A at v matches $MO^{d-1+y}X^x$. Then:
   - There is at least one edge incident to v such that the output of A_1 contains an M. Therefore the output of A_2 on this edge is in $\{\boxed{MX}, \boxed{MOX}, \boxed{MPOX}\}$. As we followed (4), the output of A_3 for this edge is also in $\{\boxed{MX}, \boxed{MOX}, \boxed{MPOX}\}$. After remapping, the output of A_4 for this edge is in $\{M, X\}$.
   - There are at least $d-1+y$ edges incident to v such that the output of A_1 contains an O. By a similar argument, the output of A_3 for each of these edges is in $\{\boxed{OX}, \boxed{MOX}, \boxed{POX}, \boxed{MPOX}\}$, and the output of A_4 is hence in $\{P, O, X\}$.

(2) We have a neighborhood in which the output of A at v matches $P^dO^yX^x$. Then by a similar reasoning:
   - We have d edges incident to v such that the output of A_3 is in $\{\boxed{POX}, \boxed{MPOX}\}$ and hence the output of A_4 is in $\{O, X\}$.
   - We have y edges incident to v such that the output of A_3 is in $\{\boxed{OX}, \boxed{MOX}, \boxed{POX}, \boxed{MPOX}\}$ and hence the output of A_4 is in $\{P, O, X\}$.

Hence the neighborhood of a white node in A_4 satisfies

$$\begin{aligned} W^* &= [MX]\,[POX]^{d-1+y}\,[MPOX]^x \mid [OX]^d\,[POX]^y\,[MPOX]^x \\ &= \left([MX]\,[POX]^{d-1} \mid [OX]^d\right)[POX]^y\,[MPOX]^x. \end{aligned}$$

By construction, the output of a black node in A_3 follows one of these patterns:

(1) $x+1$ copies of $\boxed{MPOX}$, $x+y$ copies of $\boxed{POX}$, and $d-x-1$ copies of $\boxed{OX}$.
(2) 1 copy of $\boxed{MX}$, $d+y-2$ copies of $\boxed{POX}$, and $x+1$ copies of $\boxed{MPOX}$.

Hence after remapping, the output of a black node in A_4 satisfies

$$\begin{aligned} B^* &= P^{d-x-1}O^{x+y}X^{x+1} \mid MO^{d+y-2}X^{x+1} \\ &= \left(MO^{d-x-2} \mid P^{d-x-1}\right)O^{x+y}X^{x+1}. \end{aligned}$$

Hence A_4 solves $\Pi^* = (W^*, B^*)$. Now observe that $W^* = B_\Delta(x, y)$ and $B^* = W_\Delta(x+1, y+x)$.

Furthermore, observe that $B_\Delta(x, y) \subseteq B_\Delta(x+1, y+x)$. Hence A_4 also solves the problem

$$\Pi'_\Delta(x+1, y+x) = \big(B_\Delta(x+1, y+x),\ W_\Delta(x+1, y+x)\big),$$

which is otherwise exactly the same as $\Pi_\Delta(x+1, y+x)$, but the roles of the black and white nodes are reversed.

3.4.7 Conclusion. We started with the assumption that A is a white algorithm that solves $\Pi_\Delta(x, y)$ in T rounds. We constructed a black algorithm A_1 that runs in $T-1$ rounds. Then we applied three steps of local postprocessing to obtain algorithm A_4; note that A_4 is also a black algorithm with the running time $T-1$, as all postprocessing steps can be done without communication. We observed that A_4 solves $\Pi'_\Delta(x+1, y+x)$.

Now if there is a black algorithm that solves $\Pi'_\Delta(x+1, y+x)$ in $T-1$ rounds, we can reverse the roles of the colors and obtain a white algorithm that solves $\Pi_\Delta(x+1, y+x)$ in $T-1$ rounds. We have the following lemma:

Lemma 3.1. *Assume that there exists a white algorithm that solves $\Pi_\Delta(x, y)$ in $T \geq 1$ rounds in trees, for $\Delta \geq 2x+y+1$. Then there exists a white algorithm that solves $\Pi_\Delta(x+1, y+x)$ in $T-1$ rounds in trees.*

By a repeated application of Lemma 3.1, we obtain the following corollary:

Corollary 3.2. *Assume that there exists a white algorithm that solves $\Pi_\Delta(x, y)$ in T rounds in trees, for $\Delta \geq x+y+T(x+1+(T-1)/2)$. Then there is a white algorithm that solves $\Pi_\Delta(x', y')$ in 0 rounds in trees for*

$$x' = x + T,$$
$$y' = y + T\big(x + (T-1)/2\big).$$

3.5 Base case

Now to make use of Corollary 3.2, it is sufficient to show that there are problems in the family $\Pi_\Delta(x, y)$ that cannot be solved with 0-round white algorithms:

Lemma 3.3. *There is no white algorithm that solves $\Pi_\Delta(x, y)$ in 0 rounds in trees when $\Delta \geq x+y+2$.*

Proof. Since by increasing x and y the problem does not get harder, it is enough to prove the statement for $\Delta = x+y+2$. As we are looking at deterministic algorithms in the port-numbering model, in a 0-round white algorithm all white nodes produce the same output. By definition, the output has to be a permutation of some word $w \in W_\Delta(x, y)$. Now there are only two possibilities:

(1) $w = \mathsf{M}\mathsf{O}^{y+1}\mathsf{X}^x$: There is some port p that is labeled with M. Now consider a black node u such that for all white neighbors v of u, port p of v is connected to u. Then all $\Delta = x+y+2$ edges incident to u are labeled with an M. However, by definition of $B_\Delta(x, y)$, at most $x+1$ edges incident to a black node can be labeled with an M.
(2) $w = \mathsf{P}^2\mathsf{O}^y\mathsf{X}^x$: By a similar reasoning, we can find a black node u such that all $\Delta = x+y+2$ edges incident to u are labeled with a P. However, at most $x+y+1$ edges incident to a black node can be labeled with a P. □

3.6 Amplifying through k-matchings

A straightforward application of Corollary 3.2 and Lemma 3.3 already gives a new lower bound—it just is not tight yet:

Theorem 3.4. *Any deterministic algorithm that finds a maximal matching in the port-numbering model in 2-colored graphs requires $\Omega(\sqrt{\Delta})$ rounds.*

Proof. Any algorithm that runs in T rounds implies a white algorithm that runs in $O(T)$ rounds, and naturally a general algorithm also has to solve the problem correctly in trees. Let us assume for a contradiction that for any

constant $c > 0$ and any positive integer Δ_0 there exists $\Delta > \Delta_0$ such that there exists a white deterministic algorithm that finds a maximal matching in the port numbering model in 2-colored graphs in less than $c\sqrt{\Delta}$ rounds.

Recall that $\Pi_\Delta(0,0)$ is equal to the maximal matching problem. If we could solve it with a white algorithm in less than $c\sqrt{\Delta}$ rounds, then by Corollary 3.2 we could also solve $\Pi_\Delta(x', y')$ in 0 rounds, where $x' = c\sqrt{\Delta}$ and $y' = c\sqrt{\Delta}(c\sqrt{\Delta}-1)/2$. For small enough c and large enough Δ_0, $x' + y' + 2 = c\sqrt{\Delta} + c\sqrt{\Delta}(c\sqrt{\Delta} - 1)/2 \le \Delta$, which contradicts Lemma 3.3. □

However, we can easily amplify this and get a linear-in-Δ lower bound. Recall k-matchings from Definition 3.1. We make the following observation:

Lemma 3.5. *If there is an algorithm that finds a k-matching in T rounds, there is also a white algorithm that solves $\Pi_\Delta(k-1, 0)$ in $T + O(1)$ rounds.*

Proof. Consider a k-matching M, and consider a white node u.

