Lower Bounds for Maximal Matchings and Maximal Independent Sets

Alkida Balliu[*], Sebastian Brandt[†], Juho Hirvonen[*], Dennis Olivetti[*], Mikaël Rabie[*‡], Jukka Suomela[*]

[*]*Aalto University, Finland*
[†]*ETH Zurich, Switzerland*
[‡]*LIP6 - Sorbonne Université, France*

Abstract—**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 maximal matchings and maximal independent sets cannot be found in $o(\Delta + \log \log n / \log \log \log n)$ rounds with any randomized algorithm in the LOCAL model of distributed computing.

As a corollary, it follows that there is no deterministic algorithm for maximal matchings or maximal independent sets that runs in $o(\Delta + \log n / \log \log n)$ rounds; this is an improvement over prior lower bounds also as a function of n.

Keywords—**Maximal matching; maximal independent set; distributed graph algorithms; lower bounds.**

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 [16]: *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 [27, 33] of distributed computing, the fastest known algorithms for these problems are:

- MM is possible in $O(\Delta + \log^* n)$ rounds [31].
- MIS is possible in $O(\Delta + \log^* n)$ rounds [7].

Nowadays we know how to find a vertex or edge coloring with $O(\Delta)$ colors in $o(\Delta) + O(\log^* n)$ rounds [4, 14]. 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. The second part is responsible for the $O(\Delta)$ term in the running time, and previously we had no idea if this is necessary.

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 [26, 27, 30].
- MIS is not possible in $f(\Delta) + o(\log^* n)$ rounds for any f [26, 27, 30].

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 [5, 34]. 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 $\Omega(\log \Delta / \log \log \Delta)$ [23–25].

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:

> There is no randomized distributed algorithm that solves MM or MIS in $o\left(\Delta + \frac{\log \log n}{\log \log \log n}\right)$ rounds in the LOCAL model (with high probability).
>
> There is no deterministic distributed algorithm that solves MM or MIS in $o\left(\Delta + \frac{\log n}{\log \log n}\right)$ rounds in the LOCAL model.

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

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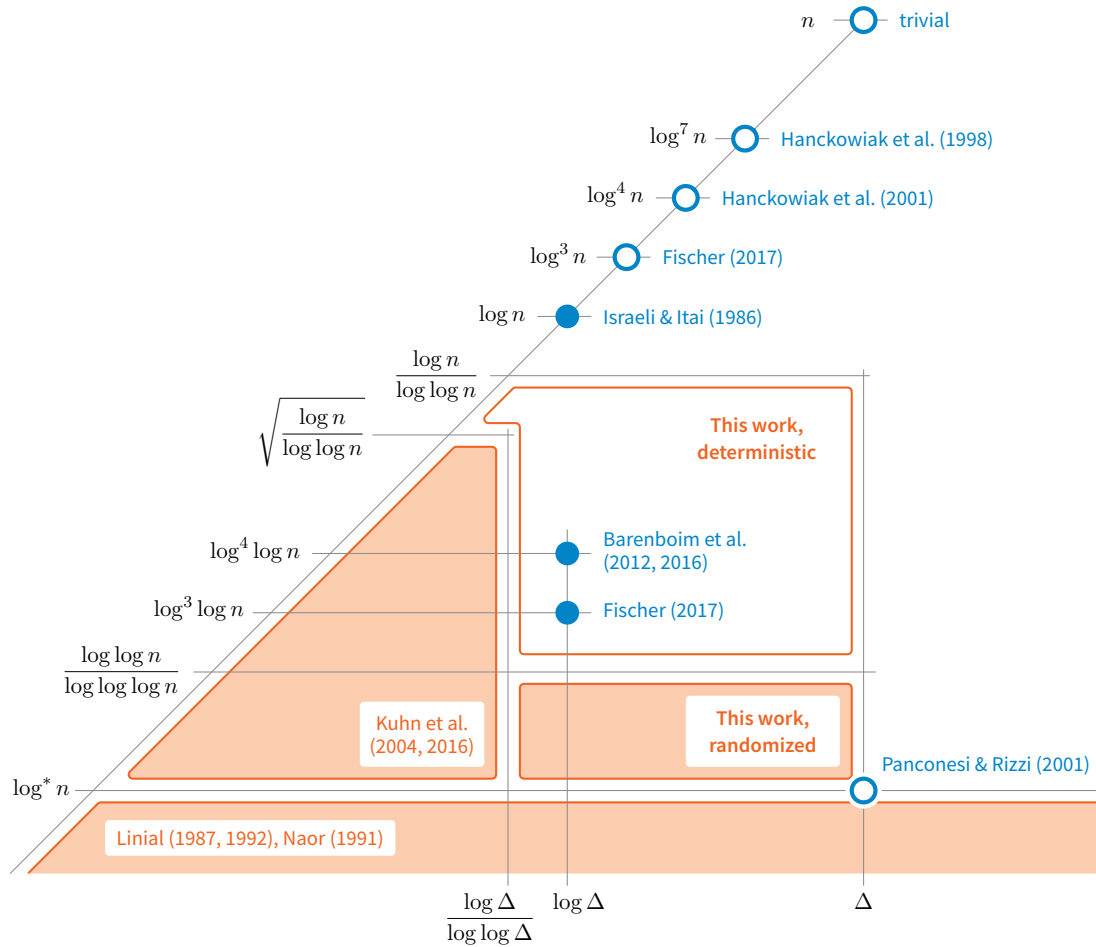- MM and MIS are exactly as hard as greedy coloring.

Figure 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.

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.

