Linear-in-Δ Lower Bounds in the LOCAL Model*

Mika Göös
mika.goos@mail.utoronto.ca
Department of Computer Science, University of Toronto, Canada

Juho Hirvonen
juho.hirvonen@aalto.fi
Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland

Jukka Suomela
jukka.suomela@aalto.fi
Helsinki Institute for Information Technology HIIT,
Department of Information and Computer Science, Aalto University, Finland

Abstract. By prior work, there is a distributed graph algorithm that finds a maximal fractional matching (maximal edge packing) in $O(\Delta)$ rounds, independently of n; here Δ is the maximum degree of the graph and n is the number of nodes in the graph. We show that this is optimal: there is no distributed algorithm that finds a maximal fractional matching in $o(\Delta)$ rounds, independently of n. Our work gives the first linear-in-Δ lower bound for a natural graph problem in the standard LOCAL model of distributed computing—prior lower bounds for a wide range of graph problems have been at best logarithmic in Δ.

* This work is an extended and revised version of a preliminary conference report [10].

1 Introduction

This work settles the distributed time complexity of the *maximal fractional matching* problem (see Section 1.2 for definitions) as a function of Δ, the maximum degree of the input graph.

By prior work [4], there is a distributed algorithm that finds a maximal fractional matching (also known as a maximal edge packing) in $O(\Delta)$ communication rounds, independently of the number of nodes. In this work, we show that this is optimal: there is no distributed algorithm that finds a maximal fractional matching in $o(\Delta)$ rounds.

This is the first linear-in-Δ lower bound for a natural graph problem in the standard LOCAL model of distributed computing. It is also a step towards understanding the complexity of the non-fractional analogue, the maximal matching problem, which is a basic symmetry breaking primitive in the field of distributed graph algorithms. For many related primitives, the prior lower bounds in the LOCAL model have been at best logarithmic in Δ.

1.1 Matchings: state-of-the-art

Simple randomised distributed algorithms that find a maximal matching in time $O(\log n)$ have been known since the 1980s [1, 16, 23]. Currently, the fastest algorithms that compute a maximal matching stand as follows:

- **Dense graphs.** There is a recent $O(\log \Delta + \log^4 \log n)$-time randomised algorithm due to Barenboim et al. [6]. The fastest known deterministic algorithm runs in time $O(\log^4 n)$ and is due to Hańćkowiak et al. [13].
- **Sparse graphs.** There is a $O(\Delta + \log^* n)$-time deterministic algorithm due to Panconesi and Rizzi [27]. Here $\log^* n$ is the iterated logarithm of n, a very slowly growing function.

Our focus is on the sparse case. It is a long-standing open problem to either improve on the algorithm of Panconesi and Rizzi, or prove it optimal. As we are dealing with two independent parameters n and Δ, we must be careful what we mean by "proving it optimal". The meaning is: (1) there is no algorithm with run-time $O(\Delta) + o(\log^* n)$; (2) nor an algorithm with run-time $o(\Delta) + O(\log^* n)$.

The first type of lower bound follows from Linial's [22] seminal work. Linial shows that 3-colouring a cycle is not possible in time $o(\log^* n)$, so by a simple reduction:

(1) Linial's result: *Maximal matchings cannot be computed in time $f(\Delta) + o(\log^* n)$ for any function f.*

Hence we have an arbitrarily large lower bound in terms of Δ, since $\Delta = 2$ on cycles. However, viewing Linial's result from the perspective of Δ is not very meaningful: the source of hardness exhibited in Linial's proof is *not* the degree of the graph but its growing size.

The second type of lower bound has remained elusive (see Barenboim and Elkin [5, Open Problem 10.6]):

(2) Open problem: *Can maximal matchings be computed in time $o(\Delta) + O(\log^* n)$?*

We conjecture that there are no such algorithms. Our linear-in-Δ lower bound for the *fractional* version of this problem builds towards proving such conjectures: the source of hardness for maximal fractional matchings is *not* the size of the graph but the growing degree. The graphs in our lower bound construction end up satisfying $\Delta = \Theta(\log^* n)$; if this could be improved to a family where $\Delta = \omega(\log^* n)$, we would obtain a negative answer to (2).

1.2 Fractional matchings

While a matching associates a weight 0 or 1 with each edge of a graph, with 1 indicating that the edge is in a matching, a fractional matching (FM) associates a weight between 0 and 1 with each edge. In both cases, the total weight of the edges incident to any given node has to be at most 1.

Formally, let $G = (V, E)$ be a simple undirected graph and let $y \colon E \to [0, 1]$ associate weights to the edges of G. Define, for each $v \in V$,

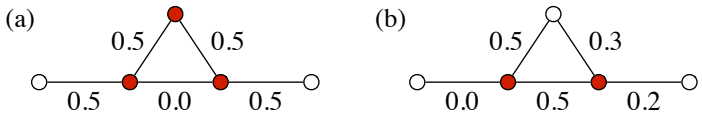
$$y[v] := \sum_{e \in E : v \in e} y(e).$$

The function y is called a *fractional matching*, or an FM for short, if $y[v] \leq 1$ for each node v. A node v is *saturated* if $y[v] = 1$.

There are two interesting varieties of fractional matchings.

- **Maximum weight.** An FM y is of *maximum weight*, if its total weight $\sum_{e \in E} y(e)$ is the maximum over all fractional matchings on G.
- **Maximality.** An FM y is *maximal*, if each edge e has at least one saturated endpoint $v \in e$.

See below for examples of (a) a maximum-weight FM, and (b) a maximal FM; the saturated nodes are highlighted.



Distributed complexity. The distributed complexity of computing maximum-weight FMs is completely understood. It is easy to see that computing an exact solution requires time $\Omega(n)$ already on odd-length path graphs (a node needs to learn the parity of its distance from an endpoint). If one settles for an approximate solution, then FMs whose total weight is at least a $(1-\epsilon)$-fraction of the maximum can be computed in time $O(\epsilon^{-1} \log \Delta)$ by the well-known results of Kuhn et al. [17–19]. This is optimal: Kuhn et al. also show that any constant-factor approximation of maximum-weight FMs requires time $\Omega(\log \Delta)$.

By contrast, the complexity of computing maximal FMs has not been understood. A maximal FM is a 1/2-approximation of a maximum-weight FM, so the results of Kuhn et al. imply that finding a maximal FM requires time $\Omega(\log \Delta)$, but this lower bound is exponentially small in comparison to the $O(\Delta)$ upper bound [4].

1.3 Contributions

We prove that the $O(\Delta)$-time algorithm [4] for maximal fractional matchings is optimal:

Theorem 1. *There is no (randomised)* LOCAL *algorithm that finds a maximal fractional matching in $o(\Delta)$ rounds, independently of n.*

To our knowledge, this is the first linear-in-Δ lower bound in the LOCAL model for a classical graph problem. Indeed, prior lower bounds have typically fallen in one of the following categories:

- they are logarithmic in Δ [17–19],
- they analyse the complexity as a function of n for a fixed Δ [7–9, 11, 21, 22, 25],
- they only hold in a model that is strictly weaker than LOCAL [15, 20].

1.4 The LOCAL model

Our result holds in the standard LOCAL model of distributed computing [22, 28]. For now, we only recall the basic setting; see Section 3 for precise definitions.

In the LOCAL model an input graph $G = (V, E)$ defines both the problem instance and the structure of the communication network. Each node $v \in V$ is a computer and each edge $\{u, v\} \in E$ is a communication link through which nodes u and v can exchange messages. Initially, each node is equipped with a *unique identifier* and, if we study randomised algorithms, a source of randomness. In each *communication round*, each node in parallel (1) sends a message to each neighbour, (2) receives a message from each neighbour, and (3) updates its local state. Eventually, all nodes have to stop and announce their local outputs—in our case the local output of a node $v \in V$ is an encoding of the weight $y(e)$ for each edge e incident to v. The *running time* t of the algorithm is the number of communication rounds until all nodes have stopped. We call an algorithm *strictly local*, or simply *local*, if $t = t(\Delta)$ is only a function of Δ, i.e., independent of n.

The LOCAL model is the strongest model commonly in use—in particular, the size of each message and the amount of local computation in each communication round is unbounded—and this makes *lower bounds* in this model very widely applicable.

2 Overview

We will first show how to prove Theorem 1 for *deterministic* distributed algorithms; then we can use fairly standard techniques to extend the results to randomised distributed algorithms.

2.1 Deterministic models

Our lower bound builds on a long line of prior research. During the course of the proof, we will visit each of the following deterministic models (see Figure 1), whose formal definitions are given in Section 3.

ID: *Deterministic* LOCAL. Each node has a unique identifier [22, 28]. This is the standard model in the field of deterministic distributed algorithms.