(1) If u is incident to at least one edge of M, then pick arbitrarily one $e \in M$ incident to u and label e with an M. Label all other edges $\{u, v\} \in M \setminus \{e\}$ with an X; note that this results in at most $k-1$ incident edges with an X. Then label all other edges $\{u, v\} \notin M$ with O or X such that the total number of incident edges with an X is exactly $k-1$.
(2) Otherwise u is not incident to any edge of M. In this case pick arbitrarily $k-1$ incident edges, label them with an X, and label all other incident edges with a P.

At this point by construction white nodes satisfy $W_\Delta(k-1, 0)$. It remains to be checked that black nodes satisfy $B_\Delta(k-1, 0)$; to this end, consider a black node v. There are two cases:

(1) Node v is incident to at least one edge of M: All edges in M are labeled with M or X and all edges not in M are labeled with P, O, or X. We have got at least one edge incident to v labeled with M or X, at most $k-1$ additional incident edges labeled with M or X, and everything else in [POX], which forms a word in
$$[\mathsf{MX}]\,[\mathsf{POX}]^{\Delta-k}\,[\mathsf{MPOX}]^{k-1}.$$
(2) Node v is not incident to an edge of M: Then by definition all white neighbors u of v were incident to at least one edge of M, and hence all incident edges are labeled with O or X, which forms a word in
$$[\mathsf{OX}]^{\Delta-k+1}\,[\mathsf{MPOX}]^{k-1}.$$ □

Now the same reasoning as Theorem 3.4 gives the following result:

Theorem 3.6. *Any deterministic algorithm that finds an $O(\sqrt{\Delta})$-matching in the port-numbering model in 2-colored graphs requires $\Omega(\sqrt{\Delta})$ rounds.*

Proof. For a contradiction, assume that the theorem statement is not true, which, by Lemma 3.5, implies that there is a T-round algorithm that solves $\Pi_\Delta(x, y)$, for $x = O(\sqrt{\Delta})$, $y = 0$, and $T = o(\sqrt{\Delta})$. By Corollary 3.2, this implies that there is a 0-round algorithm for $\Pi_\Delta(x', y')$, where $x' = O(\sqrt{\Delta})$ and $y' = o(\Delta)$. Note that, for Δ large enough, $\Delta - x' - y'$ is much larger than 2. But by Lemma 3.3, since $\Delta \ge x' + y' + 2$, $\Pi_\Delta(x', y')$ cannot be solved in 0 rounds, and hence we reached a contradiction. □

Now it turns out that Theorem 3.6 is tight, and we can use it to prove a tight lower bound for maximal matchings:

Theorem 3.7. *Any deterministic algorithm that finds a maximal matching in the port-numbering model in 2-colored graphs requires $\Omega(\Delta)$ rounds.*

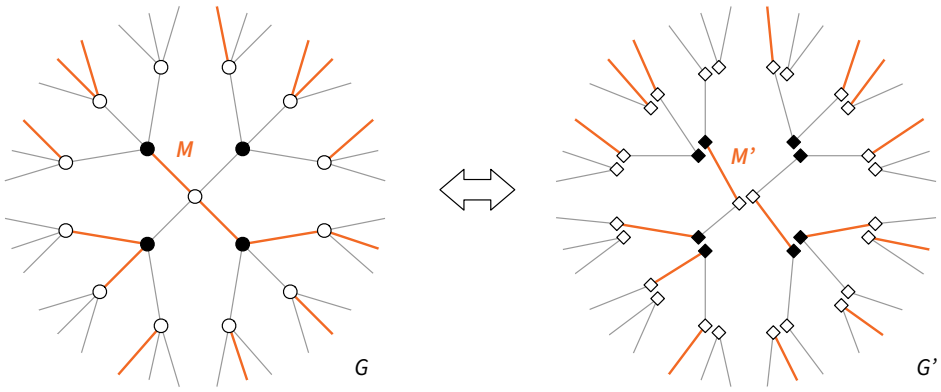


Fig. 7. Splitting nodes of degree $\Delta = 4$ in $\sqrt{\Delta} = 2$ mininodes, each of degree $\sqrt{\Delta} = 2$. A maximal matching M' of G' defines a 2-matching M in G.

Proof. Assume that A is an algorithm that finds a maximal matching in $f(\Delta) = o(\Delta)$ rounds. We use A to construct algorithm A' that finds an $O(\sqrt{\Delta})$-matching in $o(\sqrt{\Delta})$ rounds; then we can apply Theorem 3.6 to see that A cannot exist.

Algorithm A' constructs a new virtual graph G' by splitting each node u of G arbitrarily in $O(\sqrt{\Delta})$ *mininodes* $u_1, u_2, \ldots$, each of degree $O(\sqrt{\Delta})$; see Figure 7. Then A' simulates A in graph G' to find a maximal matching M' of G'. Now M' defines an $O(\sqrt{\Delta})$-matching M in the original graph G. To see this, note that each mininode is incident to at most one edge of M', and hence each original node is incident to $O(\sqrt{\Delta})$ edges of M. Furthermore, if a node u is not incident to any edge of M, then all neighbors of each mininode are matched in M', and hence all original neighbors of u are incident to at least one edge of M.

To conclude the proof, note that the maximum degree of the virtual graph G' is $\Delta' = O(\sqrt{\Delta})$, and hence the simulation of A in G' completes in $f(\Delta') = o(\sqrt{\Delta})$ rounds. □

3.7 Bounds for finite graphs

We now prove the existence of a certain family of graphs, that will be later used to apply the ideas explained in Section 3.2.3.

Lemma 3.8. *There exists a constant $c > 0$ such that for any n large enough and any Δ satisfying that at least one of n and Δ is even and $2 \le \Delta < n/10$, there exists a Δ-regular connected graph $G = (V, E)$ where $|V| = n$ that contains a node $v \in V$ such that the radius-t neighborhood of v is isomorphic to a Δ-regular tree, for some $t \ge c \log_\Delta n$.*

Proof. For $\Delta = 2$ the statement follows by considering a cycle of n nodes, so in the following we assume $\Delta > 2$. A Δ-regular balanced tree of depth t contains $f_\Delta(t) = (\Delta(\Delta-1)^t - 2)/(\Delta - 2)$ nodes. We choose the maximal t such that $n - f_\Delta(t) - (\Delta-1)^t > \Delta$. Note that we always have $t \ge 1$, as $n > 10\Delta$ and $\Delta + f_\Delta(1) + (\Delta-1)^1 = 3\Delta$. We build a Δ-regular balanced tree T of depth t. Tree T has $\Delta(\Delta-1)^{t-1}$ leaves of degree 1, and all the other nodes are of degree Δ. Finally we turn T into a Δ-regular graph: we add $(\Delta-1)^t$ new nodes, called *bottom nodes*, and connect those nodes with the leaves; each bottom node is connected to Δ distinct leaf nodes, and each leaf node is connected to distinct $\Delta - 1$ bottom nodes.

So far we have got $f_\Delta(t) + (\Delta-1)^t$ nodes in graph T; we are still missing $N = n - f_\Delta(t) - (\Delta-1)^t > \Delta$ nodes. Notice that if Δ is odd, $f_\Delta(t)$ and $(\Delta-1)^t$ are even, meaning that N is even. We now use the fact that for any $\Delta \geq 2$ and $N > \Delta$ such that one of them is even, a Δ-regular connected graph H of size N always exists: arrange the N nodes in a cycle; if Δ is even, node i is connected to its $\Delta/2$ predecessors and successors; if Δ is odd, N is even and you can also connect i to the node diametrically opposed.

