Fast Deterministic Algorithms for Highly-Dynamic Networks

Keren Censor-Hillel
Technion, Haifa, Israel
ckeren@cs.technion.ac.il

Neta Dafni
Technion, Haifa, Israel
netad@cs.technion.ac.il

Victor I. Kolobov
Technion, Haifa, Israel
tkolobov@cs.technion.ac.il

Ami Paz
Faculty of Computer Science, Universität Wien, Austria
ami.paz@univie.ac.at

Gregory Schwartzman
Japan Advanced Institute of Science and Technology, Ishikawa, Japan
greg@jaist.ac.jp

Abstract

This paper provides an algorithmic framework for obtaining fast distributed algorithms for a highly-dynamic setting, in which *arbitrarily many* edge changes may occur in each round. Our algorithm significantly improves upon prior work in its combination of (1) having an $O(1)$ amortized time complexity, (2) using only $O(\log n)$-bit messages, (3) not posing any restrictions on the dynamic behavior of the environment, (4) being deterministic, (5) having strong guarantees for intermediate solutions, and (6) being applicable for a wide family of tasks.

The tasks for which we deduce such an algorithm are maximal matching, $(degree + 1)$-coloring, 2-approximation for minimum weight vertex cover, and maximal independent set (which is the most subtle case). For some of these tasks, node insertions can also be among the allowed topology changes, and for some of them also abrupt node deletions.

2012 ACM Subject Classification Theory of computation

Keywords and phrases dynamic distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2020.28

Related Version A full version of the paper is available at `https://arxiv.org/abs/1901.04008` .

Funding *Keren Censor-Hillel*: This project has received funding from the European Union's Horizon 2020 Research And Innovation Program under grant agreement no.755839.
Ami Paz: We acknowledge the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N.
Gregory Schwartzman: This work was supported by JSPS Kakenhi Grant Number JP19K20216 and JP18H05291.

Acknowledgements The authors are indebted to Yannic Maus for invaluable discussions which helped us pinpoint the exact definition of fixing in dynamic networks that we eventually use. We also thank Juho Hirvonen for comments about an earlier draft of this work, and Shay Solomon for useful discussions about his work in [1, 2]. We thank Hagit Attiya and Michal Dory for their input about related work.

1 Introduction

We present a family of deterministic distributed algorithms that rapidly fix solutions for fundamental tasks even in a highly-dynamic environment. Specifically, we provide algorithms for maximal matching, (*degree* + 1)-coloring, 2-approximation for the minimum *weighted* vertex cover (2-MWVC), and maximal independent set (MIS). We further show that for some of these tasks, fast fixing is also possible with node insertions and deletions. Here, we consider the severe case of *abrupt* deletions, where a deleted node does not have a chance to inform its neighbors about its upcoming departure from the system.

Our algorithms enjoy the combination of (1) having an $O(1)$ amortized time complexity, (2) using only $O(\log n)$-bit messages, (3) not posing any restrictions on the dynamic behavior of the environment and in particular not requiring topology changes to be spaced in time, (4) being deterministic, (5) having strong guarantees for intermediate solutions, and (6) being applicable for a wide family of tasks. In recent years, there has been much progress on distributed dynamic algorithms, achieving different combinations of the above promises. Our algorithms significantly improve upon all prior work by that they guarantee the combination of all the above properties. We elaborate upon – and compare to – prior work in Section 1.4.

We stress that as opposed to centralized dynamic data structures, not posing any restrictions on the dynamic behavior of the environment is vital in the distributed setting, as the input graph is the communication graph itself. More concretely, in centralized dynamic data structures when multiple topology changes occur, we can simply handle them one by one. However, in our setting, nodes cannot communicate over a deleted edge, and so we cannot sequentially apply an independent update algorithm for each topology change – an edge deletion affects the communication already when it *happens*, not only when it is *handled*.