2. Related work

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 for a small Δ, with a complexity of the form $f(\Delta) + O(\log^* n)$.
- Deterministic algorithms for a large Δ, with super-logarithmic complexities as a function of n.
- Randomized algorithms for a large Δ, with subloga-rithmic 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 [28, 29], Israeli and Itai [22], Hanckowiak et al. [19, 20], and Barenboim et al. [6, 8] for more prior work on maximal matchings and maximal independent sets.

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

- Linial [26, 27] and Naor [30] show that there is no deterministic or randomized algorithm for MM or MIS that runs in $o(\log^* n)$ rounds, even if we have $\Delta = 2$.
- Kuhn et al. [23, 24, 25] show that there is no deter-ministic or randomized algorithm for MM or MIS that runs in $o\left(\log \Delta / \log \log \Delta + \sqrt{\log n / \log \log n}\right)$ 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.

TABLE 1. EFFICIENT ALGORITHMS FOR MM AND MIS.

Problem	Δ	Randomness	Complexity	Reference
MM	small	deterministic	$O(\Delta + \log^* n)$	Panconesi and Rizzi [31]
	large	deterministic	$O(\log^2 \Delta \cdot \log n)$	Fischer [13]
	large	randomized	$O(\log \Delta + \log^3 \log n)$	Barenboim et al. [6, 8], Fischer [13]
MIS	small	deterministic	$O(\Delta + \log^* n)$	Barenboim et al. [7]
	large	deterministic	$2^{O(\sqrt{\log n})}$	Panconesi and Srinivasan [32]
	large	randomized	$O(\log \Delta) + 2^{O(\sqrt{\log \log n})}$	Ghaffari [17]

However, could we get the best of the 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 for either of these problems that runs in time $o\big(\Delta + \frac{\log \log n}{\log \log \log n}\big)$, and hence certainly 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 [13], either.

With this result we resolve Open Problem 11.6 in Barenboim and Elkin [5], and present a proof for the conjecture of Göös et al. [18].

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 [5]. 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 [7]. However, in 2015 the paths diverged: Barenboim [4] and Fraigniaud et al. [14] 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.

However, there is a variant of $(\Delta + 1)$-vertex coloring that is closely related to MIS: *greedy coloring* [15]. 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.

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 [21].

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 [18].

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 [18, 21], the techniques of the present work are entirely unrelated to those.

Speedup simulation technique. The technique that we use in this work is based on *speedup simulation*. 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 [26, 27] and Naor [30] 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 [10, 12], as well as to prove lower bounds for *weak coloring* [3, 9].

In principle, the approach can be used with *any* locally checkable graph problem, in a mechanical manner [9]. 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
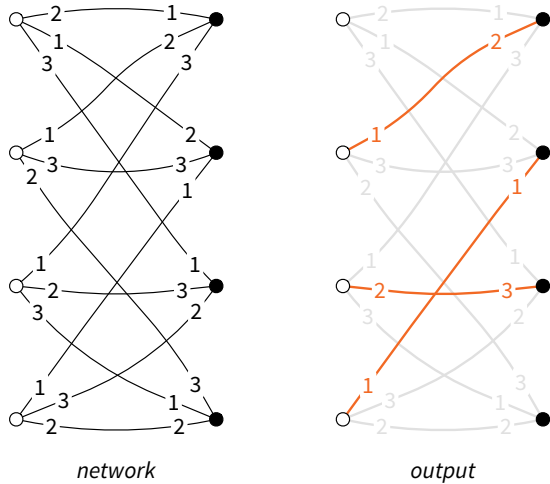
Figure 2. A 3-regular bipartite port-numbered network and a maximal matching.

and we can nevertheless apply speedup simulation for any pair of them.

The formalism that we use is closely related to [9]—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.

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 [19]: 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 $\Omega(\log \Delta / \log \log \Delta)$ [23–25], 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.

3.1. Lower bound idea

Our plan is to prove a lower bound using a *speedup simulation argument* [3, 9, 10, 12, 26, 27, 30]. 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. 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 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 this way find a k-matching in $O(\Delta/k)$ rounds. Could we define that Π_i is the problem of finding an i-matching, and 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?

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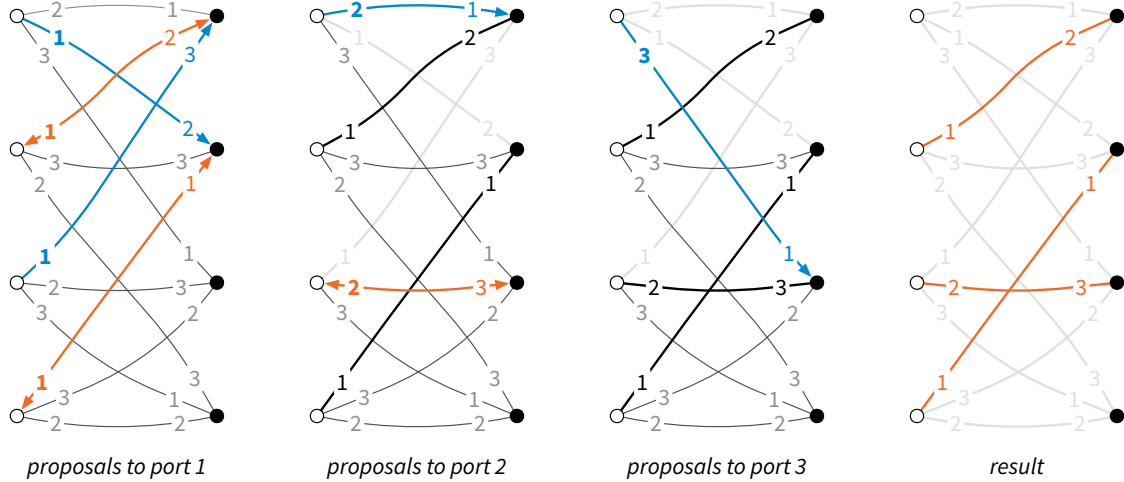
Figure 3. The proposal algorithm finds a maximal matching in $O(\Delta)$ rounds—orange arrows are accepted proposals and blue arrows are rejected proposals.