So far we have got two connected components, T and H, both of them Δ-regular graphs with n nodes in total. In order to connect T and H, we pick an arbitrary edge $\{u, v\}$ of H and an arbitrary edge $\{w, x\}$ adjacent to a bottom node x of T. We remove such edges and add the edges $\{u, w\}$ and $\{v, x\}$.

The obtained graph G contains exactly n nodes, it is Δ-regular, and the t-radius ball around the node corresponding to the root of T is a Δ-regular tree. Moreover, we can notice that $t = \Omega(\log_\Delta f_\Delta(t))$. By the choice of t, we have $f_\Delta(t) - (\Delta-1)^t < n - \Delta < f_\Delta(t+1) - (\Delta-1)^{t+1}$, hence $t = \Omega(\log_\Delta n)$. □

Lemma 3.9. *There exists a constant $c > 0$ such that for any n and Δ large enough satisfying that at least one of n and Δ is even and $\Delta < n/10$, any deterministic algorithm that finds a maximal matching in Δ-regular 2-colored graphs of n nodes requires at least $c \min\{\Delta, \log_\Delta n\}$ rounds in the port-numbering model.*

Proof. We can apply the idea explained in Section 3.2.3 to convert the lower bound of Theorem 3.7, presented for infinite Δ-regular trees, into a lower bound for Δ-regular graphs containing n nodes.

Let us assume for a contradiction that for any constant $c > 0$ and any positive integers n_0 and Δ_0 there exist $n > n_0$ and $\Delta > \Delta_0$ satisfying $\Delta < n/10$ such that there is a deterministic algorithm that finds a maximal matching in Δ-regular 2-colored graphs of n nodes in less than $c \min\{\Delta, \log_\Delta n\}$ rounds.

We fix c to be the minimum of 0.1, the constant c of Lemma 3.8, and the constant hidden in the Ω-notation of Theorem 3.7. We will later fix n_0 and Δ_0 to be large enough constants.

We run this algorithm on an infinite Δ-regular tree, by claiming that the number of nodes is n. Since the running time of the algorithm is less than $c \log_\Delta n$, the algorithm sees at most $\Delta^{c \log_\Delta n} = n^c < n$ nodes, and hence does not detect that we lied about the size of the graph. Since T is also less than $c\Delta$, by Theorem 3.7 the algorithm must fail in some neighborhood, if Δ is large enough. We fix Δ_0 to make Δ satisfy such a requirement.

Now, we fix n_0 large enough and apply Lemma 3.8 to construct a Δ-regular graph of n nodes containing a node with the same neighborhood in which the algorithm failed, and the algorithm must fail in such a graph as well, a contradiction. □

Note that Δ and $\log_\Delta n$ become roughly the same by setting $\Delta \approx \log n / \log\log n$. Hence, if we consider Δ to be an upper bound for the maximum degree, we directly get the following.

Corollary 3.10. *Any deterministic algorithm that finds a maximal matching in 2-colored graphs of maximum degree at most Δ requires $\Omega(\min\{\Delta, \log n / \log\log n\})$ rounds in the port-numbering model.*

The following corollary gives a lower bound for the running time of any algorithm for maximal matching, if we express it solely as a function of n.

Corollary 3.11. *Any deterministic algorithm that finds a maximal matching in Δ-regular 2-colored graphs requires $\Omega(\log n / \log\log n)$ rounds in the port-numbering model.*

We have now come to the conclusion of this section: we have a tight linear-in-Δ lower bound for deterministic algorithms in the port-numbering model. In Section 4 we show how to extend the same argument so that it also covers

randomized algorithms in the usual LOCAL model (and as a simple corollary, also gives a lower bound for MIS). The extension is primarily a matter of technicalities—we will use the same ideas and the same formalism as what we have already used in this section, and then analyze how the probability of a failure increases when we speed up randomized algorithms.

Before presenting the randomized lower bound, we briefly discuss the question of *how* we came up with the right problem family $\Pi_\Delta(x, y)$ that enabled us to complete the lower bound proof—we hope similar ideas might eventually lead to a lower bound for other graph problems besides MM and MIS.

3.8 Behind the scenes: how the right problem family was discovered

Let us now look back at the ingredients of the lower-bound proof. The most interesting part is the speedup simulation of Section 3.4. Here algorithm A_1 follows the standard idea that is similar to what was used already by Linial [32, 33]. Algorithm A_2 is related to the idea of *simplification by maximality* in [12]—this is something we can do without losing any power in the speedup simulation argument. On the other hand, algorithm A_4 merely renames the output labels. Hence algorithm A_3 is the only place in which truly novel ideas are needed.

Now if we already had the insight that the problem family $\Pi_\Delta(x, y)$ defined in (2) is suitable for the purpose of the speedup simulation, then inventing A_3 is not that hard with a bit of trial and error and pattern matching. However, at least to us it was not at all obvious that (2) would be the right relaxation of the maximal matching problem (1).

While some amount of lucky guessing was needed, it was greatly simplified by following this approach:

- We start with the formulation of (1); let us call this problem Π_0. This is an edge labeling problem with 3 labels.
- We apply the automatic speedup simulation framework of [12] to obtain a problem Π_1 that is *exactly* one round faster to solve than Π_0. Then simplify Π_1 as much as possible without losing this property. It turns out that Π_1 is an edge labeling problem with 4 labels.
- Repeat the same process to obtain problem Π_2, which is an edge labeling problem with 6 labels.
- At this point we can see that the structure of problems Π_0, Π_1, and Π_2 is vaguely similar, but the set of labels is rapidly expanding and this also makes the problem description much more difficult to comprehend.
- A key idea is this: given problem Π_2 with 6 labels, we can construct another problem Π_2' that is at least as easy to solve as Π_2 by *identifying some labels* of Π_2. For example, if the set of output labels in Π_2 is $\{1, 2, 3, 4, 5, 6\}$, we could replace both 1 and 2 with a new label 12 and obtain a new problem Π_2' with the alphabet $\{12, 3, 4, 5, 6\}$. Trivially, given an algorithm for solving Π_2 we can also solve Π_2' by remapping the outputs. However, it is not at all obvious that Π_2' is a nontrivial problem.
- Many such identifications result in a problem Π_2' that can be shown to be trivial (e.g. we can construct a problem that solves it in 0 or 1 round). However, by greedily exploring possible identifications we can find a way to map 6 output labels to 4 output labels such that the resulting problem Π_2' still seems to be almost as hard to solve as the maximal matching problem.
- At this point we have obtained the following problems: Π_0 with 3 labels, Π_1 with 4 labels, and Π_2' also with 4 labels. For Π_0 we had labels $\{\mathsf{M}, \mathsf{P}, \mathsf{O}\}$ with a simple interpretation. The next step is to try to rename the labels of Π_1 and Π_2' so that they would also use the familiar labels $\{\mathsf{M}, \mathsf{P}, \mathsf{O}\}$ plus some additional extra label X. It turns out that there is a way to rename the labels so that both Π_1 and Π_2' have some vague resemblance to the original formulation of (1). Here a helpful guidance was to consider posets similar to (3) and (4) that visualize some parts of the problem structure, and try to find a labeling that preserves the structure of the poset.

Now, in essence, (2) is inspired by a relaxation and a generalization of problems Π_0, Π_1, and Π_2' constructed above, and algorithm A_3 captures the key idea of mapping 6 output labels to 4 output labels.

While the idea is arguably vague and difficult to generalize, we highlight some ingredients that turned out to be instrumental:

- We do not try to first guess a family of problems; we apply the speedup simulation framework from [12] in a mechanical manner for a couple of iterations and see what is the family of problems that emerges. Only after that we try to relate it to natural graph problems, such as k-matchings.
- We keep the size of the output alphabet manageable by collapsing multiple distinct labels to one label. However, we allow for some new "unnatural" labels (here: X) that emerge in the process in addition to the "natural" labels that we had in the original problem (here: M, P, O).