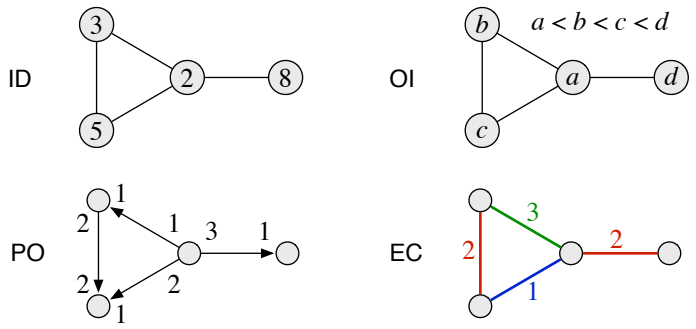


Figure 1: Deterministic models that are discussed in this work.

OI: *Order-invariance.* The output of an algorithm is not allowed to change if we relabel the nodes while preserving the relative order of the labels [25]. Equivalently, the algorithm can only compare the identifiers, not access their numerical value.

PO: *Port numbering and orientation.* For each node, there is an ordering on the incident edges, and all edges carry an orientation [24]. Each node knows the orientations of the incident edges, and a node of degree d can refer to its incident edges with d distinct port numbers: incoming messages are labelled with the port numbers, and outgoing messages are addressed with port numbers.

EC: *Edge colouring.* A proper edge colouring with $O(\Delta)$ colours is given [15]. Each node knows the colours of the incident edges, and the edge colours play the same role as a port numbering in communication: incoming messages are labelled with the edge colours, and outgoing messages are addressed with edge colours.

There are two key differences between PO and EC: (1) The edge orientation provides additional symmetry-breaking information in PO, and this information is not available in EC. (2) For each edge $\{u, v\}$, nodes u and v can use the same label (edge colour) to refer to each other in EC, but possibly different labels (port numbers) to refer to each other in PO.

The models are listed here roughly in the order of decreasing strength. For example, the ID model is strictly stronger than OI, which is strictly stronger than PO. However, the EC model is not directly comparable:

- There are problems that are trivial to solve in ID, OI, and PO but impossible to solve in EC with any deterministic algorithm. A simple example is graph colouring in 1-regular graphs.
- There are also problems that can be solved with a local algorithm in EC but they do not admit a local algorithm in ID or OI, nor any algorithm in PO. A simple example is maximal matching in cycles: In the EC model we can find a maximal matching in $O(1)$ rounds by iterating through the colour classes and greedily selecting all available edges in each class [15]. In essence, we can use the existence of the edge colouring to circumvent Linial's lower bound [22].

2.2 Proof outline

In short, our proof is an application of techniques that were introduced in two of our earlier works [9, 15], followed by a straightforward reduction that extends the result to randomised algorithms.

A weak deterministic lower bound. In our prior work [15] we showed that *maximal matchings* cannot be computed in time $o(\Delta)$ in the EC model. The lower-bound construction there is a regular graph, and as such, tells us very little about the fractional matching problem, since maximal fractional matchings are trivial to compute in regular graphs.

Nevertheless, we use a similar *unfold-and-mix* argument on what will be called *loopy* EC*-graphs* to prove the following intermediate result in Section 4:

Step 1. *The maximal FM problem cannot be solved in time $o(\Delta)$ on loopy* EC*-graphs with deterministic distributed algorithms.*

The proof heavily exploits the limited symmetry breaking capabilities of the EC model. To continue, we need to argue that similar limitations exist in the ID model.

Strengthening the deterministic lower bound. To extend the lower bound to the ID model, we give a series of local simulation results

$$\mathsf{EC} \rightsquigarrow \mathsf{PO} \rightsquigarrow \mathsf{OI} \rightsquigarrow \mathsf{ID},$$

which state that a local algorithm for the maximal fractional matching problem in one model can be simulated fast in the model preceding it. That is, even though the models EC, PO, OI, and ID are generally very different, we show that the models are roughly equally powerful for computing a maximal fractional matching.

This part of the argument applies ideas from another prior work [9]. There, we showed that, for a large class of optimisation problems, a run-time preserving simulation $\mathsf{PO} \rightsquigarrow \mathsf{ID}$ exists. Unfortunately, the maximal fractional matching problem is not included in the scope of this result (fractional matchings are not *simple* in the sense of [9]), so we may not apply this result directly in a black-box fashion. In addition, this general result does not hold for the EC model.

Nevertheless, we spend Section 5 extending the methods of [9] and show that they can be tailored to the case of fractional matchings:

Step 2. *If the maximal FM problem can be solved in time $t(\Delta)$ on* ID*-graphs with deterministic distributed algorithms, then it can be solved in time $t(\Theta(\Delta))$ on loopy* EC*-graphs with deterministic distributed algorithms.*

Extending to randomised algorithms. So far we have proved Theorem 1 for deterministic algorithms. We will now extend the result so that it also holds for randomised algorithms (more specifically, for Monte Carlo algorithms that may fail to produce a feasible solution with some small probability).

The maximal FM problem is an example of a *locally checkable* problem: there is a local algorithm that can check whether a proposed function y is a feasible solution. It is known that randomness does not help a local algorithm in solving a locally checkable problem with bounded outputs [25]: if there is a $t(\Delta)$-time

randomised algorithm, then there is a $t(\Delta)$-time deterministic algorithm. In the FM problem, the local outputs are not necessarily bounded—a feasible solution y may use a superconstant number of bits to represent the edge weights $y(e)$. However, we can circumvent this technicality and strengthen Step 2 as follows; the details are given in Appendix A:

Step 2'. *If the maximal FM problem can be solved in time $t(\Delta)$ on* ID*-graphs with randomised distributed algorithms, then it can be solved in time $t(\Theta(\Delta))$ on loopy* EC*-graphs with deterministic distributed algorithms.*

In combination with Step 1, this proves Theorem 1.

3 Tools of the trade

Before we dive into the lower-bound proof, we recall the definitions of the four models mentioned in Section 2.1, and describe the standard tools that are used in their analysis. In what follows, we will only discuss deterministic algorithms—see Appendix A for the details on how to extend the results to randomised algorithms.

3.1 Locality

Distributed algorithms are typically described in terms of networked state machines: the nodes of a network exchange messages for t synchronous communication rounds after which they produce their local outputs (cf. Section 1.4).

Instead, for the purposes of our lower-bound analysis, we view an algorithm $\mathcal{A}$ simply as a function that associates to each pair (G, v) an output $\mathcal{A}(G, v)$ in a way that respects *locality*. That is, an algorithm $\mathcal{A}$ is said to have run-time t, if the output $\mathcal{A}(G, v)$ depends only on the information that is available in the radius-t neighbourhood around v. More formally, define

$$\tau_t(G, v)$$

as the restriction of the structure (G, v) to the t-neighbourhood of v. That is, $\tau_t(G, v)$ consists of the nodes and edges of G that are within distance t from v; here the distance of an edge $\{u, w\}$ from v is defined as $\min\{\mathrm{dist}(v, u), \mathrm{dist}(v, w)\} + 1$. A t-time algorithm $\mathcal{A}$ is then a mapping that satisfies

$$\mathcal{A}(G, v) = \mathcal{A}(\tau_t(G, v)). \tag{1}$$

(Note that, according to our definition, a node needs to use an algorithm with run-time at least 1 to learn its own degree. While this might seem restrictive, we adopt this convention merely for technical convenience: our algorithms are at most 1 round slower than algorithms in the more natural model where the degree is known at the start.)

The information contained in $\tau_t(G, v)$ depends on which of the models EC, PO, OI, and ID we are studying. For each model we define an associated graph class.

3.2 Identifier-based networks

An ID*-graph* is simply a graph G whose nodes are assigned unique identifiers; namely, $V(G) \subseteq \mathbb{N}$. Any mapping $\mathcal{A}$ satisfying (1) is a t-time ID-algorithm.

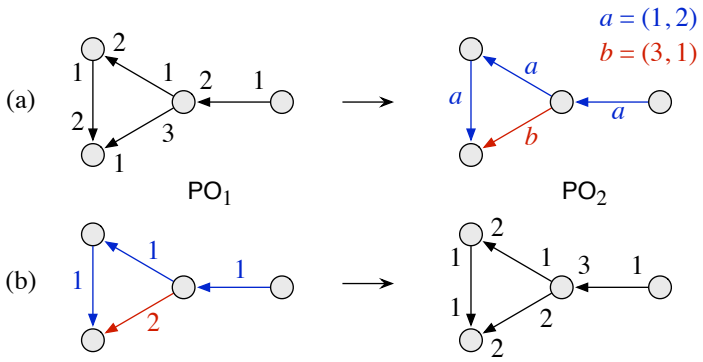


Figure 2: Two equivalent definitions of PO-graphs: (PO_1) a node of degree d can refer to incident edges with labels $1, 2, \ldots, d$; (PO_2) edges are coloured so that incoming edges have distinct colours and outgoing edges have distinct colours.