1.1 Motivation

Each of the aforementioned problems is a locally-checkable labeling (LCL) problem. The notion of an LCL is a celebrated concept in distributed computing, first defined by Naor and Stockmeyer [32] in order to capture tasks in which nodes can efficiently detect inconsistencies, motivated by the unstable nature of distributed systems. Since the publication of this pioneering work, the complexity of solving tasks that can be described as LCLs has been extensively studied in the distributed setting. We ask the following question, paraphrased in correspondence with the title of [32]:

Question: What can be fixed locally?

We begin by recalling the definition of LCLs of [32], restricting our attention to LCLs with *radius* $r = 1$. A *centered star* is a pair (H, s) where H is a star graph and s is its center. An LCL $\mathcal{L}$ is a tuple $(\Sigma, \Gamma, \mathcal{C})$, where Σ is a set of *input labels*, Γ is a set of *output labels*, and $\mathcal{C}$ is a set of *locally consistent labelings*. Each element of $\mathcal{C}$ is a centered star, with a label in $\Sigma \times \Gamma$ for each of its nodes.[1]

A labeling $\lambda : V \to \Sigma \times \Gamma$ is called $\mathcal{L}$-legal for a graph $G = (V, E)$, if for every $v \in V$, there exists a centered star (H, s) in $\mathcal{C}$ with a label-pair at each node, which is consistent with λ in the following sense: there exists a mapping π that maps the star centered at v in G into (H, s), with $\pi(v) = s$, such that for every node w in the star centered at v, the label-pair given by λ is the same as the label-pair of the node $\pi(w)$ in (H, s).

[1] In the work of Naor and Stockmeyer [32] the set of labels Σ has a fixed size, while here we omit this limitation in order to give more power to the labelings. However, algorithmically, we always keep the size of messages small even when labels are large, by sending only pieces of them.

As explained in [32], the set $\mathcal{C}$ defines allowed labels for neighborhoods, as opposed to defining a set of forbidden ones. If the LCL has no inputs, then one can simply choose a default input label, i.e., $|\Sigma| = 1$. An algorithm that solves the problem defined by an LCL $\mathcal{L}$ is an algorithm whose output on a graph G is an $\mathcal{L}$-legal labeling.

Not all LCLs are easily fixable. The following variant of the *sinkless orientation* problem [13] is an example of an LCL problem that is not easily fixable. Each node has a label that corresponds to an orientation of its edges, such that labels at endpoints of an edge are consistent, and such that there is no node of degree greater than 1 that is a sink, i.e., has no outgoing edge. It is easy to verify that every graph has a valid labeling[2], and that this is an LCL. To see that this LCL cannot be fixed within an amortized complexity of $O(1)$, consider a graph on n nodes that evolves dynamically, creating two paths of roughly $n/2$ nodes each. Each path must be oriented consistently with a single sink in one of its endpoints. Inserting an edge between the sinks of the two paths forces the orientation of all of the edges in one of the sub-paths to flip, which takes $\Omega(n)$ rounds. Deleting this edge induces again two paths with a single sink each, and repeating the process of inserting an edge between the new sinks and deleting it causes a linear number of rounds that can be attributed to only two topology changes, which implies an amortized time of $\Omega(n)$. This holds even if topology changes do not happen concurrently, and even if the messages can be of arbitrarily large size.

1.2 The challenges

For any LCL problem we address, we assume that the system begins with a globally correct labeling, and thus what an algorithm needs to do as a consequence of topology changes is to have the affected nodes update their labels. Naturally, for some problems, the update procedure may also require that a node updates the labels of its neighbors (more precisely, this is accomplished via sending messages to its neighbors requiring them to update their labels). For example, in a solution for maximal matching this might occur when an edge that is in the matching is deleted, and its endpoints need to match themselves to other neighbors. At a first glance, this may sound as a simple and straightforward approach for fixing matchings and problems of local flavor. However, this approach turns out to be far from trivial, and below we describe multiple key challenges that we must overcome in order to implement it successfully.