While many of the elements in our proof have some flexibility—we often somewhat liberally "round down" and simplify problems in order to keep the proofs and definitions as simple as possible—we are not aware of any way of generalizing (1) to something similar to (2) so that we could use a natural 3-symbol alphabet and still prove a speedup result.

Finally, we point out that the new label X that emerged in the mechanical process was not exactly a wildcard symbol. It behaved a bit like a wildcard in certain contexts, and we simply forced this simple interpretation by relaxing the problem and allowing it to behave like a wildcard in all contexts.

4 LOWER BOUNDS IN THE LOCAL MODEL

In Section 3 we gave a linear-in-Δ lower bound for deterministic distributed algorithms in the port-numbering model. Now we would like to

(1) extend the result from the port-numbering model to the LOCAL model,
(2) extend the result from deterministic algorithms to randomized algorithms.

4.1 LOCAL model

In the area of distributed graph algorithms, there are two widely used models of computing: *LOCAL* and *CONGEST* [40]. The LOCAL model is strictly stronger than the CONGEST model; as we are interested in lower bounds, we will here use the LOCAL model to make our result as widely applicable as possible.

In the LOCAL model, each node is labeled with a *unique identifier*. If there are n nodes in the network, the unique identifiers are some subset of $\{1, 2, \ldots, \mathrm{poly}(n)\}$. Put otherwise, the unique identifiers are $O(\log n)$-bit natural numbers. We assume that the nodes know n, which can be done without loss of generality as we are proving impossibility results.

Other than the assumption of unique identifiers, the model of computing is the same as what we already used in Section 3: nodes are computational entities, edges are communication links, and computation proceeds in synchronous communication rounds. The computational power of each entity and the bandwidth is not limited, that is, each node can send messages of arbitrary size and can perform local computation of arbitrary complexity. Initially, each node knows its own unique identifier. In each round, each node can:

- send an arbitrary message to each of its neighbors;
- receive a message from each neighbor;
- perform some local computation and update its own state.

A distributed algorithm runs in T rounds in the LOCAL model if after T rounds each node has terminated and outputted a local output, and these local outputs constitute a feasible global solution. Since the communication power is not

limited, a distributed algorithm that runs in T rounds in the LOCAL model can be seen as a procedure that does the following at each node v:

(1) gather the topology of the network up to distance T form v;
(2) perform unbounded local computation to produce its local output.

In a *randomized* algorithm the nodes are labeled also with a stream of random bits. We are primarily interested in Monte Carlo algorithms that find a maximal matching with high probability, which we here define so that the running time is fixed and the global success probability is at least $1 - 1/n$.

4.2 High-level plan

We will use a construction similar to Section 3. In particular, we will prove a lower bound for the problem of finding a maximal matching in a 2-colored regular tree. Naturally, the same lower bound then holds also in the general case, even if we do not have a 2-coloring.

Ideally, we would like to first extend the proof so that it holds also with unique identifiers, this way derive a lower bound for *deterministic* algorithms in the LOCAL model, and then later add random input bits to derive a lower bound for *randomized* algorithms in the LOCAL model.

Unfortunately, the speedup simulation technique does not work well with unique identifiers. If we look at e.g. regions D_1 and D_2 in Figure 5, the inputs in these parts are no longer independent: for example, if D_1 contains a node with unique identifier 1, we know that D_2 cannot contain such a node.

Hence, as usual, we take a different route [13]: we first add randomness. We repeat the analysis of Section 3 for randomized Monte Carlo algorithms in the port-numbering model. This way we can still use independence: random bits in D_2 are independent of random bits in D_1.

Once we have a lower bound for Monte Carlo randomized algorithms in the port-numbering model, it is straightforward to turn this into a lower bound for Monte Carlo randomized algorithms in the LOCAL model. To see this, we first observe that the lower bound holds also even if all nodes are labeled with the value of n (such extra information does not change anything in the proof). But now if we know n, it is easy to use randomness to construct $O(\log n)$-bit labels that are unique with high probability. Hence a Monte Carlo randomized algorithm in the LOCAL model can be turned into a Monte Carlo randomized algorithm in the port-numbering model (with knowledge of n), and our lower bound applies. Finally, a lower bound for randomized algorithms in the LOCAL model trivially implies a lower bound also for deterministic algorithms in the LOCAL model.

4.3 White and black randomized algorithms

Let us now extend the concepts of white and black algorithms to white and black *randomized* algorithms. As discussed above, we use the port-numbering model augmented with randomness; in brief, each node is labeled with a random bit string, and in a time-T algorithm, the output of a node u may depend on the random bit strings of all nodes v that are within distance T from u.

We say that A is a *white randomized algorithm* for $\Pi = (W, B)$ with a *local error probability* p if the following holds:

(1) White nodes produce labels for the incident edges, and black nodes produce an empty output.
(2) For each white node, the labels of the incident edges always form a word in W.
(3) For each black node, the labels of the incident edges form a word in B with probability at least $1 - p$.

A *black randomized algorithm* is analogous, with the roles of white and black nodes reversed.

Note that if we are given any Monte Carlo randomized algorithm A that solves Π with high probability in time T, we can always turn it into a white randomized algorithm (or black randomized algorithm) A' that solves Π with a local error probability $1/n$ and running time $T + O(1)$: the local failure probability cannot exceed the global failure probability, and if some white nodes are unhappy (violating constraints in W), we can locally fix it so that all white nodes are happy and only black nodes are unhappy (e.g. all unhappy white nodes simply pick the first word of W). Hence it is sufficient to prove a lower bound for white randomized algorithms.

4.4 Speedup simulation

We will first prove a probabilistic version of Lemma 3.1; the proof follows the same strategy as [13]:

Lemma 4.1. *Assume that there exists a white randomized algorithm that solves $\Pi_\Delta(x, y)$ in $T \geq 1$ rounds in trees, for $\Delta \geq 2x + y + 1$, with local error probability $p \leq 1/4^{\Delta+1}$. Then there exists a white randomized algorithm that solves $\Pi_\Delta(x + 1, y + x)$ in $T - 1$ rounds in trees, with local error probability at most $q = 5\Delta p^{\frac{1}{\Delta+1}}$.*

To prove the lemma, assume that A is the white randomized algorithm with running time T and local error probability p. Let $\alpha \in [0, 1]$ be a parameter that we will fix later.

4.4.1 Preliminaries. Let $N(x, r)$ denote the radius-r neighborhood of node x. Throughout this section, we will use u, u', u_i etc. to refer to black nodes and v, v', v_i etc. to refer to white nodes. For a black node u and a white node v adjacent to u, define

$$U(u) = N(u, T - 1),$$
$$V(v) = N(v, T),$$
$$D(u, v) = V(v) \setminus U(u).$$

With this notation, the regions in Figure 5 are

$$U = U(u), \quad V_i = V(v_i), \quad D_i = D(u, v_i).$$

It will be convenient to assume that the *port numbering is chosen independently and uniformly at random.* If the local error probability is at most p for an adversarial choice of port numbering, it is at most p also for a random port numbering. In what follows, e.g. $U(u)$ refers to all random choices within distance $T - 1$ from u: both the random bit string given to the nodes and the randomly chosen port numbers.

Definition 4.1 (typical labels)**.** We say that x is a *typical* label for edge $\{u, v\}$ w.r.t. $U(u)$ if

$$\Pr_{D(u,v)}\left[A \text{ labels } \{u, v\} \text{ with } x \;\middle|\; U(u)\right] \geq \alpha.$$

Otherwise the label is *atypical*.

Note that as long as $\alpha \leq 1/4$, there is always at least one typical label for each edge (recall that the alphabet size is 4).