An OI-*graph* is an ordered graph $(G, \preceq)$ where $\preceq$ is a linear order on $V(G)$. An OI-algorithm $\mathcal{A}$ operates on OI-graphs in such a way that if $(G, \preceq, v)$ and $(G', \preceq', v')$ are isomorphic (as ordered structures), then $\mathcal{A}(G, \preceq, v) = \mathcal{A}(G', \preceq', v')$.

Every ID-graph G is naturally an OI-graph $(G, \leq)$ under the usual order $\leq$ on $\mathbb{N}$. In the converse direction, we often convert an OI-graph $(G, \preceq)$ into an ID-graph by specifying an ID-assignment $\varphi\colon V(G) \to \mathbb{N}$ that *respects* $\preceq$ in the sense that $v \preceq u$ implies $\varphi(v) \leq \varphi(u)$. The resulting ID-graph is denoted $\varphi(G)$.

3.3 Anonymous networks

On anonymous networks the nodes do not have identifiers. The only symmetry breaking information is now provided in an *edge colouring* of a suitable type. This means that whenever there is an isomorphism between (G, v) and (G', v') that preserves edge colours, we will have $\mathcal{A}(G, v) = \mathcal{A}(G', v')$.

An EC-*graph* carries a proper edge colouring $E(G) \to \{1, \ldots, k\}$, where $k = O(\Delta)$. That is, if two edges are adjacent, they have distinct colours.

A PO-*graph* is a directed graph whose edges are coloured in the following way: if (u, v) and (u, w) are outgoing edges incident to u, then they have distinct colours; and if (v, u) and (w, u) are incoming edges incident to u, then they have distinct colours. Thus, we may have (v, u) and (u, w) coloured the same.

We find it convenient to treat PO-graphs as edge-coloured digraphs, even if this view is slightly nonstandard. Usually, PO-graphs are defined as digraphs with a *port numbering*, i.e., each node is given an ordering of its neighbours. This is equivalent to our definition as it is easy to give local simulations in both directions: A port numbering gives rise to an edge colouring where an edge (u, v) is coloured with (i, j) if v is the i-th neighbour of u and u is the j-th neighbour of v (see Figure 2a). Conversely, we can derive a port numbering from an edge colouring—using some agreed-upon ordering of the edge colours, first take all outgoing edges ordered by their colours, and then take all incoming edges ordered by their colours (see Figure 2b). (Note that this does not give a one-to-one correspondence between port-numbered graphs and edge-coloured graphs, but what matters is that we can simulate any algorithm designed for one model in the other model with the same run-time).

We are not done with defining EC and PO algorithms. We still need to restrict their power by requiring that their outputs are *invariant under graph lifts*, as defined next.

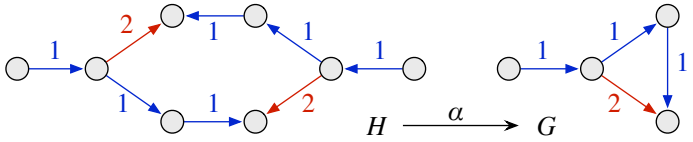


Figure 3: H is a lift of G.

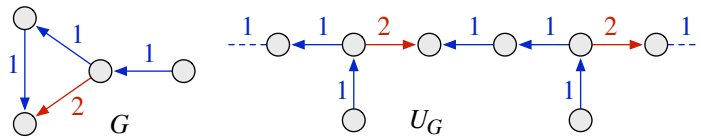


Figure 4: Universal cover U_G of G.

3.4 Lifts

A graph H is said to be a *lift* of another graph G if there exists an onto graph homomorphism $\alpha\colon V(H) \to V(G)$ that is a *covering map*, i.e., α preserves node degrees, $\deg_H(v) = \deg_G(\alpha(v))$; see Figure 3. Our discussion of lifts always takes place in either EC or PO; in this context we require that a covering map preserves edge colours.

The defining characteristic of anonymous models is that the output of an algorithm is invariant under taking lifts. That is, if $\alpha\colon V(H) \to V(G)$ is a covering map, then

$$\mathcal{A}(H, v) = \mathcal{A}(G, \alpha(v)), \qquad \text{for each } v \in V(H). \tag{2}$$

Since an isomorphism between H and G is a special case of a covering map, the condition (2) generalises the discussion in Section 3.3. We will be exploiting this limitation extensively in analysing the models EC and PO.

Graphs are partially ordered by the *lift* relation. For any connected graph G, there are two graphs U_G and F_G of special interest that are related to G via lifts.

Universal cover U_G. The *universal cover* U_G of G is an unfolded tree-like version of G; see Figure 4. More precisely, U_G is the unique tree that is a lift of G. Thus, if G is a tree, $U_G = G$; if G has cycles, U_G is infinite. In passing from G to U_G we lose all the cycle structure that is present in G. The universal cover is often used to model the information that a distributed algorithm—even with unlimited running time—is able to collect on an anonymous network [2].

Factor graph F_G. The *factor graph* F_G of G is the smallest graph F such that G is a lift of F; see Figure 5. In general, F_G is a multigraph with loops and parallel edges. It is the most concise representation of all the global symmetry breaking information available in G. For example, in the extreme case when G is vertex-transitive, F_G consists of just one node and some loops.

An input graph to an algorithm is always required to be simple (no loops or parallel edges). However, we find it convenient to *virtually* run EC and PO-algorithms $\mathcal{A}$ on multigraphs F with the understanding that the output $\mathcal{A}(F, v)$ is interpreted as if we had run $\mathcal{A}$ on a simple lift of F and then mapped the solution back to F according to (2). That is, to determine $\mathcal{A}(F, v)$ where F is a multigraph, do the following:

1. Lift F to a simple graph G (e.g., take $G = U_F$) via some $\alpha\colon V(G) \to V(F)$.

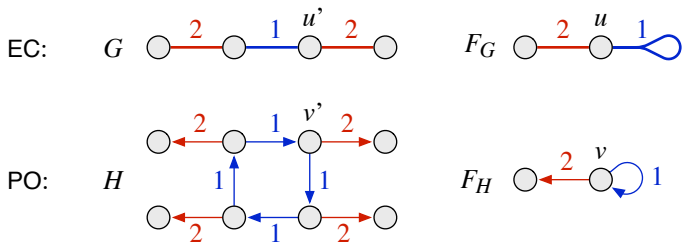


Figure 5: Factor graphs and loops. We follow the convention that undirected loops in EC-graphs count as a single incident edge, while directed loops in PO-graphs count as two incident edges: an incoming edge and an outgoing edge. In this example, both u and its preimage u' are nodes of degree 2; they are incident to one edge of colour 1 and one edge of colour 2. Both v and its preimage v' are nodes of degree 3; they are incident to two outgoing edges of colours 1 and 2, and one incoming edge of colour 1.

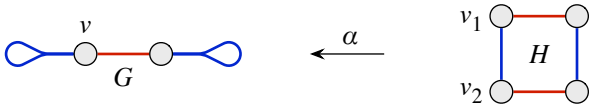


Figure 6: EC-graph G is loopy. Assume that an EC-algorithm $\mathcal{A}$ produces an output in which node v is unsaturated. Then we can construct a simple EC-graph H that is a lift of G via $\alpha\colon V(H) \to V(G)$ such that $\alpha(v_1) = \alpha(v_2) = v$ and $\{v_1, v_2\} \in E(H)$. If we apply $\mathcal{A}$ to H, both v_1 and v_2 are unsaturated; hence $\mathcal{A}$ fails to produce a maximal FM.

2. Execute $\mathcal{A}$ on (G, u) for some $u \in \alpha^{-1}(v)$.
3. Interpret the output of u as an output of v.

In what follows we refer to multigraphs simply as graphs.

3.5 Loops

In EC-graphs, a single loop on a node contributes +1 to its degree, whereas in PO-graphs, a single (directed) loop contributes +2 to its degree, once for the tail and once for the head. This is reflected in the way we draw loops—see Figure 5.

The loop count on a node $v \in V(G)$ measures the inability of v to break local symmetries. Indeed, if v has ℓ loops, then in any simple lift H of G each node $u \in V(H)$ that is mapped to v by the covering map will have ℓ distinct neighbours $w_1, \ldots, w_\ell$ that, too, get mapped to v. Thus, an anonymous algorithm is forced to have the same output on u as on each of $w_1, \ldots, w_\ell$.

We consider loops as an important resource.