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 labellings of the edges incident to a white node, and B is the set of feasible labellings for the edges incident to a black node.

Here we will assume that 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 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. $W = \{000, 001\}$. We emphasize that the order of the elements does not matter here, and we could equally well write e.g. $W = \{000, 010\}$. 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 $W = 0^2[01]$.

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, $\Sigma = \{M, P, O\}$, with the following rules:

$$W = MO^{\Delta-1} \mid P^\Delta,$$
$$B = M[PO]^{\Delta-1} \mid O^\Delta. \tag{1}$$

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.

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.

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). Then it is also easy to construct a finite Δ-regular graph G' such that the radius-T neighborhood of u in G (including the port numbering) is isomorphic to the radius-T neighborhood of some node u' in G', and therefore A will also fail around u' in G'.

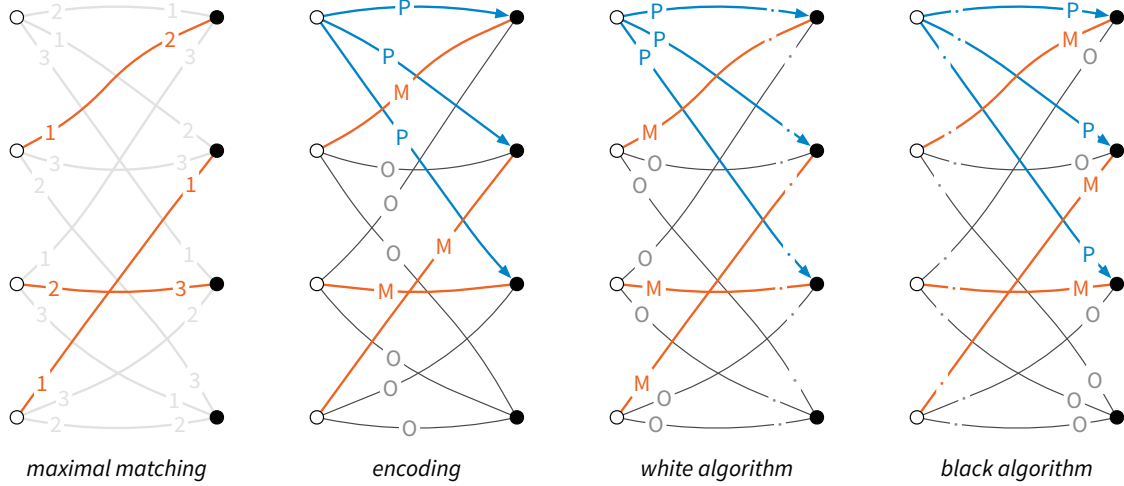| maximal matching | encoding | white algorithm | black algorithm |

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

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 $\Sigma = \{M, P, O, X\}$, 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

$$W_\Delta(x, y) = \Big(\mathsf{MO}^{d-1} \,\Big|\, \mathsf{P}^d\Big)\mathsf{O}^y\mathsf{X}^x,$$

$$B_\Delta(x, y) = \Big([\mathsf{MX}][\mathsf{POX}]^{d-1} \,\Big|\, [\mathsf{OX}]^d\Big)[\mathsf{POX}]^y[\mathsf{MPOX}]^x, \tag{2}$$

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

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

$$\begin{array}{c} \mathsf{M} \\ \searrow \\ \qquad\qquad \mathsf{X} \qquad\qquad (3) \\ \nearrow \\ \mathsf{P} \longrightarrow \mathsf{O} \end{array}$$

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. 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 they do 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
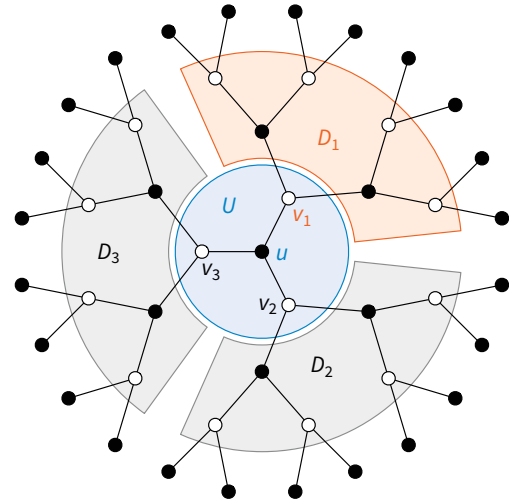

Figure 5. Speedup simulation, for $T = 2$ and $\Delta = 3$. The radius-$(T-1)$ neighborhood of u is U, and the radius-T neighborhood of v_i is $V_i = U \cup D_i$. The key observation is that D_i and D_j are disjoint for $i \ne j$.

and sometimes helpful. Unmatched black nodes can accept P instead of O once.
- $\Pi_\Delta(1, 0)$: All white nodes will use X instead of P or O once—again, this is always feasible and sometimes helpful. 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 $T \ge 1$ rounds in trees, for a sufficiently large Δ. Throughout this section, let $d = \Delta - x - y$.

486

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.