Definition 4.2 (lucky neighborhoods)**.** We say that $U(u)$ is *lucky* if the following holds: for each edge $\{u, v\}$ we can pick any typical label and this makes node u happy, i.e., any combination of typical labels forms a word in $B_\Delta(x, y)$. Otherwise, we call $U(u)$ *unlucky*.

Lemma 4.2. *The probability that $U(u)$ is unlucky is at most p/α^Δ.*

Proof. If $U(u)$ is unlucky, then for each edge $e_i = \{u, v_i\}$ we can choose a typical edge label x_i so that u is unhappy. By definition, given $U(u)$, algorithm A will output x_i on edge e_i with probability at least α, and this only depends on $D(u, v_i)$. As the regions $D(u, v_i)$ are independent, we see that u is unhappy with probability at least α^Δ.

Let β be the probability that $U(u)$ is unlucky. Node u is then unhappy with probability at least $\beta\alpha^\Delta$, which is at most p by assumption. □

Definition 4.3 (nice neighborhoods)**.** We say that $V(v)$ is *nice* if the following holds for each edge $e = \{v, u\}$ incident to v: the label assigned by algorithm A to e is a typical label for $\{u, v\}$ w.r.t. $U(u)$. Otherwise $V(v)$ is *bad*.

Lemma 4.3. *The probability that $V(v)$ is bad is at most $4\Delta\alpha$.*

Proof. There are Δ edges incident to v; let us focus on one of them, say $e = \{v, u\}$. We next analyze the probability that e has an atypical label.

We first fix $U(u)$. Then we see which labels are atypical for e w.r.t. $U(u)$. Now consider each possible label x; if x is atypical, then the probability that A outputs x for e given $U(u)$ is less than α. Summing over all possible atypical labels (at most 4) and all possible choices of $U(u)$, the probability that the output of A for e is atypical is at most 4α.

The claim follows by union bound. □

Definition 4.4 (friendly neighborhoods)**.** We say that $V(v)$ is *friendly* if:

(1) $V(v)$ is nice,
(2) $U(u)$ is lucky for each black neighbor u of v.

Otherwise $V(v)$ is *unfriendly*.

Lemma 4.4. *The probability that $V(v)$ is unfriendly is at most $\Delta(4\alpha + p/\alpha^\Delta)$.*

Proof. There are $\Delta + 1$ bad events: one possible bad neighborhood, with probability at most $4\Delta\alpha$, and Δ possible unlucky neighborhoods, each with probability at most p/α^Δ. By union bound, we have an unfriendly neighborhood with probability at most $\Delta(4\alpha + p/\alpha^\Delta)$. □

Now if we choose $\alpha = p^{\frac{1}{\Delta+1}} \le 1/4$, we obtain:

Corollary 4.5. *The probability that $V(v)$ is unfriendly is at most $q = 5\Delta p^{\frac{1}{\Delta+1}}$.*

4.4.2 Speedup simulation for friendly neighborhoods. We will first look at speedup simulation in friendly neighborhoods. The high-level plan is this:

- Speedup simulation is well-defined for each black node u such that $U(u)$ is lucky.
- Speedup simulation results in a good output for each white node v such that $V(v)$ is friendly.

Using the notation of Section 3.4, we redefine A_1 as follows:

Set S_i consists of all typical labels for e_i w.r.t. $U(u)$.

Let us now look at how to extend algorithms A_2 to A_4 and their analysis to the probabilistic setting, assuming a friendly neighborhood:

- Algorithm A_2: The mapping from 15 sets to 6 labels is identical to the deterministic version.

- Output of A_2: The analysis holds verbatim, as we assumed that u is lucky and hence any combination of typical labels has to make u happy.
- Algorithm A_3: The mapping from 6 sets to 4 labels is identical to the deterministic version.
- Algorithm A_4: The mapping from 4 labels to 4 labels is identical to the deterministic version.
- Output of A_4: As we look at a white node in a friendly neighborhood, we know that the output of A is typical, and hence the output of A is contained in the sets of typical labels that A_1 outputs. The rest of the analysis holds verbatim.

4.4.3 Speedup simulation for all neighborhoods. Let us now address the case of unfriendly neighborhoods. We modify A_4 so that if node u is unlucky, we produce some arbitrary fixed output from $W_\Delta(x+1, y+x)$. This way A_4 is always well-defined and the output of a black node is always in $W_\Delta(x+1, y+x)$.

Furthermore, we know that the labels incident to a white node v are in $B_\Delta(x+1, y+x)$ whenever $V(v)$ is friendly, and this happens with probability at least $1-q$. We conclude that A_4 is a black randomized algorithm that solves $\Pi'_\Delta(x+1, y+x)$ with local error probability at most q.

Finally, we obtain a white randomized algorithm for $\Pi_\Delta(x+1, y+x)$ by reversing the roles of white and black nodes. This concludes the proof of Lemma 4.1.

4.5 Multiple speedup steps

By a repeated application of Lemma 4.1, we obtain the following corollary that is analogous to Corollary 3.2:

Corollary 4.6. *Assume that there exists a white randomized algorithm that solves $\Pi_\Delta(x, y)$ in T rounds in trees with local error probability p, for $\Delta \geq x + y + T(x + 1 + (T-1)/2)$. Then there is a white randomized algorithm that solves $\Pi_\Delta(x', y')$ in 0 rounds in trees with local error probability p' for*

$$\begin{aligned} x' &= x + T, \\ y' &= y + T\big(x + (T-1)/2\big), \\ p' &\leq (5\Delta)^2 p^{1/(\Delta+1)^T}. \end{aligned}$$

Proof. Let p_i be the local error probability after applying Lemma 4.1 for i times, with $p_0 = p$ and $p_T = p'$. We prove by induction that

$$p_i \leq (5\Delta)^2 p^{1/(\Delta+1)^i}.$$

The base case is $p_0 \leq (5\Delta)^2 p$, which trivially holds. For the inductive step, note that

$$\begin{aligned} p_{i+1} &\leq 5\Delta \cdot p_i^{1/(\Delta+1)} \\ &\leq 5\Delta \cdot \left((5\Delta)^2 p^{1/(\Delta+1)^i}\right)^{1/(\Delta+1)} \\ &\leq (5\Delta)^2 \cdot p^{1/(\Delta+1)^{i+1}}. \end{aligned}$$

□

4.6 Base case

Lemma 4.7. *There is no white randomized algorithm that solves $\Pi_\Delta(x, y)$, for $x, y \leq \Delta/8$ and $\Delta \geq 8$, in 0 rounds in trees with local error probability $p \leq 1/\Delta^\Delta$.*

Proof. Fix a white randomized algorithm A that runs in 0 rounds. As each white node has no knowledge of their neighborhood, they all have the same probability q to choose the output $\mathsf{M}\mathsf{O}^{\Delta-x-1}X^x$, and probability $1-q$ to output $\mathsf{P}^{\Delta-x-y}\mathsf{O}^y\mathsf{X}^x$. There are two cases:

- $q \geq \frac{1}{2}$: Consider a black node u with Δ white neighbors $v_1, v_2, \ldots, v_\Delta$. As the port numbering of v_i is chosen uniformly at random, the probability that v_i labels the edge $\{v_i, u\}$ with M is $q/\Delta \geq 1/(2\Delta)$. Now if the first $x+2$ white neighbors output M on the connecting edge, node u will be unhappy, and this happens with probability at least
$$\frac{1}{(2\Delta)^{x+2}} \geq \frac{1}{(2\Delta)^{\Delta/8+2}} \geq \frac{1}{\Delta^\Delta}.$$
- $q \leq \frac{1}{2}$: Now a black node u is unhappy if all white neighbors produce an output of type $\mathsf{P}^{\Delta-x-y}\mathsf{O}^y\mathsf{X}^x$, and furthermore the first $x+y+1$ neighbors output P on the connecting edge; this happens with probability at least
$$\frac{1}{2^\Delta} \cdot \frac{1}{\Delta^{x+y+1}} \geq \frac{1}{2^\Delta} \cdot \frac{1}{\Delta^{\Delta/4+1}} \geq \frac{1}{\Delta^\Delta}.$$ □