Definition 1. An edge-coloured graph G is called *k-loopy* if each node in F_G has at least k loops. A graph is simply *loopy* if it is 1-loopy.

When computing maximal fractional matchings on a loopy graph G, an anonymous algorithm must saturate all the nodes. Otherwise, if $v \in V(G)$ is a node that does not get saturated, the loopiness of G implies that v has a neighbour u (can be $u = v$ via a loop) that produces the same output as v. But now neither endpoint of $\{u, v\}$ is saturated, which contradicts maximality; see Figure 6. We record this observation.

Lemma 1. *Any* EC*-algorithm for the maximal FM problem computes a fully saturated FM on loopy* EC*-graphs.* □

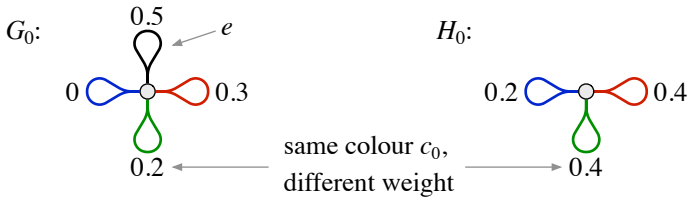


Figure 7: Base case. By removing a loop e with a non-zero weight, we force the algorithm to change the weight of at least one edge that is present in both G_0 and H_0.

4 Lower bound in EC

In this section we carry out Step 1 of our lower-bound plan. To do this we extend the previous lower bound result [15] to the case of maximal fractional matchings.

4.1 Strategy

Let $\mathcal{A}$ be any EC-algorithm computing a maximal fractional matching. We construct inductively a sequence of EC-graph pairs

$$(G_i, H_i), \quad i = 0, 1, \ldots, \Delta - 2,$$

that witness $\mathcal{A}$ having run-time greater than i. Each of the graphs G_i and H_i will have maximum degree at most Δ, so for $i = \Delta - 2$, we will have the desired lower bound. More precisely, we show that there are nodes $g_i \in V(G_i)$ and $h_i \in V(H_i)$ satisfying the following property:

(P1) The i-neighbourhoods $\tau_i(G_i, g_i)$ and $\tau_i(H_i, h_i)$ are isomorphic, yet

$$\mathcal{A}(G_i, g_i) \neq \mathcal{A}(H_i, h_i).$$

Moreover, there is a loop of some colour c_i adjacent to both g_i and h_i such that the outputs disagree on its weight.

We will also make use of the following additional properties in the construction:

(P2) The graphs G_i and H_i are $(\Delta - 1 - i)$-loopy. Consequently, $\mathcal{A}$ will saturate all their nodes by Lemma 1.

(P3) When the loops are ignored, both G_i and H_i are trees.

4.2 Base case ($i = 0$)

Let G_0 consist of a single node v that has Δ differently coloured loops. When $\mathcal{A}$ is run on G_0, it saturates v by assigning at least one loop e a non-zero weight; see Figure 7. Letting $H_0 := G_0 - e$ it is now easy to check that the pair (G_0, H_0) satisfies (P1–P3) for $g_0 = h_0 = v$. For example, we have $\tau_0(G_0, v) \cong \tau_0(H_0, v)$ because both 0-neighbourhoods consist of a single isolated node of degree 0. Recall our convention that the loops are at distance 1 from v.

4.3 Inductive step

Suppose (G_i, H_i) is a pair satisfying (P1–P3). For convenience, we write G, H, g, h, and c in place of G_i, H_i, g_i, h_i, and c_i. Also, we let $e \in E(G)$ and $f \in E(H)$ be the colour-c loops adjacent to g and h to which $\mathcal{A}$ assigns different weights.

To construct the pair (G_{i+1}, H_{i+1}) we unfold and mix; see Figure 8.

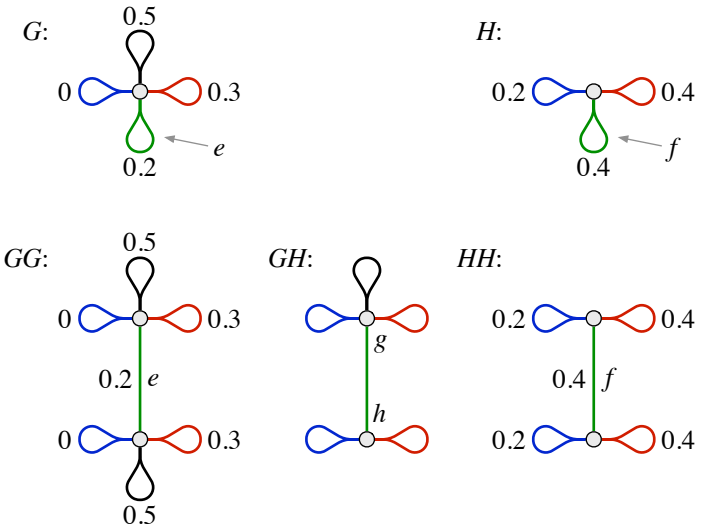


Figure 8: Unfold and mix. The weights of e and f differ; hence the weight of $\{g, h\}$ is different from the weight of e or f.

Unfolding. First, we unfold the loop e in G to obtain a 2-lift GG of G. That is, GG consists of two disjoint copies of $G - e$ and a new edge of colour c (which we still call e) that connects the two copies of g in GG. For notational purposes, we fix some identification $V(G) \subseteq V(GG)$ so that we can easily talk about one of the copies. Similarly, we construct a 2-lift HH of H by unfolding the loop f.

Recall that $\mathcal{A}$ cannot tell apart G from GG, or H from HH. In particular $\mathcal{A}$ continues to assign unequal weights to e and f in these lifts.

Mixing. Next, we mix together the graphs GG and HH to obtain a graph GH defined as follows: GH contains a copy of $G - e$, a copy of $H - f$, and a new colour-c edge that connects the nodes g and h. For notational purposes, we let $V(GH) := V(G) \cup V(H)$, where we tacitly assume that $V(G) \cap V(H) = \varnothing$.

Analysis. Consider the weight that $\mathcal{A}$ assigns to the colour-c edge $\{g, h\}$ in GH. Since $\mathcal{A}$ gives the edges e and f different weights in GG and HH, we must have that the weight of $\{g, h\}$ differs from the weight of e or the weight of f (or both). We assume the former (the latter case is analogous), and argue that the pair

$$(G_{i+1}, H_{i+1}) := (GG, GH)$$

satisfies the properties (P1–P3). It is easy to check that (P2) and (P3) are satisfied by the construction; it remains is to find the nodes $g_{i+1} \in V(GG)$ and $h_{i+1} \in V(GH)$ that satisfy (P1).

To this end, we exploit the following property of fractional matchings:

Fact 1 (Propagation principle)**.** *Assume that y and y' are fractional matchings that saturate a node v. If y and y' disagree on some edge incident to v, there must be another edge incident to v where y and y' disagree.*

Our idea is to apply this principle in a fully saturated graph, where the disagreements propagate until they are resolved at a loop; this is where we locate g_{i+1} and h_{i+1}. See Figure 9 for an example.

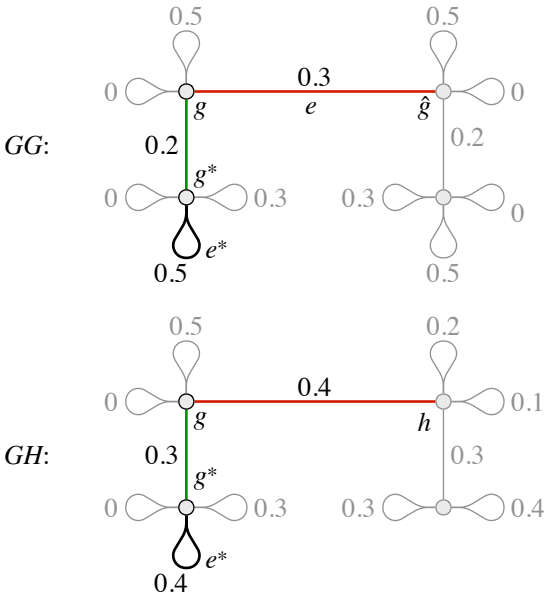


Figure 9: Propagation. The weights of e and $\{g, h\}$ differ. We apply the propagation principle towards the common part G that is shared by GG and GH. The graphs are loopy and hence all nodes are saturated by $\mathcal{A}$; we will eventually find a loop e^* that is present in both GG and GH, with different weights.