(1) Defining *fixing* and *amortized complexity*. We need to define what *fixing the solution* means. We aim for our algorithm to work in a very harsh setting, in which it might be the case that there are so many topology changes that we never actually obtain a globally correct labeling, but still we maintain strong guarantees for intermediate labelings. Notice that this is in stark contrast to centralized dynamic data structures, which can always consider globally correct solutions since topology changes may be handled one-at-a-time because they only affect the input and not the computation itself. This is also the case for the majority of previous distributed algorithms: they are designed under the assumption that topology changes are spaced well enough in time so that it is possible to obtain a globally correct solution before the next topology change happens.

[2] If G is a tree, choose an arbitrary root and orient the edges away from the root. Otherwise, choose a cycle in G and orient its edges cyclically, then imagine contracting its nodes into a single super-node and orient edges towards this super-node along some spanning tree, and orient other edges arbitrarily.

(2) Coping with concurrent fixing with a timestamp mechanism. Because we might need a node to change the labels of its neighbors and not only its own label in order to fix the solution, we make sure that concurrent fixing always happens for nodes that are not too close, and other nodes wait even if their labeled stars are not yet correct (e.g., to avoid two nodes u, v, trying to get matched to the same node w concurrently). To this end, our method is to assign a timestamp to each node involved in a change, and *fix* a node only if its timestamp is a local minimum in some short-radius neighborhood, thus avoiding conflicting concurrent fixes. We call such a node *active*.

(3) Detecting and aborting conflicting timestamps. Such a timestamp mechanism alone is still insufficient: the uncontrolled number of topology changes may, for example, suddenly connect two nodes that were previously far enough so that they could become active simultaneously, but after concluding that they can both become active, an edge insertion now makes them part of the same short-radius neighborhood. We carefully take care of such cases where our timestamps have been cheated by the topology changes, by detecting such occurrences and *aborting the fixing*, without harming the amortized complexity guarantees.

(4) Bounding the size of timestamps to cope with message size restrictions. Finally, the restriction on the size of messages forbids unbounded timestamps, despite an unbounded number of rounds (e.g., times). To resolve this issue, we utilize ideas from the literature on shared memory algorithms, e.g., [3], for deterministically hashing the timestamps into a small bounded domain so that the nodes can afford sending a hashed timestamp in a single small message, and we do so in a way that preserves the total order over timestamps.

1.3 Our contributions

Our main contribution is thus deterministic dynamic distributed fixing algorithms for several fundamental problems. Our algorithms share a common approach, and only minor modifications that are specific to each labeling are required. In some cases we can also handle a node insertion/deletion, which is a-priori possibly harder to deal with, because it may affect more nodes while in the amortized analysis we count it as a single topology change.

The following theorem summarizes the end-results, which hold in a model with an unbounded number of topology changes that may occur concurrently, and when only a logarithmic number of bits can be sent in a message.

▶ **Theorem 1.** *There is a deterministic dynamic distributed fixing algorithm for* ***(degree+1)-coloring*** *and for a* ***2-approximation of a minimum weight vertex cover****, which handles edge insertions/deletions and node insertions in $O(1)$ amortized rounds.*

There are deterministic dynamic distributed fixing algorithms for ***maximal matching****,* ***$(\Delta + 1)$-coloring*** *(where Δ is the maximum node degree) and* ***MIS****, which handle edge/node insertions/deletions in $O(1)$ amortized rounds.*

Section 3 shows our algorithm for maximal matching. This is developed and modified in the full version of the paper to present our 2-MWVC algorithm. We mention that the labeling for the solution of 2-MWVC that we maintain is not the naïve one that only indicates which nodes are in the cover, but rather contains information about dual variables that correspond to edge weights, and allow the fast fixing.

Section 4 gives our algorithm for MIS. In the MIS case, the restriction of message size imposes an additional, huge difficulty. The reason is that if an MIS node v needs to leave the MIS because an edge is inserted between v and some other MIS node u, then all other neighbors of v who were previously not in the MIS are now possibly not covered by an MIS neighbor. Yet, they cannot all be moved into the MIS, as they may have an arbitrary topology among them. With unbounded messages this can be handled using very large neighborhood information but such an approach is ruled out by the the restriction of $O(\log n)$-bit messages.