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

$$\{X\} \mapsto \boxed{X},$$
$$\{M\}, \{M, X\} \mapsto \boxed{MX},$$
$$\{O\}, \{O, X\} \mapsto \boxed{OX},$$
$$\{M, O\}, \{M, O, X\} \mapsto \boxed{MOX},$$
$$\{P\}, \{P, O\}, \{P, X\}, \{P, O, X\} \mapsto \boxed{POX},$$
$$\{M, P\}, \{M, P, O\}, \{M, P, X\}, \{M, P, O, X\} \mapsto \boxed{MPOX}.$$

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 $\{M, O, X\}$ with the label $\boxed{MOX}$.

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 6 do not overlap—hence if there is some input in which D_1 is "bad", and another input in which D_2 is "bad", we can also construct an input in which both D_1 and D_2 are simultaneously "bad". We make the following observations:

1) There can be at most $x + 1$ edges incident to u with a label in $\{\boxed{MX}, \boxed{MOX}, \boxed{MPOX}\}$. 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 $\{\boxed{POX}, \boxed{MPOX}\}$, then there has to be at least one edge with a label in $\{\boxed{X}, \boxed{MX}\}$. 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.

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:

$$\begin{array}{ccccc} \boxed{X} & \longrightarrow & \boxed{OX} & \longrightarrow & \boxed{POX} \\ \downarrow & & \downarrow & & \downarrow \\ \boxed{MX} & \longrightarrow & \boxed{MOX} & \longrightarrow & \boxed{MPOX} \end{array} \qquad (4)$$

There are two cases:

1) There are at most $x + y$ copies of $\boxed{POX}$ on edges incident to u. We also know that there are at most $x + 1$ copies of labels $\{\boxed{MX}, \boxed{MOX}, \boxed{MPOX}\}$. Hence there have to be at least $\Delta - (x + y) - (x + 1) = d - x - 1$ copies of labels $\{\boxed{X}, \boxed{OX}\}$. We proceed as follows:

   - Replace all of $\{\boxed{MX}, \boxed{MOX}\}$ with $\boxed{MPOX}$.
   - Replace some of $\{\boxed{X}, \boxed{OX}, \boxed{POX}\}$ with $\boxed{MPOX}$ so that the total number of $\boxed{MPOX}$ is exactly $x + 1$.
   - Replace some of the remaining $\{\boxed{X}, \boxed{OX}\}$ with $\boxed{POX}$ so that the total number of $\boxed{POX}$ is exactly $x + y$.
   - Replace all of the remaining $\boxed{X}$ with $\boxed{OX}$.

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

2) There are more than $x + y$ copies of $\boxed{POX}$. Then we know that there is at least one copy of $\{\boxed{X}, \boxed{MX}\}$. We first proceed as follows:

   - Replace all $\boxed{OX}$ with $\boxed{POX}$ and all $\boxed{MOX}$ with $\boxed{MPOX}$.
   - If needed, replace one $\boxed{X}$ with $\boxed{MX}$ so that there is at least one copy of $\boxed{MX}$.
   - Replace all remaining copies of $\boxed{X}$ with $\boxed{POX}$.

   At this point we have at least one $\boxed{MX}$ and all other labels are in $\{\boxed{MPOX}, \boxed{POX}\}$. Furthermore, as we originally had at most $x + 1$ copies of $\{\boxed{MX}, \boxed{MOX}, \boxed{MPOX}\}$, and we created at most one new label in this set, we have now got at most $x + 2$ copies of $\{\boxed{MX}, \boxed{MPOX}\}$. Hence we have got at least $d + y - 2$ copies of $\boxed{POX}$. We continue:

   - Replace some of $\boxed{MX}$ with $\boxed{MPOX}$ so that the total number of $\boxed{MX}$ is exactly 1.
   - Replace some of $\boxed{POX}$ with $\boxed{MPOX}$ so that the total number of $\boxed{POX}$ is exactly $d + y - 2$ and hence the total number of $\boxed{MPOX}$ is $x + 1$.

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

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

Note that we have completely eliminated labels $\boxed{\text{X}}$ and $\boxed{\text{MOX}}$ and we are left with the following four labels:

$$
\begin{array}{ccc}
\cdot & \longrightarrow \boxed{\text{OX}} & \longrightarrow \boxed{\text{POX}} \\
\downarrow & \downarrow & \downarrow \\
\boxed{\text{MX}} & \longrightarrow \cdot & \longrightarrow \boxed{\text{MPOX}}
\end{array}
\tag{5}
$$

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

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

Note that after this mapping, poset (4) restricted to $\{\text{M}, \text{P}, \text{O}, \text{X}\}$ is the same as poset (3).

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. There are two cases:

1) We have a neighborhood in which the output of A at v matches $\text{MO}^{d-1+y}\text{X}^x$. 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 is in $\{\boxed{\text{MX}}, \boxed{\text{MOX}}, \boxed{\text{MPOX}}\}$. As we followed (4), the output of A_3 for this edge is also in $\{\boxed{\text{MX}}, \boxed{\text{MOX}}, \boxed{\text{MPOX}}\}$. After remapping, the output of A_4 for this edge is in $\{\text{M}, \text{X}\}$.
   - 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 $\{\boxed{\text{OX}}, \boxed{\text{MOX}}, \boxed{\text{POX}}, \boxed{\text{MPOX}}\}$, and the output of A_4 is hence in $\{\text{P}, \text{O}, \text{X}\}$.

2) We have a neighborhood in which the output of A at v matches $\text{P}^d\text{O}^y\text{X}^x$. Then by a similar reasoning:

   - We have d edges incident to v such that the output of A_3 is in $\{\boxed{\text{POX}}, \boxed{\text{MPOX}}\}$ and hence the output of A_4 is in $\{\text{O}, \text{X}\}$.
   - We have y edges incident to v such that the output of A_3 is in $\{\boxed{\text{OX}}, \boxed{\text{MOX}}, \boxed{\text{POX}}, \boxed{\text{MPOX}}\}$ and hence the output of A_4 is in $\{\text{P}, \text{O}, \text{X}\}$.