4.7 Putting things together

Lemma 4.8. *For a sufficiently large Δ, there is no white algorithm that solves $\Pi_\Delta(x, 0)$ for $x \leq \sqrt{\Delta}$ in $T \leq \sqrt{\Delta}/16$ rounds in trees with local error probability $p < 2^{-\Delta^{2T+2}}$.*

Proof. Assume that A solves $\Pi_\Delta(x, 0)$ in $T \leq \sqrt{\Delta}/16$ rounds with local error probability p. By Corollary 4.6, we obtain a white randomized algorithm A' that solves $\Pi_\Delta(x', y')$ in 0 rounds with local error probability p', where
$$\begin{aligned} x' &= x + T \leq \frac{17}{16}\sqrt{\Delta} \leq \frac{1}{8}\Delta, \\ y' &= y + T\big(x + (T-1)/2\big) \leq \frac{33}{512}\Delta \leq \frac{1}{8}\Delta, \\ p' &\leq (5\Delta)^2 p^{1/(\Delta+1)^T}. \end{aligned}$$
By Lemma 4.7 we have $p' > 1/\Delta^\Delta$ and therefore
$$p > \frac{1}{\big((5\Delta)^2\Delta^\Delta\big)^{(\Delta+1)^T}} > \frac{1}{2^{\Delta^{2T+2}}}.$$ □

Theorem 4.9. *There exists a constant $c > 0$ such that for any n and Δ large enough satisfying that at least one of n and Δ is even and $\Delta < n/10$, any algorithm that finds a $\sqrt{\Delta}$-matching in Δ-regular graphs of n nodes with probability at least $1 - 1/n$ requires at least $c \min\{\sqrt{\Delta}, \log_\Delta \log n\}$ rounds.*

Proof. Let us assume for a contradiction that for any constant $c > 0$ and any positive integers n_0 and Δ_0 there exist $n > n_0$ and $\Delta > \Delta_0$ satisfying $\Delta < n/10$ such that there is an algorithm that finds a $\sqrt{\Delta}$-matching in Δ-regular graphs of n nodes in less than $c \min\{\sqrt{\Delta}, \log_\Delta \log n\}$ rounds with probability at least $1 - 1/n$.

By Lemma 3.5 there is then a white randomized algorithm A that solves $\Pi_\Delta(\sqrt{\Delta}, 0)$ with only an additional constant overhead. Let $\bar{c}$ be the largest $c \leq 0.1$ such that the obtained algorithm A runs in $T < \sqrt{\Delta}/16$ rounds.

We fix c to be the minimum of $\bar{c}$ and the constant c of Lemma 3.8. Then, we fix Δ_0 as the minimum value required to apply Lemma 4.8, and n_0 as the minimum value required to apply Lemma 3.8. We then apply Lemma 3.8 to construct a Δ-regular graph of n nodes that contains a node v such that the radius-t neighborhood of v is isomorphic to a Δ-regular tree, for some $t \geq c \log_\Delta n$. Since $T < c \log_\Delta \log n \leq t$ and $T < \sqrt{\Delta}/16$, by Lemma 4.8, algorithm A fails in such a

neighborhood with probability at least

$$\frac{1}{2^{\Delta^{2T+2}}} > \frac{1}{2^{\Delta^{\log_\Delta \log n}}} = \frac{1}{n}.$$

Hence the global success probability cannot be $1 - 1/n$. □

Now the same idea as in Theorem 3.7 gives our main result:

Theorem 4.10. *There exists a constant $c > 0$ such that for any n and Δ large enough satisfying that at least one of n and Δ is even and $\Delta < n/10$, any LOCAL-model algorithm that finds a maximal matching in Δ-regular graphs of n nodes with probability at least $1 - 1/n$ requires at least $c \min\{\Delta, \log_\Delta \log n\}$ rounds.*

Note that Δ and $\log_\Delta \log n$ become roughly the same by setting $\Delta \approx \log\log n / \log\log\log n$. Hence, if we consider Δ to be an upper bound of the maximum degree, we directly get the following.

Corollary 4.11. *Any LOCAL-model algorithm that finds a maximal matching in graphs of maximum degree at most Δ with probability at least $1 - 1/n$ requires $\Omega(\min\{\Delta, \log\log n / \log\log\log n\})$ rounds.*

The following corollary gives a lower bound for the running time of any algorithm for maximal matching, if we express it solely as a function of n.

Corollary 4.12. *Any LOCAL-model algorithm that finds a maximal matching in Δ-regular graphs with probability at least $1 - 1/n$ requires $\Omega(\log\log n / \log\log\log n)$ rounds.*

4.8 Deterministic lower bound via speedup simulation

Our randomized lower bound implies also a stronger deterministic lower bound. The proof is again based on a speedup argument: we show that the existence of a (too) fast deterministic algorithm implies the existence of an even faster deterministic algorithm. Using the simple observation that the randomized complexity of a problem is at most its deterministic complexity, we will then obtain a contradiction with Theorem 4.10. Our proof is similar to Chang et al. [16, Theorem 6], and the coloring algorithm follows the idea of Barenboim et al. [11, Remark 3.6].

Theorem 4.13. *There exists a constant $c > 0$ such that for any n and Δ large enough satisfying that at least one of n and Δ is even and $\Delta < n/10$, any deterministic LOCAL-model algorithm that finds a maximal matching in Δ-regular graphs of n nodes requires at least $c \min\{\Delta, \log_\Delta n\}$ rounds.*

Proof. Let us assume for a contradiction that for any constant $c > 0$ and any positive integers N_0 and Δ_0 there exist $N > N_0$ and $\Delta > \Delta_0$ satisfying $\Delta < N/10$ such that there is a deterministic algorithm A that finds a maximal matching in Δ-regular graphs of N nodes in less than $T = c \min\{\Delta, \log_\Delta N\}$ rounds. We use this assumption to create a randomized algorithm that violates Theorem 4.10.

Let $\hat{c}$, N_0, and Δ_0 be, respectively, the constant c, the minimum size n, and the minimum degree Δ, required to apply Theorem 4.10. Let $\bar{c}$ be the minimum of 0.1 and $\hat{c}$. Let $c = \epsilon\bar{c}$ for some positive constant ϵ to be fixed later. We get that there exists an algorithm A that finds a maximal matching in Δ-regular graphs of N nodes in less than $T = c \min\{\Delta, \log_\Delta N\}$ rounds, for some specific $N > N_0$ and $\Delta > \Delta_0$. We show how to construct an algorithm A' that is able to find a maximal matching in any Δ-regular graphs of $n = 2^N$ nodes in less than $\epsilon^{-1}T$ rounds. Note that $T = c \min\{\Delta, \log_\Delta N\} = c \min\{\Delta, \log_\Delta \log n\}$. This implies that we can solve maximal matching on Δ-regular graphs ($\Delta > \Delta_0$) of size $n = 2^N > N_0$ in $\epsilon^{-1}T = \bar{c} \min\{\Delta, \log_\Delta \log n\}$, a contradiction with Theorem 4.10.