Nevertheless, we prove that with some modifications to our algorithmic approach, we can also handle MIS without the need to inform nodes about entire neighborhoods. The road we take here is that instead of fixing its neighborhood, a node tells its neighbors that they should become active themselves in order to fix their labeled stars. On the surface, this would entail an unacceptable overhead for the amortized complexity that is proportional to the degree of the node. The crux in our algorithm and analysis is in blaming previous topology changes for such a situation – for every node u in the neighborhood of v which is only dominated by v, there is a previous topology change (namely, an insertion of an edge $\{u, w\}$, where w may or may not be v) for which we did not need to fix the label of w. This accounting argument allows us to amortize the round complexity all the way down to $O(1)$, and the same technique is utilized to handle node insertions and deletions. In the full version of the paper, we present our algorithm for $(\mathit{degree} + 1)$-coloring, as well as a generalization of our algorithm, by defining a family of graph labelings, in the flavor of the LCL definition, which can all be fixed in constant amortized time.

1.4 Related work

The end results of our work provide fast fixing for fundamental graph problems, whose static algorithmic complexity has been extensively studied in the distributed setting. A full overview of the known results merits an entire survey paper on its own (see, e.g., [8,35]). An additional line of beautiful work studies the landscape of distributed complexities of LCL problems, and the fundamental question of using randomness (see, e.g., [5,6,14,17,18,23]).

For dynamic distributed computing, there is a rich history of research on the important paradigm of *self-stabilization* (see, e.g., the book [20]) and in particular on symmetry breaking (see, e.g., the survey [24]). Related notions of error confinement and fault-local mending have been studied in [4,30,31]. Our model greatly differs from the above. There are many additional models of dynamic distributed computation (e.g., [12,29]), which are very different from the one we consider in this paper.

Some of the oldest works in similar models to ours are [22,26], who provide algorithms for distance-related tasks. Constant-time algorithms were given in [28] for symmetry-breaking problems assuming unlimited bandwidth and a single topology change at a time. The work of [15], provides a randomized algorithm that uses small messages to fix an MIS in $O(1)$-amortized update time for a non-adaptive oblivious adversary, still assuming a single change at a time. The latter left as an open question the complexity of fixing an MIS in the sequential dynamic setting. This was picked up in [1,2,21,25], giving the first non-trivial sequential MIS algorithms, which were recently revised and improved [10,19]. Specifically, the algorithm of [1] achieves an $O(\min\{\Delta, m^{3/4}\})$ amortized message complexity and $O(1)$-amortized round complexity and adjustment complexity (the number of vertices that change their output after each update) for an adaptive non-oblivious adversary in the distributed setting. However, they handle only a single change at a time, and sometimes need to know the number of edges, which is global knowledge that our work avoids assuming. In fact, if one is happy with restricting the algorithm to work only in a model with a single topology change at a time, then sending timestamps is not required, so $O(1)$-bit messages suffice in our algorithm for MIS, resembling what [1] obtains.

[33] provides a neat log-starization technique, which translates logarithmic static distributed algorithms into a dynamic setting such that their amortized time complexity becomes $O(\log^* n)$. This assumes a single change at a time and large messages. [34] shows that maximal matching have $O(1)$ amortized complexity, even when counting messages and not only rounds, but assuming a single change at a time.

The $(\Delta + 1)$-coloring algorithm of [9] also implies fixing in a self-stabilizing manner – after the topology stops changing, only $O(\Delta + \log^* n)$ rounds are required in order to obtain a valid coloring, where Δ bounds the degrees of all the nodes at all times.