Hence the neighborhood of a white node in A_4 satisfies

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

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

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

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

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

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.

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 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 1, we obtain the following corollary:

Corollary 1. 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 1, 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 2. There is no white algorithm that solves $\Pi_\Delta(x, y)$ in 0 rounds in trees when $\Delta = x + y + 2$.

Proof. 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) $w = \mathsf{MO}^{y+1}\mathsf{X}^x$: 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. $\square$

3.6. Amplifying through k-matchings

A straightforward application of Corollary 1 and Lemma 2 already gives a new lower bound—it just is not tight yet:

Theorem 1. There is no deterministic algorithm that finds a maximal matching in $o(\sqrt{\Delta})$ rounds in the port-numbering model in 2-colored graphs.

Proof. Any algorithm that runs in $o(\sqrt{\Delta})$ rounds implies a white algorithm that runs in $o(\sqrt{\Delta})$ rounds, and naturally a general algorithm also has to solve the problem correctly in trees. Recall that $\Pi_\Delta(0, 0)$ is equal to the maximal matching problem. If we could solve it with a white algorithm in $o(\sqrt{\Delta})$ rounds, then by Corollary 1 we could also solve $\Pi_\Delta(x', y')$ for $\Delta \gg x' + y' + 2$ in 0 rounds, which contradicts Lemma 2. $\square$

However, we can easily amplify this and get a linear-in-Δ lower bound. Recall k-matchings from Definition 1. We make the following observation:

Lemma 3. 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:

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

$$[\mathsf{OX}]^{\Delta-k+1}[\mathsf{MPOX}]^{k-1}. \qquad \square$$

Now the same reasoning as Theorem 1 gives the following result:

Theorem 2. There is no deterministic algorithm that finds an $O(\sqrt{\Delta})$-matching in $o(\sqrt{\Delta})$ rounds in the port-numbering model in 2-colored graphs.

Proof. Plug in $x = O(\sqrt{\Delta})$, $y = 0$, $d = \Delta - x$, and $T = o(\sqrt{\Delta})$ in Corollary 1, and we obtain $\Delta - x' - y' = \Delta - o(\Delta) \gg 2$; then apply Lemma 2 and Lemma 3. $\square$

Now it turns out that Theorem 2 is tight, and we can use it to prove a tight lower bound for maximal matchings:

Theorem 3. There is no deterministic algorithm that finds a maximal matching in $o(\Delta)$ rounds in the port-numbering model in 2-colored graphs.

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 2 to see that A cannot exist.
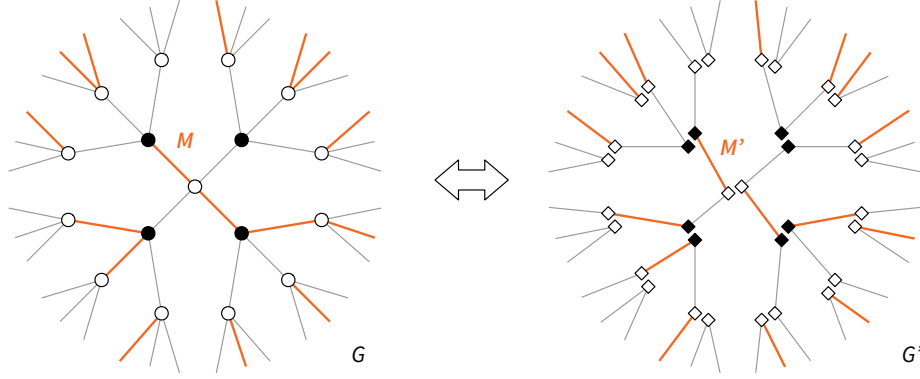
Figure 7. Splitting nodes of degree $\Delta = 4$ in $\sqrt{\Delta} = 2$ mininodes, each of degree $\sqrt{\Delta} = 2$. A maximal matching M' of G' defines a 2-matching M in G.

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. □

Corollary 2. There is no deterministic algorithm that finds a maximal matching in $o(\Delta + \log n / \log \log n)$ rounds in the port-numbering model in 2-colored graphs.

Proof. Consider the case $\Delta \approx \log n / \log \log n$, and follow the same idea as in the proof of Theorem 3. Any algorithm A with a running time $T = o(\Delta + \log n / \log \log n) = o(\Delta)$ has to fail in some radius-T neighborhood, and for large enough Δ, there are fewer than $\Delta^T < \Delta^\Delta$ nodes in the neighborhood. Hence we can construct a Δ-regular graph G with $n \approx \Delta^\Delta$ nodes in which A has to fail. □

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.

3.7. 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 [26, 27]. Algorithm A_2 is related to the idea of *simplification by maximality* in [9]—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:

1) We start with the formulation of (1); let us call this problem Π_0. This is an edge labeling problem with 3 labels.
2) We apply the automatic speedup simulation framework of [9] 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.
3) Repeat the same process to obtain problem Π_2, which is an edge labeling problem with 6 labels.
4) 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.

5) 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.

6) 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.

7) 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 [9] 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: $\mathsf{M}, \mathsf{P}, \mathsf{O}$).

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.

LOCAL model. In the area of distributed graph algorithms, there are two widely used models of computing: *LOCAL* and *CONGEST* [33]. 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 global success probability is at least $1 - 1/n$.

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 [10]: 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.

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.

4.1. Speedup simulation

We will first prove a probabilistic version of Lemma 1; the proof follows the same strategy as [10]:

Lemma 4. 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 α be a parameter that we will fix later.