Let G be any Δ-regular graph of n nodes. We show how to simulate A on G, for any possible given assignment of unique identifiers in $\{1, \ldots, \mathrm{poly}(n)\}$ for the nodes of G. In order to simulate A we need to compute a labeling that *locally* looks like an assignment of unique identifiers of size N, and we need to make sure that each neighborhood contains less than N nodes. We compute an N-coloring of G^{2T+2}, the $(2T+2)$th power of G, using three iterations of Linial's coloring algorithm [33, Corollary 4.1]. This algorithm can be used to color a k-colored graph of maximum degree $\bar{\Delta}$ with e.g.

$$O\big(\bar{\Delta}^2\big(\log\log\log k + \log\bar{\Delta}\big)\big)$$

colors in three rounds. The power graph G^{2T+2} has maximum degree $\bar{\Delta} \le \Delta^{2T+2}$ and thus we get a coloring of G^{2T+2} with

$$O\big(\Delta^{2(2T+2)}\big(\log\log N + \log\Delta^{2T+2}\big)\big)$$

colors. Since $T \le c \log_\Delta N$, we see that each T-radius neighborhood contains $O(\Delta^T) < N$ nodes and that the number of colors is less than N. The coloring can be computed in time $O(T)$ in the original graph G.

Finally we simulate A on the computed coloring. If we look at the radius-$(T+1)$ neighborhood of any node in G, the number of nodes is less than N, and the coloring looks like an assignment of unique identifiers from $\{1, 2, \ldots, N\}$, and thus the output of A is well defined and correct by a standard argument—see e.g. [16, Theorem 6]. Furthermore, the simulation of A on G can be done in T rounds. Our simulation runs in $O(T)$ rounds on total.

We can easily transform this deterministic algorithm into a randomized algorithm with the same runtime on the same graph class: in the beginning, each node simply picks a random value from $\{1, 2, \ldots, n^3\}$, interprets its value as an identifier, and then runs the deterministic simulation with these identifiers. The picked values are globally unique with probability at least $1 - 1/n$, guaranteeing that the randomized algorithm is correct w.h.p. As previously stated, the obtained running time gives a contradiction with Theorem 4.10. □

By setting $\Delta \approx \log n / \log\log n$ in Theorem 4.13 we directly get the following.

Corollary 4.14. *Any deterministic LOCAL-model algorithm that finds a maximal matching in Δ-regular graphs requires $\Omega(\log n / \log\log n)$ rounds.*

If we consider Δ to be an upper bound of the maximum degree, we directly get the following.

Corollary 4.15. *Any deterministic LOCAL-model algorithm that finds a maximal matching in graphs of maximum degree Δ requires $\Omega(\min\{\Delta, \log n / \log\log n\})$ rounds.*

4.9 Lower bounds for MIS

If we have any algorithm that finds an MIS, we can simulate it in the line graph and obtain an algorithm for MM; the simulation overhead is constant and the increase in the maximum degree is also bounded by a constant factor. We obtain:

Corollary 4.16. *Any LOCAL-model algorithm that finds a maximal independent set in graphs of maximum degree Δ with probability at least $1 - 1/n$ requires $\Omega(\min\{\Delta, \log\log n / \log\log\log n\})$ rounds.*

Corollary 4.17. *Any deterministic LOCAL-model algorithm that finds a maximal independent set in graphs of maximum degree Δ requires $\Omega(\min\{\Delta, \log n / \log\log n\})$ rounds.*

4.10 Matching upper bounds

Recall the bound from Theorem 4.10: it excludes the possibility of randomized distributed algorithms that find a maximal matching in $o(\Delta) + o(\log\log n/\log\log\log n)$ rounds in general. The first term is optimal in general: there are deterministic and randomized algorithms that find a maximal matching in $O(\Delta) + o(\log\log n/\log\log\log n)$ rounds [38]. The second term is optimal for randomized MM algorithms in trees: there is a randomized algorithm that finds a maximal matching in trees in $o(\Delta) + O(\log\log n/\log\log\log n)$ rounds [11, Theorem 7.3]. Note that the construction of our lower bounds ensures that they already hold in trees.

5 DISCUSSION

We have learned that the prior algorithms for MM and MIS with a running time of $O(\Delta + \log^* n)$ rounds [9, 38] are optimal for a wide range of parameters: in order to solve MM or MIS in time $o(\Delta) + g(n)$, we must increase $g(n)$ from $O(\log^* n)$ all the way close to $O(\log\log n)$.

A priori, one might have expected that we should be able to achieve a smooth transition for MM algorithms between $O(\Delta + \log^* n)$ and $O(\log\Delta + \log^c \log n)$, which are the currently best known complexities as a function of n and Δ, respectively. However, this turned out to be not the case. We conjecture that the qualitative gap between $\log^* n$ and $\log^c \log n$ regions is closely related to similar gaps in the landscape of locally checkable labeling problems [16].

5.1 Open questions

After the preliminary conference version of this work, one of the key open questions was related to the complexity of maximal independent set in the region $\Delta \gg \log\log n$, but thanks to the recent work by Rozhoň and Ghaffari [41], this question is now largely settled. We now know that the maximal independent set problem cannot be much harder to solve than the maximal matching problem.

However, the complexity of vertex coloring and edge coloring is still wide open. Here is one concrete example of a big open question, closely related to the theme of the current work: Is it possible to find a $(\Delta + 1)$-vertex coloring in $O(\log\Delta + \log^* n)$ rounds, for all Δ?

Several more fine-grained questions related to the complexity of MM still remain: For example, we proved that MM requires $\Omega(\min\{\Delta, \log\log n/\log\log\log n\})$ rounds in trees, even for randomized algorithms, and the dependency on n is also tight in trees [11]. In general graphs the current randomized algorithms take $O(\log\Delta + \log^c \log n)$ rounds for $c > 1$. A natural open question is whether or not MM in general graphs is strictly harder than MM in trees, if we allow the dependency on Δ to be $O(\log\Delta)$. Proving such a separation through the speedup simulation technique would require one to be able to handle cycles, which is an interesting open question by itself. Also, when we consider algorithms that take $\operatorname{poly}\log\log n$ rounds as a function of n, there is a gap in the complexity as a function of Δ: the current lower bound is $\Omega(\log\Delta/\log\log\Delta)$, while the upper bound is $O(\log\Delta)$; it is wide open whether the speedup simulation technique is applicable in the study of such questions.

5.2 Recent follow-up work

Computers played a key role in the present work especially in the discovery of an appropriate problem family (2): speedup simulation is a fairly laborious but mechanical process and hence well suited to be automated. The computer program that we used to help us with this project has been now further developed into a freely available open-source

web application known as Round Eliminator [37], and it has already been used to assist in the discovery of several other lower and upper bounds, see e.g. [3, 5, 14].

Recent work has also led to an even more fine-grained characterization of the distributed computational complexity of maximal matchings. Recall the trivial algorithm for bipartite maximal matching from Section 3; one can verify that the round complexity of the algorithm is exactly $2\Delta - 1$. While the present work showed that this is asymptotically optimal, the latest follow-up work [14] showed that this is exactly optimal: the problem cannot be solved in $2\Delta - 2$ rounds.

ACKNOWLEDGMENTS

This work is an extended and revised version of a preliminary conference report that appeared in the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019).

We would like to thank Mohsen Ghaffari for pointing out that the randomized lower bound implies a better deterministic lower bound, and Seth Pettie for pointing out that the bound is tight for trees, as well as for many other helpful comments. Many thanks also to Laurent Feuilloley and Tuomo Lempiäinen for comments and discussions, and to the reviewers of the previous versions of this work for their numerous helpful suggestions.

This work was supported in part by the Academy of Finland, Grants 285721 and 314888 and the Project ESTATE (ANR-16-CE25-0009-03).