We consider the following fully saturated fractional matchings on G:

$$y = \text{the FM determined by } \mathcal{A}\text{'s output on the nodes } V(G) \text{ in } GG,$$
$$y' = \text{the FM determined by } \mathcal{A}\text{'s output on the nodes } V(G) \text{ in } GH.$$

Starting at the node $g \in V(G)$ we already know by assumption that y and y' disagree on the colour-c edge incident to g. Thus, by the propagation principle, y and y' disagree on some other edge incident to g. If this edge is not a loop, it connects to a neighbour $g' \in V(G)$ of g and the argument can be continued: because y and y' disagree on $\{g, g'\}$, there must be another edge incident to g' where y and y' disagree, and so on. Since G does not have any cycles (apart from the loops), this process has to terminate at some node $g^* \in V(G)$ such that y and y' disagree on a loop $e^* \neq e$ incident to g^*. Note that e^* is a loop in both GG and GH, too. Thus, we have found our candidate $g_{i+1} = h_{i+1} = g^*$.

To finish the proof, we need to show that

$$\tau_{i+1}(GG, g^*) \cong \tau_{i+1}(GH, g^*). \tag{3}$$

The critical case is when $g^* = g$ as this node is the closest among $V(G)$ to seeing the topological differences between the graphs GG and GH. Starting from g and stepping along the colour-c edge towards the differences, we arrive, in GG, at a node $\hat{g}$ that is a copy of $g \in V(G)$, and in GH, at the node h. But these nodes satisfy

$$\tau_i(GG, \hat{g}) \cong \tau_i(GH, h)$$

by our induction assumption. Using this, (3) follows.

5 Local simulations

Now that we have an $\Omega(\Delta)$ time lower bound in the EC model, our next goal is to extend this result to the ID model. In this section we implement Step 2 of our

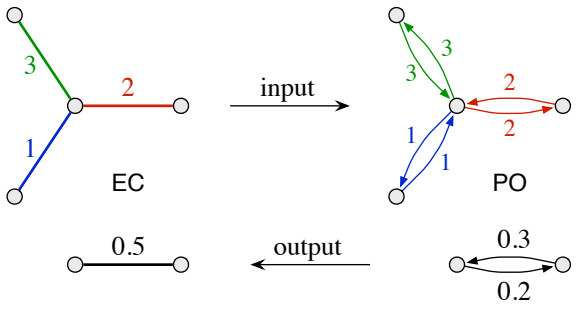


Figure 10: EC $\rightsquigarrow$ PO. Mapping an EC-graph G into a PO-graph $G_{\leftrightarrows}$, and mapping the output of a PO-algorithm back to the original graph.

plan and give a series of local simulations

$$\mathsf{EC} \rightsquigarrow \mathsf{PO} \rightsquigarrow \mathsf{OI} \rightsquigarrow \mathsf{ID}.$$

Here, each simulation preserves the running time of an algorithm up to a constant factor. In particular, together with Step 1, this will imply the $\Omega(\Delta)$ time lower bound in the ID model.

5.1 Simulation EC $\rightsquigarrow$ PO

We start with the easiest simulation. Suppose there is a t-time PO-algorithm for the maximal fractional matching problem on graphs of maximum degree Δ; we describe a t-time EC-algorithm for graphs of maximum degree $\Delta/2$.

The local simulation is simple; see Figure 10. On an EC-graph G we interpret each edge $\{u, v\}$ of colour c as two directed edges (u, v) and (v, u), both of colour c; this interpretation makes G into a PO-graph $G_{\leftrightarrows}$. We can now locally simulate the PO-algorithm on $G_{\leftrightarrows}$ to obtain an FM y as output. Finally, we transform y back to an FM of G: the edge $\{u, v\}$ is assigned weight $y(u, v) + y(v, u)$.

5.2 Tricky identifiers

When we are computing a maximal fractional matching $y\colon E(G) \to [0, 1]$, we have, a priori, infinitely many choices for the weight $y(e)$ of an edge. For example, in a path on nodes v_1, v_2, and v_3, we can freely choose $y(\{v_1, v_2\}) \in [0, 1]$ provided we set $y(\{v_2, v_3\}) = 1 - y(\{v_1, v_2\})$. In particular, an ID-algorithm can output edge weights that depend on the node identifiers whose magnitude is not bounded.

Unbounded outputs are tricky from the perspective of proving lower bounds (we will have the same challenge in Appendix A when we deal with randomness). The main result of the recent work [9] is a run-time preserving local simulation PO $\rightsquigarrow$ ID, but the result only holds under the assumption that the solution can be encoded using finitely many values per node on graphs of maximum degree Δ. This restriction has its source in an earlier local simulation OI $\rightsquigarrow$ ID due to Naor and Stockmeyer [25] that is crucially using Ramsey's theorem. In fact, these two local simulation results fail if unbounded outputs are allowed; counterexamples include even natural graph problems [14].

In conclusion, we need an ad hoc argument to establish that an ID-algorithm cannot benefit from unique identifiers in the case of the maximal fractional matching problem.

5.3 Simulation $\mathsf{PO} \rightsquigarrow \mathsf{OI}$

Before we address the question of simulating ID-algorithms, we first salvage one part of the result in [9]: there is local simulation $\mathsf{PO} \rightsquigarrow \mathsf{OI}$ that applies to many locally checkable problems, regardless of the size of the output encoding. Even though this simulation works off-the-shelf in our present setting, we cannot use this result in a black-box fashion, as we need to access its inner workings later in the analysis. Thus, we proceed with a self-contained proof.

The following presentation is considerably simpler than that in [9], since we are only interested in a simulation that produces a *locally maximal* fractional matching, not in a simulation that also provides approximation guarantees on the *total weight*, as does the original result.

PO-checkability. Maximal fractional matchings are not only locally checkable, but also PO-*checkable*: there is a local PO-algorithm that can check whether a given y is a maximal FM. An important consequence of PO-checkability is that if H is a lift of G then any PO-algorithm produces a feasible solution on H if and only if it produces a feasible solution on G.

Order homogeneity. The key to the simulation $\mathsf{PO} \rightsquigarrow \mathsf{OI}$ is a *canonical linear order* that can be computed for any tree-like PO-neighbourhood. To define this ordering, let d denote the maximum number of edge colours appearing in the input PO-graphs that have maximum degree Δ, and let T denote the infinite $2d$-regular d-edge-coloured PO-tree. We fix a homogeneous linear order for T:

Lemma 2. *There is a linear order $\preceq$ on $V(T)$ such that all the ordered neighbourhoods $(T, \preceq, v)$, $v \in V(T)$, are pairwise isomorphic (i.e., up to any radius).*

Proof. The tree T can be thought of as a Cayley graph of the free group on d generators, and the free group admits a linear order that is invariant under the group acting on itself by multiplication; for details, see Neumann [26] and the discussion in [9, §5]. □

For an alternative, combinatorial proof of Lemma 2, Appendix B.

Simulation. Let $\mathcal{A}_{\mathsf{OI}}$ be any t-time OI-algorithm solving a PO-checkable problem; we describe a t-time PO-algorithm $\mathcal{A}_{\mathsf{PO}}$ solving the same problem.

The algorithm $\mathcal{A}_{\mathsf{PO}}$ operates on a PO-graph G as follows; see Figure 11. Given a PO-neighbourhood $\tau := \tau_t(U_G, v)$, we first embed τ in T: we choose an arbitrary node $u \in V(T)$, identify v with u, and let the rest of the embedding $\tau \subseteq (T, u)$ be dictated uniquely by the edge colours. We then use the ordering $\preceq$ inherited from T to order the nodes of τ. By Lemma 2, the resulting structure $(\tau, \preceq)$ is independent of the choice of u, i.e., the isomorphism type of $(\tau, \preceq)$ is only a function of τ. Finally, we simulate

$$\mathcal{A}_{\mathsf{PO}}(\tau) := \mathcal{A}_{\mathsf{OI}}(\tau, \preceq). \tag{4}$$

To see that the output of $\mathcal{A}_{\mathsf{PO}}$ is feasible, we argue as follows. Embed the universal cover U_G as a subgraph of $(T, \preceq)$ in a way that respects edge colours. Again, all possible embeddings are isomorphic; we call the inherited ordering

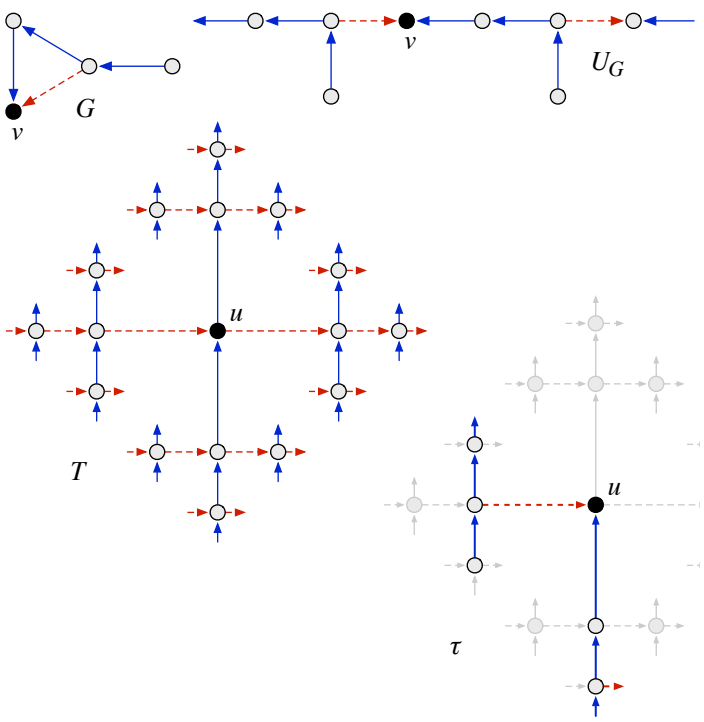


Figure 11: Given a PO-graph G, algorithm $\mathcal{A}_{\mathsf{PO}}$ simulates the execution of $\mathcal{A}_{\mathsf{OI}}$ on OI-graph τ. The linear order on $V(\tau)$ is inherited from the regular tree T. As T is homogeneous, the linear order does not depend on the choice of node u in T.