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

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

We will need to keep track of two distinct ingredients: port numbering and random bits. Given a region $X(\cdot)$, we write $X_p(\cdot)$ for the port numbers assigned to the edges incident to the nodes of $X(\cdot)$, and $X_R(\cdot)$ for the random bits assigned to the nodes of $X(\cdot)$. For example, $U_p(u)$ refers to the port numbering within distance $T - 1$ from a black node u, while $U(u)$ refers to all information (both random bits and the port numbers).

Definition 2 (typical labels)**.** We say that x is a *typical* label for edge $\{u, v\}$ w.r.t. $V_p(v)$ and $U_R(u)$ if

$$\Pr_{D_R(u,v)}\left[A \text{ labels } e_i \text{ with } x \mid V_p(v), U_R(u)\right] \geq \alpha.$$

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 and for each port numbering (recall that the alphabet size is 4).

Definition 3 (potential labels)**.** We say that x is a *potential* label for edge $\{u, v\}$ w.r.t. $U(u)$ if there exists some port numbering $D_p(u, v)$ such that x is a typical label for $\{u, v\}$ w.r.t. $V_p(v)$ and $U_R(u)$.

Again, as long as $\alpha \leq 1/4$, there is always at least one potential label.

Definition 4 (lucky bits)**.** We say that random bits $U_R(u)$ are *lucky* given $U_p(u)$ if the following holds: for each edge $\{u, v\}$ we can pick any potential label and this makes node u happy, i.e., any combination of potential labels forms a word in $B_\Delta(x, y)$. Otherwise, we call $U_R(u)$ *unlucky*.

Lemma 5. Given any port numbering $U_p(u)$, the probability that $U_R(u)$ is unlucky is at most p/α^Δ.

Proof. If $U_R(u)$ is unlucky, then there is some way to choose port numberings $D_p(u, v_i)$ and edge labels x_i for the edges $e_i = \{u, v_i\}$ such that x_i is typical w.r.t. $V_p(v_i), U_R(u)$ and u is unhappy. By definition, given $V_p(v_i), U_R(u)$, algorithm A will output x_i on edge e_i with probability at least α, and this only depends on $D_R(u, v_i)$. As the regions $D_R(u, v_i)$ are independent, we see that u is unhappy with probability at least α^Δ for this specific choice of port numbering.

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

Definition 5 (nice bits)**.** We say that random bits in $V_R(v)$ are *nice* w.r.t. $V_p(v)$ if the output of A for v produces a typical label for every edge incident to v. Otherwise the bits are *bad*.

Lemma 6. Given any port numbering $V_p(v)$, the probability that $V_R(v)$ is bad is at most $4\Delta\alpha$.

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

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

The claim follows by union bound. $\square$

Definition 6 (friendly bits)**.** We say that random bits in $V_R(v)$ are *friendly* w.r.t. $V_p(v)$ if:

1) The random bits in $V_R(v)$ are nice w.r.t. $V_p(v)$.

2) For each black neighbor u of v, the bits in $U_R(u)$ are lucky w.r.t. $U_p(u)$.

Otherwise the bits are *unfriendly*.

Lemma 7. Given any port numbering $V_p(v)$, the probability that $V_R(v)$ 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 $\Delta(4\alpha + p/\alpha^\Delta)$. $\square$

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

Corollary 3. Given any port numbering $V_p(v)$, the probability that $V_R(v)$ is unfriendly is at most $q = 5\Delta p^{\frac{1}{\Delta+1}}$.

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 potential 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 potential 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 potential labels that A_1 outputs. The rest of the analysis holds verbatim.

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 $V_R(v)$ 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.

4.2. Multiple speedup steps

By a repeated application of Lemma 4, we obtain the following corollary that is analogous to Corollary 1:

Corollary 4. 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

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

Proof. Let p_i be the local error probability after applying Lemma 4 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

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

4.3. Base case

Lemma 8. 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 $\mathsf{M}\mathsf{O}^{\Delta-x-1}\mathsf{X}^x$, and probability $1 - q$ to output $\mathsf{P}^{\Delta-x-y}\mathsf{O}^y\mathsf{X}^x$. There are two cases:

- $q \geq \frac{1}{2}$: There is some port i such that the probability that A labels i with M is at least $q/\Delta \geq 1/(2\Delta)$. Let us consider a black node u such that all white neighbors v of u have their port i connected to u. Now if the first $x + 2$ 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 there is some port i such that the probability that A labels i with P is at least $1/(2\Delta)$.

A black node u is unhappy if all 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

$$\frac{1}{2^\Delta} \cdot \frac{1}{\Delta^{x+y+1}} \geq \frac{1}{2^\Delta} \cdot \frac{1}{\Delta^{\Delta/4+1}} \geq \frac{1}{\Delta^\Delta}. \quad \square$$

4.4. Putting things together

Lemma 9. 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, we obtain a white randomized algorithm A' that solves $\Pi_\Delta(x', y')$ in 0 rounds with local error probability p', where

$$
\begin{aligned}
x' &= x + T \leq \frac{17}{16}\sqrt{\Delta} \leq \frac{1}{8}\Delta, \\
y' &= y + T(x + (T-1)/2) \leq \frac{33}{512}\Delta \leq \frac{1}{8}\Delta, \\
p' &\leq (5\Delta)^2 p^{1/(\Delta+1)^T}.
\end{aligned}
$$

By Lemma 8 we have $p' > 1/\Delta^\Delta$ and therefore

$$p > \frac{1}{\left((5\Delta)^2\Delta^\Delta\right)^{(\Delta+1)^T}} > \frac{1}{2^{\Delta^{2T+2}}}. \quad \square$$

Theorem 4. There is no randomized algorithm that finds a $\sqrt{\Delta}$-matching in

$$T = o\left(\sqrt{\Delta} + \frac{\log\log n}{\log\log\log n}\right)$$

rounds with probability at least $1 - 1/n$.