REFERENCES

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

[2] Matti Åstrand and Jukka Suomela. 2010. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010)*. ACM Press, 294–302. https://doi.org/10.1145/1810479.1810533

[3] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. 2019. Classification of distributed binary labeling problems. arXiv:1911.13294 http://arxiv.org/abs/1911.13294

[4] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. 2019. Lower bounds for maximal matchings and maximal independent sets. In *Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*. IEEE, 481–497. https://doi.org/10.1109/FOCS.2019.00037 arXiv:1901.02441

[5] Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. 2020. Distributed Lower Bounds for Ruling Sets. arXiv:2004.08282 http://arxiv.org/abs/2004.08282

[6] Alkida Balliu, Juho Hirvonen, Dennis Olivetti, and Jukka Suomela. 2019. Hardness of Minimal Symmetry Breaking in Distributed Computing. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*. ACM Press, 369–378. https://doi.org/10.1145/3293611.3331605 arXiv:1811.01643

[7] Leonid Barenboim. 2016. Deterministic (Δ+1)-Coloring in Sublinear (in Δ) Time in Static, Dynamic, and Faulty Networks. *J. ACM* 63, 5 (2016), 1–22. https://doi.org/10.1145/2979675

[8] Leonid Barenboim and Michael Elkin. 2013. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Vol. 4. https://doi.org/10.2200/S00520ED1V01Y201307DCT011

[9] Leonid Barenboim, Michael Elkin, and Fabian Kuhn. 2014. Distributed (Δ+1)-Coloring in Linear (in Δ) Time. *SIAM J. Comput.* 43, 1 (2014), 72–95. https://doi.org/10.1137/12088848X

[10] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2012. The Locality of Distributed Symmetry Breaking. In *Proc. 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*. IEEE, 321–330. https://doi.org/10.1109/FOCS.2012.60

[11] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2016. The Locality of Distributed Symmetry Breaking. *J. ACM* 63, 3 (2016), 1–45. https://doi.org/10.1145/2903137

[12] Sebastian Brandt. 2019. An Automatic Speedup Theorem for Distributed Problems. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*. ACM Press, 379–388. https://doi.org/10.1145/3293611.3331611 arXiv:1902.09958

[13] Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. 2016. A lower bound for the distributed Lovász local lemma. In *Proc. 48th ACM Symposium on Theory of Computing (STOC 2016)*. ACM Press, 479–488. https://doi.org/10.1145/2897518.2897570

[14] Sebastian Brandt and Dennis Olivetti. 2020. Truly Tight-in-Δ Bounds for Bipartite Maximal Matching and Variants. arXiv:2002.08216 http://arxiv.org/abs/2002.08216

[15] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. 2018. The Complexity of Distributed Edge Coloring with Small Palettes. In *Proc. 29th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. Society for Industrial and Applied Mathematics, 2633–2652. https://doi.org/10.1137/1.9781611975031.168

[16] Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. 2016. An Exponential Separation between Randomized and Deterministic Complexity in the LOCAL Model. In *Proc. 57th IEEE Symposium on Foundations of Computer Science (FOCS 2016)*. IEEE, 615–624. https://doi.org/10.1109/FOCS.2016.72

[17] Corinna Coupette and Christoph Lenzen. 2020. A Breezing Proof of the KMW Bound. arXiv:2002.06005 http://arxiv.org/abs/2002.06005

[18] Manuela Fischer. 2017. Improved Deterministic Distributed Matching via Rounding. In *Proc. 31st International Symposium on Distributed Computing (DISC 2017)*. 17:1–17:15. https://doi.org/10.4230/LIPIcs.DISC.2017.17

[19] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. 2016. Local Conflict Coloring. In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2016)*. IEEE, 625–634. https://doi.org/10.1109/FOCS.2016.73

[20] Cyril Gavoille, Ralf Klasing, Adrian Kosowski, Łukasz Kuszner, and Alfredo Navarra. 2009. On the complexity of distributed graph coloring with local minimality constraints. *Networks* 54, 1 (2009), 12–19. https://doi.org/10.1002/net.20293

[21] Mohsen Ghaffari. 2016. An Improved Distributed Algorithm for Maximal Independent Set. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 270–277. https://doi.org/10.1137/1.9781611974331.ch20

[22] Mohsen Ghaffari. 2017. LOCAL Algorithms: The Chasm Between Deterministic and Randomized. In *6th Workshop on Advances in Distributed Graph Algorithms (ADGA 2017)*. http://adga.hiit.fi/2017/ghaffari.pdf

[23] Mika Göös, Juho Hirvonen, and Jukka Suomela. 2017. Linear-in-Δ lower bounds in the LOCAL model. *Distributed Computing* 30, 5 (2017), 325–338. https://doi.org/10.1007/s00446-015-0245-8

[24] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. 1998. On the Distributed Complexity of Computing Maximal Matchings. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*. ACM/SIAM, 219–225.

[25] Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. 2001. On the Distributed Complexity of Computing Maximal Matchings. *SIAM Journal on Discrete Mathematics* 15, 1 (2001), 41–57. https://doi.org/10.1137/S0895480100373121

[26] Juho Hirvonen and Jukka Suomela. 2012. Distributed maximal matching: greedy is optimal. In *Proc. 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2012)*. ACM Press, 165–174. https://doi.org/10.1145/2332432.2332464

[27] Amos Israeli and A. Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. *Inform. Process. Lett.* 22, 2 (1986), 77–80. https://doi.org/10.1016/0020-0190(86)90144-4

[28] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2004. What cannot be computed locally!. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2004)*. ACM Press, New York, New York, USA, 300–309. https://doi.org/10.1145/1011767.1011811

[29] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2006. The price of being near-sighted. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*. ACM Press, New York, New York, USA, 980–989. https://doi.org/10.1145/1109557.1109666

[30] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2016. Local Computation: Lower and Upper Bounds. *J. ACM* 63, 2 (2016), 1–44. https://doi.org/10.1145/2742012

[31] Fabian Kuhn and Rogert Wattenhofer. 2006. On the complexity of distributed graph coloring. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006)*. 7–15. https://doi.org/10.1145/1146381.1146387

[32] Nathan Linial. 1987. Distributive graph algorithms – Global solutions from local data. In *Proc. 28th Annual Symposium on Foundations of Computer Science (FOCS 1987)*. IEEE, 331–335. https://doi.org/10.1109/SFCS.1987.20

[33] Nathan Linial. 1992. Locality in Distributed Graph Algorithms. *SIAM J. Comput.* 21, 1 (1992), 193–201. https://doi.org/10.1137/0221015

[34] Michael Luby. 1985. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th Annual ACM Symposium on Theory of Computing (STOC 1985)*. ACM Press, New York, New York, USA, 1–10. https://doi.org/10.1145/22145.22146

[35] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053. https://doi.org/10.1137/0215074

[36] Moni Naor. 1991. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics* 4, 3 (1991), 409–412. https://doi.org/10.1137/0404036

[37] Dennis Olivetti. 2020. Round Eliminator: a tool for automatic speedup simulation. https://github.com/olidennis/round-eliminator

[38] Alessandro Panconesi and Romeo Rizzi. 2001. Some simple distributed algorithms for sparse networks. *Distributed Computing* 14, 2 (2001), 97–100. https://doi.org/10.1007/PL00008932

[39] Alessandro Panconesi and Aravind Srinivasan. 1996. On the Complexity of Distributed Network Decomposition. *Journal of Algorithms* 20, 2 (1996), 356–374. https://doi.org/10.1006/jagm.1996.0017

[40] David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719772

[41] Václav Rozhoň and Mohsen Ghaffari. 2020. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proc. 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*. https://doi.org/doi.org/10.1145/3357713.3384298 arXiv:1907.10937

[42] Jukka Suomela. 2014. Lower Bounds for Local Algorithms. In *3rd Workshop on Advances in Distributed Graph Algorithms (ADGA 2014)*. http://adga2014.hiit.fi/jukka.pdf

[43] Márió Szegedy and Sundar Vishwanathan. 1993. Locality based graph coloring. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC 1993)*. 201–207. https://doi.org/10.1145/167088.167156