$(U_G, \preceq)$ the *canonical ordering* of U_G. Our definition of $\mathcal{A}_{\mathsf{PO}}$ and the order homogeneity of $(T, \preceq)$ now imply that

$$\mathcal{A}_{\mathsf{PO}}(U_G, v) = \mathcal{A}_{\mathsf{OI}}(U_G, \preceq, v) \qquad \text{for all } v \in V(U_G).$$

Therefore, the output of $\mathcal{A}_{\mathsf{PO}}$ is feasible on U_G. Finally, by PO-checkability, the output of $\mathcal{A}_{\mathsf{PO}}$ is feasible also on G, as desired.

5.4 Simulation OI $\rightsquigarrow$ ID

The reason why an ID-algorithm $\mathcal{A}$ cannot benefit from unbounded identifiers is due to the propagation principle. We formalise this in two steps.

(i) We use the Naor–Stockmeyer OI $\rightsquigarrow$ ID result to see that $\mathcal{A}$ can be forced to output fully saturated FMs on so-called *loopy* OI-neighbourhoods.

(ii) We then observe that, on these neighbourhoods, $\mathcal{A}$ behaves like an OI-algorithm: $\mathcal{A}$'s output cannot change if we relabel a node in an order-preserving fashion, because the changes in the output would have to propagate outside of $\mathcal{A}$'s run-time.

That is, our simulation OI $\rightsquigarrow$ ID will work only on certain types of neighbourhoods (in contrast to our previous simulations), but this will be sufficient for the purposes of the lower bound proof.

Step (i). Let $\mathcal{A}$ be a t-time ID-algorithm that computes a maximal fractional matching on graphs of maximum degree Δ.

From $\mathcal{A}$ we can derive, by a straightforward simulation, a t-time *binary-valued* ID-algorithm $\mathcal{A}^*$ that indicates whether $\mathcal{A}$ saturates a node. That is, $\mathcal{A}^*(G, v) := 1$ if $\mathcal{A}$ saturates v in G, otherwise $\mathcal{A}^*(G, v) := 0$. Such saturation indicators $\mathcal{A}^*$ were considered previously in [3, §4].

Because (and *only* because) $\mathcal{A}^*$ outputs finitely many values, we can now apply the Ramsey technique of Naor and Stockmeyer [25, Lemma 3.2]. To avoid notational clutter, we use a version of their result that follows from the application of the infinite Ramsey's theorem (rather than the finite):

Lemma 3 (Naor and Stockmeyer)**.** *There is an infinite set $I \subseteq \mathbb{N}$ such that $\mathcal{A}^*$ is an* OI*-algorithm when restricted to graphs whose identifiers are in I.* □

We say that $\tau_t(U_G, \preceq, v)$ is a *loopy* OI-neighbourhood if G is a loopy PO-graph and $(U_G, \preceq)$ is the canonically ordered universal cover of G. We also denote by $B_t(v) \subseteq V(U_G)$ the node set of $\tau_t(U_G, v)$.

Our saturation indicator $\mathcal{A}^*$ is useful in proving the following lemma, which encapsulates step (i) of our argument.

Lemma 4. *Let $\tau := \tau_t(U_G, \preceq, v)$ be loopy. If $\varphi\colon B_t(v) \to I$ is an* ID*-assignment to the nodes of τ that respects $\preceq$, then $\mathcal{A}$ saturates v under φ.*

Proof. By loopiness of G, the node v has a neighbour $u \in V(U_G)$ such that $\tau_t(U_G, v) \cong \tau_t(U_G, u)$ as PO-neighbourhoods. By order homogeneity, $\tau_t(U_G, \preceq, v) \cong \tau_t(U_G, \preceq, v)$ as OI-neighbourhoods. By Lemma 3, this forces $\mathcal{A}^*$ to output the same on v and u under any ID-assignment $\varphi'\colon B_t(v) \cup B_t(u) \to I$ that respects $\preceq$. But $\mathcal{A}^*$ cannot output two adjacent 0's if $\mathcal{A}$ is to produce a maximal fractional matching. Hence, $\mathcal{A}^*$ outputs 1 on $\varphi'(\tau)$. Finally, by order-invariance, $\mathcal{A}^*$ outputs 1 on $\varphi(\tau)$, which proves the claim. □

Step (ii). Define J as an infinite subset of I that is obtained by picking every $(m+1)$-th identifier from I, where m is the maximum number of nodes in a $(2t+1)$-neighbourhood of maximum degree Δ. That is, for any two $j, j' \in J$, $j < j'$, there are m distinct identifiers $i \in I$ with $j < i < j'$.

The next lemma states that $\mathcal{A}$ behaves like an OI-algorithm on loopy neighbourhoods that have identifiers from J.

Lemma 5. *Assume that $\tau := \tau_t(U_G, \preceq, v)$ is loopy. If $\varphi_1, \varphi_2\colon B_t(v) \to J$ are any two* ID*-assignments that respect $\preceq$, then $\mathcal{A}(\varphi_1(\tau)) = \mathcal{A}(\varphi_2(\tau))$.*

Proof. We first consider the case where φ_1 and φ_2 disagree only on a single node $v^* \in B_t(v)$. Towards a contradiction suppose that

$$\mathcal{A}(\varphi_1(\tau)) \neq \mathcal{A}(\varphi_2(\tau)). \tag{5}$$

We start with partial ID-assignments for U_G that are defined on the nodes $B_{2t+1}(v)$; this will suffice for running $\mathcal{A}$ on the nodes $B_{t+1}(v)$. Indeed, because $J \subseteq I$ is sufficiently sparse, we can extend φ_1 and φ_2 into assignments $\bar{\varphi}_1, \bar{\varphi}_2\colon B_{2t+1}(v) \to I$ such that

- $\bar{\varphi}_1$ and $\bar{\varphi}_2$ respect $\preceq$, and
- $\bar{\varphi}_1$ and $\bar{\varphi}_2$ still disagree only on the node v^*.

Let y_i, $i = 1, 2$, be the fractional matching defined on the edges incident to $B_{t+1}(v)$ that is determined by the output of $\mathcal{A}$ on the nodes $B_{t+1}(v)$ under the assignment $\bar{\varphi}_i$. By Lemma 4, all the nodes $B_{t+1}(v)$ are saturated in both y_1 and y_2.

Let $D \subseteq U_G$ be the subgraph consisting of the edges e with $y_1(e) \neq y_2(e)$ and of the nodes that are incident to such edges; by (5), we have $v \in V(D)$. Now we can reinterpret the propagation principle from Section 4:

Fact 2 (Propagation principle)**.** *For each node $u \in B_{t+1}(v) \cap V(D)$ we have $\deg_D(u) \geq 2$.*

Using the fact that $D \subseteq U_G$ is a tree, we can start a simple walk at $v \in V(D)$, take the first step away from v^*, and finally arrive at a node $u \in B_{t+1}(v) \cap V(D)$ that has $\operatorname{dist}(u, v^*) \geq t + 1$, i.e, the node u does not see the difference between the assignments $\bar{\varphi}_1$ and $\bar{\varphi}_2$. But this is a contradiction: as the t-neighbourhoods $\bar{\varphi}_i(\tau_t(U_G, u))$, $i = 1, 2$, are the same, so should be the weights output by $\mathcal{A}$.