Proof. Assume that such an algorithm exists. By Lemma 3 there is then a white randomized algorithm A that solves $\Pi_\Delta(\sqrt{\Delta}, 0)$. Consider the family of Δ-regular graphs with

$$\sqrt{\Delta} \approx \frac{\log\log n}{\log\log\log n}.$$

For a large enough Δ, we would have $T < \sqrt{\Delta}/16$. We can construct a graph that contains a local neighborhood that is a Δ-regular tree of depth T, and by Lemma 9, algorithm A fails in such a neighborhood with probability at least

$$\frac{1}{2^{\Delta^{2T+2}}} > \frac{1}{2^{2^{\sqrt{\Delta}\log(\sqrt{\Delta})/3}}} > \frac{1}{2^{2^{\log\log n/3}}} > \frac{1}{n}.$$

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

Now the same idea as in Theorem 3 gives our main result:

Theorem 5. There is no randomized distributed algorithm that finds a maximal matching in $o\left(\Delta + \frac{\log\log n}{\log\log\log n}\right)$ rounds with probability at least $1 - 1/n$.

4.5. 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: a fast deterministic algorithm implies a faster randomized algorithm, which is in contradiction with Theorem 5. Our proof is similar to Chang et al. [11, Theorem 6], and the coloring algorithm follows the idea of Barenboim et al. [8, Remark 3.6].

Theorem 6. There is no deterministic distributed algorithm that finds a maximal matching in $o\big(\Delta + \frac{\log n}{\log \log n}\big)$ rounds.

Proof. Assume for a contradiction that such a deterministic algorithm A exists. Specifically assume that for each N there exists A_N which computes a maximal matching on graphs of size N when the unique identifiers come from the set $\{1, 2, \ldots, N\}$. We now construct a randomized algorithm violating the lower bound given in Theorem 5 as follows.

Consider any graph G on n nodes with maximum degree $\Delta = \Theta(\log \log n / \log \log \log n)$. We will choose $N = \log n$ and simulate A_N on each local neighborhood of G. Note that with this choice, G has maximum degree $\Delta = \Theta(\log N / \log \log N)$.

In order to simulate A_N we need to compute a labeling that *locally* looks like an assignment of unique identifiers of size N. First, each node picks a random value from $\{1, 2, \ldots, n^3\}$. These values are globally unique with probability at least $1 - 1/n$. The rest of the simulation is deterministic.

Let $T = T(\Delta, N)$ denote the running time of A_N. We compute an N-coloring of G^{2T+2}, the $(2T + 2)$th power of G, using three iterations of Linial's coloring algorithm [27, 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. By our assumptions on Δ and N, the number of colors is $o(N)$. The coloring can be computed in time $O(T)$ in the original graph G.

Finally we simulate A_N on the computed coloring. If we look at the radius-$(T+1)$ neighborhood of any node in G, the coloring looks like an assignment of unique identifiers from $\{1, 2, \ldots, N\}$, and thus the output of A_N is well defined and correct by a standard argument—see e.g. [11, Theorem 6]. Furthermore, the simulation of A_N on G can be done in T rounds.

The simulation works with probability at least $1 - 1/n$ and runs in

$$O(T) = o(\Delta + \log \log n / \log \log \log n)$$

rounds in total in graphs of maximum degree $\Delta = \Theta(\log \log n / \log \log \log n)$. By observing that the lower bound construction used in the proof of Theorem 5 satisfies $\Delta = \Theta(\log \log n / \log \log \log n)$, we obtain a contradiction. $\qquad \square$

4.6. 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. We obtain:

Corollary 5. There is no randomized distributed algorithm that finds a maximal independent set in $o\big(\Delta + \frac{\log \log n}{\log \log \log n}\big)$ rounds with probability at least $1 - 1/n$.

Corollary 6. There is no deterministic distributed algorithm that finds a maximal independent set in $o\big(\Delta + \frac{\log n}{\log \log n}\big)$ rounds.

4.7. Matching upper bounds

Recall the bound from Theorem 5: there is no randomized distributed algorithm that finds a maximal matching in $o(\Delta + \log \log n / \log \log \log n)$ rounds.

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 [31].

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 [8, Theorem 7.3]. Note that the construction of our lower bounds ensures that they already hold on trees.

4.8. Discussion

We have learned that the prior algorithms for MM and MIS with a running time of $O(\Delta + \log^* n)$ rounds [7, 31] 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 $O(\log \Delta + \log^c \log n)$, possibly achieving intermediate trade-offs such as $O(\sqrt{\Delta} + \log \log \log n)$, but 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 [11].

The following problems are now some examples of the main open questions that still remain wide open:

- Small Δ: What is the complexity of $(\Delta + 1)$-vertex coloring and $(2\Delta - 1)$-edge coloring in the region $\Delta \ll \log \log n$? Can we solve these, e.g., in time $O(\log \Delta + \log^* n)$?

- Large Δ: What is the (deterministic or randomized) complexity of MIS in the region $\Delta \gg \log \log n$? Is MIS much harder than MM in this region?

Acknowledgments

References

[1] N. Alon, L. Babai, and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7, no. 4, pp. 567–583, 1986.

[2] M. Åstrand and J. Suomela, "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, 2010, pp. 294–302.

[3] A. Balliu, J. Hirvonen, D. Olivetti, and J. Suomela, "Hardness of minimal symmetry breaking in distributed computing," 2018. [Online]. Available: http://arxiv.org/abs/1811.01643

[4] L. Barenboim, "Deterministic $(\Delta+1)$-Coloring in Sublinear (in Δ) Time in Static, Dynamic, and Faulty Networks," *Journal of the ACM*, vol. 63, no. 5, pp. 1–22, 2016.

[5] L. Barenboim and M. Elkin, *Distributed Graph Coloring: Fundamentals and Recent Developments*, 2013, vol. 4, no. 1.

[6] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider, "The Locality of Distributed Symmetry Breaking," in *Proc. 53rd Annual Symposium on Foundations of Computer Science (FOCS 2012)*. IEEE, 2012, pp. 321–330.

[7] L. Barenboim, M. Elkin, and F. Kuhn, "Distributed $(\Delta+1)$-Coloring in Linear (in Δ) Time," *SIAM Journal on Computing*, vol. 43, no. 1, pp. 72–95, 2014.

[8] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider, "The Locality of Distributed Symmetry Breaking," *Journal of the ACM*, vol. 63, no. 3, pp. 1–45, 2016.

[9] S. Brandt, "An Automatic Speedup Theorem for Distributed Problems," 2019. [Online]. Available: http://arxiv.org/abs/1902.09958

[10] S. Brandt, O. Fischer, J. Hirvonen, B. Keller, T. Lempiäinen, J. Rybicki, J. Suomela, and J. Uitto, "A lower bound for the distributed Lovász local lemma," in *Proc. 48th ACM Symposium on Theory of Computing (STOC 2016)*. ACM Press, 2016, pp. 479–488.

[11] Y.-J. Chang, T. Kopelowitz, and S. Pettie, "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, 2016, pp. 615–624.

[12] Y.-J. Chang, Q. He, W. Li, S. Pettie, and J. Uitto, "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, 2018, pp. 2633–2652.

[13] M. Fischer, "Improved Deterministic Distributed Matching via Rounding," in *Proc. 31st International Symposium on Distributed Computing (DISC 2017)*, 2017, pp. 17:1–17:15.

[14] P. Fraigniaud, M. Heinrich, and A. Kosowski, "Local Conflict Coloring," in *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2016, pp. 625–634.

[15] C. Gavoille, R. Klasing, A. Kosowski, Ł. Kuszner, and A. Navarra, "On the complexity of distributed graph coloring with local minimality constraints," *Networks*, vol. 54, no. 1, pp. 12–19, 2009.

[16] M. Ghaffari, "LOCAL Algorithms: The Chasm Between Deterministic and Randomized," in *6th Workshop on Advances in Distributed Graph Algorithms (ADGA 2017)*, 2017. [Online]. Available: http://adga.hiit.fi/2017/ghaffari.pdf

[17] ——, "An Improved Distributed Algorithm for Maximal Independent Set," in *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2016, pp. 270–277.

[18] M. Göös, J. Hirvonen, and J. Suomela, "Linear-in-Δ lower bounds in the LOCAL model," *Distributed Computing*, vol. 30, no. 5, pp. 325–338, 2017.

[19] M. Hanckowiak, M. Karonski, and A. Panconesi, "On the Distributed Complexity of Computing Maximal Matchings," in *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*. ACM/SIAM, 1998, pp. 219–225.

[20] ——, "On the Distributed Complexity of Computing Maximal Matchings," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 41–57, 2001.

[21] J. Hirvonen and J. Suomela, "Distributed maximal matching: greedy is optimal," in *Proc. 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2012)*. ACM Press, 2012, pp. 165–174.

[22] A. Israeli and A. Itai, "A fast and simple randomized parallel algorithm for maximal matching," *Information Processing Letters*, vol. 22, no. 2, pp. 77–80, 1986.

[23] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "What cannot be computed locally!" in *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2004)*. New York, New York, USA: ACM Press, 2004, pp. 300–309.

[24] ——, "The price of being near-sighted," in *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*. New York, New York, USA: ACM

Press, 2006, pp. 980–989.

[25] ——, "Local Computation: Lower and Upper Bounds," *Journal of the ACM*, vol. 63, no. 2, pp. 1–44, 2016.

[26] N. Linial, "Distributive graph algorithms Global solutions from local data," in *Proc. 28th Annual Symposium on Foundations of Computer Science (FOCS 1987)*. IEEE, 1987, pp. 331–335.

[27] ——, "Locality in Distributed Graph Algorithms," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193–201, 1992.

[28] M. Luby, "A simple parallel algorithm for the maximal independent set problem," in *Proc. 17th Annual ACM Symposium on Theory of Computing (STOC 1985)*. New York, New York, USA: ACM Press, 1985, pp. 1–10.

[29] ——, "A Simple Parallel Algorithm for the Maximal Independent Set Problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986.

[30] M. Naor, "A lower bound on probabilistic algorithms for distributive ring coloring," *SIAM Journal on Discrete Mathematics*, vol. 4, no. 3, pp. 409–412, 1991.

[31] A. Panconesi and R. Rizzi, "Some simple distributed algorithms for sparse networks," *Distributed Computing*, vol. 14, no. 2, pp. 97–100, 2001.

[32] A. Panconesi and A. Srinivasan, "On the Complexity of Distributed Network Decomposition," *Journal of Algorithms*, vol. 20, no. 2, pp. 356–374, 1996.

[33] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[34] J. Suomela, "Lower Bounds for Local Algorithms," in *3rd Workshop on Advances in Distributed Graph Algorithms (ADGA 2014)*, 2014. [Online]. Available: http://adga2014.hiit.fi/jukka.pdf