General case. If $\varphi_1, \varphi_2 \colon B_t(v) \to J$ are any two assignments respecting $\preceq$, they can be related to one another by a series of assignments

$$\varphi_1 = \pi_1, \pi_2, \ldots, \pi_k = \varphi_2,$$

where any two consecutive assignments π_i and π_{i+1} both respect $\preceq$ and disagree on exactly one node. Thus, the claim follows from the analysis above. □

Let $\mathcal{A}_{\mathsf{OI}}$ be any t-time OI-algorithm that agrees with the order-invariant output of $\mathcal{A}$ on loopy OI-neighbourhoods that have identifiers from J. We now obtain the final form of our $\mathsf{OI} \rightsquigarrow \mathsf{ID}$ simulation:

Corollary 1. *If G is a loopy PO-graph, $\mathcal{A}_{\mathsf{OI}}$ produces a maximal fractional matching on the canonically ordered universal cover $(U_G, \preceq)$.*

Proof. The claim follows by a standard argument [25, Lemma 3.2] from two facts: J is large enough; and maximal fractional matchings are locally checkable. □

5.5 Concluding Theorem 1

To get the final lower bound of Theorem 1 we reason backwards. We will first consider the case of deterministic algorithms. Assume that $\mathcal{A}$ is a t-time ID-algorithm that computes a maximal fractional matching on any graph of maximum degree Δ.

$\mathsf{OI} \rightsquigarrow \mathsf{ID}$: Corollary 1 above gives us a t-time OI-algorithm $\mathcal{A}_{\mathsf{OI}}$ that computes a maximal fractional matching on the canonically ordered universal cover $(U_G, \preceq)$ for any loopy PO-graph G of maximum degree Δ.

$\mathsf{PO} \rightsquigarrow \mathsf{OI}$: Simulation (4) in Section 5.3 queries the output of $\mathcal{A}_{\mathsf{OI}}$ only on $(U_G, \preceq)$. This gives us a t-time PO-algorithm $\mathcal{A}_{\mathsf{PO}}$ that computes a maximal fractional matching on any loopy PO-graph G of maximum degree Δ.

$\mathsf{EC} \rightsquigarrow \mathsf{PO}$: The simple simulation in Section 5.1 gives us a t-time EC-algorithm $\mathcal{A}_{\mathsf{EC}}$ that computes a maximal fractional matching on any loopy EC-graph G of maximum degree $\Delta/2$.

But now we can use the construction of Section 4: there is a loopy EC-graph of maximum degree $\Delta/2$ where $\mathcal{A}_{\mathsf{EC}}$ runs for $\Omega(\Delta)$ rounds. Hence the running time of $\mathcal{A}$ is also $\Omega(\Delta)$.

This completes the proof of Theorem 1 for deterministic algorithms—Appendix A shows how to extend it to randomised Monte Carlo algorithms.

6 Discussion

We have now a complete characterisation of the distributed time complexity of maximal fractional matchings in the region $\Delta \ll n$:

1. There is a deterministic distributed algorithm that finds a maximal fractional matching in $O(\Delta)$ rounds, independently of n.

2. There is no (deterministic or randomised) distributed algorithm that finds a maximal fractional matching in $o(\Delta)$ rounds, independently of n.

Any maximal matching is also a maximal fractional matching. However, our lower bound does not have any nontrivial implications on the distributed time complexity of maximal matchings: Linial's lower bound [22] already shows that there is no distributed algorithm that finds a maximal matching in $o(\Delta)$ rounds, independently of n.

As discussed in Section 1.1, a major open question is whether there is a distributed algorithm that finds a maximal matchings in $o(\Delta) + O(\log^* n)$ rounds. A more careful analysis of our construction would show that a maximal (fractional) matching cannot be found in $o(\Delta) + o(\log^* n)$ rounds, but the proof cannot be extended directly to algorithms with a running time of $o(\Delta) + O(\log^* n)$.

Informally, the key obstacle is that we can no longer argue that ID and OI are equally strong from the perspective of algorithms with a running time of $\Theta(\log^* n)$: in the ID model, such algorithms can produce, e.g., a vertex colouring, which may possibly help with symmetry breaking. Therefore, a natural first step towards stronger lower bounds would be to extend the techniques of Section 4 so that they hold also for vertex-coloured graphs. This suggests the following concrete open question that is currently just beyond the reach of our techniques:

- Is there a deterministic distributed algorithm that finds a maximal matching in $o(\Delta)$ rounds in bipartite, 2-vertex-coloured graphs in the port-numbering model?

Note that there is a simple algorithm that solves the problem in $O(\Delta)$ rounds: nodes of colour 1 send proposals to their neighbours, one by one, until one of the proposals is accepted, and nodes of colour 2 accept the first proposal that they get, breaking ties with port numbers [12]. However, it is not known if the problem can be solved in $o(\Delta)$ rounds, independently of n. Proving such a lower bound could be a stepping stone towards resolving the open questions related to the distributed time complexity of maximal matchings, as well as other problems for which the fastest current algorithms have linear-in-Δ running times for $\Delta \ll n$.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback. The combinatorial proof in Appendix B is joint work with Christoph Lenzen and Roger Wattenhofer.

This work is supported in part by the Academy of Finland, Grants 132380 and 252018, and by the Research Funds of the University of Helsinki. Much of this research was done while the authors were affiliated with the Department of Computer Science, University of Helsinki.

References

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

[2] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 82–93. ACM Press, 1980. doi:10.1145/800141.804655.

[3] Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. A local 2-approximation algorithm for the vertex cover problem. In *Proc. 23rd International Symposium on Distributed Computing (DISC 2009)*, volume 5805 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2009. doi:10.1007/978-3-642-04355-0_21.

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

[5] Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

[6] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 321–330. IEEE Computer Society Press, 2012. doi:10.1109/FOCS.2012.60.

[7] Andrzej Czygrinow, Michał Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-87779-0_6.

[8] Patrik Floréen, Marja Hassinen, Joel Kaasinen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximability of max-min and min-max linear programs. *Theory of Computing Systems*, 49(4):672–697, 2011. doi:10.1007/s00224-010-9303-6.

[9] Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. *Journal of the ACM*, 60(5):39:1–23, 2013. doi:10.1145/2528405. arXiv:1201.6675.

[10] Mika Göös, Juho Hirvonen, and Jukka Suomela. Linear-in-Δ lower bounds in the LOCAL model. In *Proc. 33rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2014)*, pages 86–95. ACM Press, 2014. doi:10.1145/2611462.2611467. arXiv:1304.1007.

[11] Mika Göös and Jukka Suomela. No sublogarithmic-time approximation scheme for bipartite vertex cover. In *Proc. 26th International Symposium on Distributed Computing (DISC 2012)*, volume 7611 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2012. doi:10.1007/978-3-642-33651-5_13. arXiv:1205.4605.

[12] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, pages 219–225. Society for Industrial and Applied Mathematics, 1998.

[13] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001. doi:10.1137/S0895480100373121.

[14] Henning Hasemann, Juho Hirvonen, Joel Rybicki, and Jukka Suomela. Deterministic local algorithms, unique identifiers, and fractional graph colouring. In *Proc. 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2012)*, volume 7355 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2012. doi:10.1007/978-3-642-31104-8_5.

[15] Juho Hirvonen and Jukka Suomela. Distributed maximal matching: greedy is optimal. In *Proc. 31st Annual ACM Symposium on Principles of Distributed Computing (PODC 2012)*, pages 165–174. ACM Press, 2012. doi:10.1145/2332432.2332464. arXiv:1110.0367.

[16] Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986. doi:10.1016/0020-0190(86)90144-4.

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

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

[19] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: lower and upper bounds, 2010. arXiv:1011.5470.

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

[21] Christoph Lenzen and Roger Wattenhofer. Leveraging Linial's locality limit. In *Proc. 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 394–407. Springer, 2008. doi:10.1007/978-3-540-87779-0_27.

[22] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

[23] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. doi: 10.1137/0215074.

[24] Alain Mayer, Moni Naor, and Larry Stockmeyer. Local computations on static and dynamic graphs. In *Proc. 3rd Israel Symposium on the Theory of Computing and Systems (ISTCS 1995)*, pages 268–278. IEEE, 1995. doi: 10.1109/ISTCS.1995.377023.

[25] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

[26] B. H. Neumann. On ordered groups. *American Journal of Mathematics*, 71(1):1–18, 1949. doi:10.2307/2372087.

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

[28] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

A Derandomisation

As discussed in Section 5.2, unbounded outputs require special care. In this appendix we note that even though Naor and Stockmeyer [25] assume bounded outputs, their result on derandomising local algorithms applies in our setting, too.

Recall that a randomised algorithm is an ID-algorithm such that each node has in addition access to a source of random bits. Let $\mathcal{A}$ be a randomised $t(\Delta)$-time algorithm that computes a maximal FM on graphs of maximum degree Δ or possibly fails with some small probability. Given an assignment of random bit strings $\rho\colon V(G) \to \{0,1\}^*$ to the nodes of a graph G, denote by $\mathcal{A}^\rho$ the *deterministic* algorithm that computes as $\mathcal{A}$, but uses ρ for randomness.

The proof of Theorem 5.1 in [25] is using the following fact whose proof we reproduce here for convenience.

Lemma 6. *For every n, there is an n-set $S_n \subseteq \mathbb{N}$ of identifiers and an assignment $\rho_n\colon S_n \to \{0,1\}^*$ such that $\mathcal{A}^{\rho_n}$ is correct on all graphs that have identifiers from S_n.*

Proof. Denote by $k = k(n)$ the number of graphs G with $V(G) \subseteq \{1,\dots,n\}$. Let $X_1,\dots,X_q \subseteq \mathbb{N}$ be any q disjoint sets of size n. Suppose for the sake of contradiction that the claim is false for each X_i. That is, for any assignment $\rho\colon X_i \to \{0,1\}^*$ of random bits, $\mathcal{A}^\rho$ fails on at least one of the k many graphs G with $V(G) \subseteq X_i$. By averaging, this implies that for each i there is a particular graph G_i, $V(G_i) \subseteq X_i$, on which $\mathcal{A}$ fails with probability at least $1/k$. Consider the graph G that is the disjoint union of the graphs $G_1,\dots,G_q$. Since $\mathcal{A}$ fails independently on each of the components G_i, the failure probability on G is at least $1-(1-1/k)^q$. But this probability can be made arbitrarily close to 1 by choosing a large enough q, which contradicts the correctness of $\mathcal{A}$. □

The deterministic algorithms $\mathcal{A}^{\rho_n}$ allow us to again obtain a $t(\Delta)$-time OI-algorithm, which establishes the $\Omega(\Delta)$ lower bound for $\mathcal{A}$. Only small modifications to Section 5.4 are needed:

- **Step (i).** Instead of the infinite set $I \subseteq \mathbb{N}$ as previously provided by Lemma 3, we can use the finite Ramsey's theorem to find arbitrarily large sets $I_n \subseteq S_n$ (i.e., $|I_n| \to \infty$ as $n \to \infty$) with the property that $\mathcal{A}^{\rho_n}$ fully saturates the nodes of a loopy OI-neighbourhood that has identifiers from I_n (Lemma 4).
- **Step (ii).** Then, passing again to sufficiently sparse subsets $J_n \subseteq I_n$, we can reprove Lemma 5 and Corollary 1, which only require that J is large enough.

This concludes the lower bound proof for randomised LOCAL algorithms.

B Combinatorial proof of Lemma 2

In T there is a unique simple directed path $x \rightsquigarrow y$ between any two nodes $x, y \in V(T)$. We use $V(x \rightsquigarrow y)$ and $E(x \rightsquigarrow y)$ to denote the nodes and edges of the path. Also, we set $V_{\mathsf{in}}(x \rightsquigarrow y) := V(x \rightsquigarrow y) \setminus \{x, y\}$. We will assign to each path $x \rightsquigarrow y$ an integer value, denoted $[\![x \rightsquigarrow y]\!]$, which will determine the relative order of the endpoints.

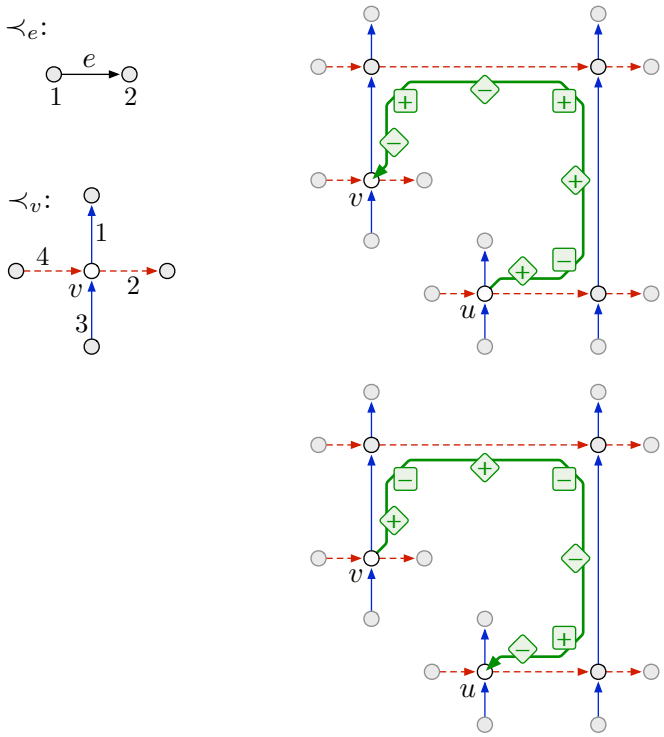


Figure 12: In this example, $[\![u \rightsquigarrow v]\!] = +1$, $[\![v \rightsquigarrow u]\!] = -1$, and hence $u \prec v$.

By definition, in the PO model, we are given the following linear orders:

- Each node $v \in V(T)$ has a linear order $\prec_v$ on its incident edges.
- Each edge $e \in E(T)$ has a linear order $\prec_e$ on its incident nodes.

For notational convenience, we extend these relations a little: for $v \in V_{\mathsf{in}}(x \rightsquigarrow y)$ we define $x \prec_v y \iff e \prec_v e'$, where e is the last edge on the path $x \rightsquigarrow v$ and e' is the first edge on the path $v \rightsquigarrow y$; similarly, for $e \in E(x \rightsquigarrow y)$, we define $x \prec_e y \iff x' \prec_e y'$, where $e = \{x', y'\}$ and x' and y' appear on the path $x \rightsquigarrow y$ in this order.

For any statement P, we will use the following type of Iverson bracket notation:

$$[P] := \begin{cases} +1 & \text{if } P \text{ is true,} \\ -1 & \text{if } P \text{ is false.} \end{cases}$$

We can now define

$$[\![x \rightsquigarrow y]\!] := \sum_{e \in E(x \rightsquigarrow y)} [x \prec_e y] \quad + \sum_{v \in V_{\mathsf{in}}(x \rightsquigarrow y)} [x \prec_v y]. \tag{6}$$

In particular, $[\![x \rightsquigarrow x]\!] = 0$. The linear order $\prec$ on $V(T)$ is now defined by setting

$$x \prec y \iff [\![x \rightsquigarrow y]\!] > 0.$$

See Figure 12. Next, we show that this is indeed a linear order.

Antisymmetry and totality. Since $[x \prec_v y] = -[y \prec_v x]$ and $[x \prec_e y] = -[y \prec_e x]$, we have the property that

$$[\![x \rightsquigarrow y]\!] = -[\![y \rightsquigarrow x]\!].$$

Moreover, if $x \neq y$, the first sum in (6) is odd iff the second sum in (6) is even. Therefore $[\![x \rightsquigarrow y]\!]$ is always odd; in particular, it is non-zero. These properties establish that either $x \prec y$ or $y \prec x$ (but never both).

Transitivity. Let $x, y, z \in V(T)$ be three distinct nodes with $x \prec y$ and $y \prec z$; we need to show that $x \prec z$. Denote by $v \in V(T)$ the unique node in the intersection of the paths $x \rightsquigarrow z$, $z \rightsquigarrow y$, and $y \rightsquigarrow x$.

Viewing the path $x \rightsquigarrow z$ piecewise as $x \rightsquigarrow v \rightsquigarrow z$ we write

$$[\![x \rightsquigarrow z]\!] = [\![x \rightsquigarrow v]\!] + [x \prec_v z] + [\![v \rightsquigarrow z]\!],$$

where it is understood that $[x \prec_v z] := 0$ in the degenerate cases where $v \in \{x, z\}$. Similar decompositions can be written for $z \rightsquigarrow y$ and $y \rightsquigarrow x$. Indeed, it is easily checked that

$$[\![x \rightsquigarrow z]\!] + [\![z \rightsquigarrow y]\!] + [\![y \rightsquigarrow x]\!] = [x \prec_v z] + [z \prec_v y] + [y \prec_v x].$$

By assumption, $[\![z \rightsquigarrow y]\!], [\![y \rightsquigarrow x]\!] \leq -1$, so we get

$$[\![x \rightsquigarrow z]\!] \geq 2 + [x \prec_v z] + [z \prec_v y] + [y \prec_v x].$$

The only way the right hand side can be negative is if

$$[x \prec_v z] = [z \prec_v y] = [y \prec_v x] = -1,$$

but this is equivalent to having $z \prec_v x \prec_v y \prec_v z$, which is impossible. Hence $[\![x \rightsquigarrow z]\!] \geq 0$. But since $x \neq z$ we must have in fact that $x \prec z$.