Perhaps the setting most relevant to ours is the one studied in [7], who also address a very similar highly-dynamic setting. They insightfully provide fast dynamic algorithms for a wide family of tasks, which can be decomposed into packing and covering problems, in the sense that a packing condition remains true when deleting edges and a covering condition remains true when inserting edges. For example, MIS is such a problem, with independence and domination being the packing and covering conditions, respectively. An innovative contribution of their algorithms is providing guarantees also for intermediate states of the algorithm, that is, guarantees that hold even while the system is in the fixing process. They show that the packing property holds for the set of edges that are present throughout the last T rounds, and that the covering property holds for the set of edges that are present in either of the last T rounds, for $T = O(\log n)$. Moreover, their algorithms have correct solutions if a constant neighborhood of a node does not change for a logarithmic number of rounds. Our algorithm guarantees correctness of labeled stars for nodes for which any topology change touching their neighborhood has already been handled. In comparison with their worst-case guarantee of $O(\log n)$ rounds for a correct solution, our algorithm only gives $O(n)$ rounds in the worst case. However, our amortized complexity is $O(1)$, our messages are of logarithmic size, and our algorithm is deterministic, while the above is randomized with messages that can be of polylogarithmic size. In addition, a recent work [16] studies subgraph problems in the same model described in our paper.

A different definition of local fixability [11, Appendix A], suitable for *sequential* dynamic data structures, requires a node to be able to fix the solution by changing only its own state. While this captures tasks such as coloring, and is helpful in the sequential setting for avoiding the need to update the state of all neighbors of a node, in the distributed setting we can settle for a less restrictive definition, as a single communication round suffices for updating states of neighbors, if needed. Our algorithmic framework captures a larger set of tasks: notably, we provide an algorithm for MIS, while [11] prove that it does not fall into their definition. In addition, [11, Section 7] raises the question of fixing (in the sequential setting) problems that are in P-SLOCAL[3] [23]. Notably, this class contains approximation tasks, and indeed for some approximation ratios we can apply our framework: Our algorithm has the flavor of sequentially iterating over nodes and fixing the labels in their neighborhood, with the additional power of the distributed setting that allows it to work concurrently on nodes that are not too close. This also resembles the definition of orderless local algorithms [27], although a formal definition for the case of fixing does not seem to be simpler than ours.

[3] Roughly speaking, SLOCAL(t) is the class of problems that admit solutions by an algorithm that iterates over all the nodes of the graph, and assigns a solution to each node based on the structure of its t-neighborhood and solutions already assigned to nodes in this neighborhood. P-SLOCAL is the class SLOCAL(polylog n).

2 Model

We assume a synchronous network that starts as an empty graph on n nodes and evolves into the graph $G_i = (V_i, E_i)$ at the beginning of round i; in most of our algorithms, one can alternatively assume any graph as the initial graph, as long as the nodes start with a labeling that is globally consistent for the problem in hand. In some cases, we also allow node insertion or deletion, and then n serves as a universal upper bound on the number of nodes in the system. Each node is aware of its unique id, the edges it is a part of, its weight if there is one, and of n. In addition, the nodes have a common notion of time, so the execution is synchronous. New nodes do not know the global round number. (We mention that in our algorithms it is sufficient for each node to know the round number modulo $15n$, and a new node can easily obtain this value from its neighbors, so we implicitly assume all nodes have this knowledge.)

In each round, each node receives *indications* about the topology changes that occurred to its incident edges. We stress that the indications are a posteriori, i.e., the nodes get them only after the changes occur, and thus cannot prepare to them in advance (these are called *abrupt* changes). After receiving the indications and performing local computation, each node can send messages of $O(\log n)$ bits to each of its neighbors.

We work in a distributed setting where each node stores its own label. A distributed fixing algorithm should update the labels of the nodes in a way that corrects the labeled stars that become incorrect due to topology changes. Naturally, for a highly-dynamic setting, we do not require a global consistent labeling in scenarios in which the system is undergoing many topology changes.

We consider four classical graph problems. In the *maximal matching* problem, the nodes have to mark a set of edges such that no two intersect, and such that no edge can be added to the set without violating this condition. In *minimum weight vertex cover* (MWVC), the nodes start with weights, and the goal is to choose a set of nodes that intersect all the edges, and have the minimum weight among all such sets; we will be interested in the 2-approximation variant of the problem, where the nodes choose a set of weight at most twice the minimum. Finally, in the *maximal independent set* (MIS) problem, the nodes must mark a set of nodes such that no two adjacent nodes are chosen, and such that no node can be added to the set.

The complexity of distributed fixing algorithms. When the labels of a star become inconsistent due to changes, a distributed fixing algorithm will perform a fixing process, which ends when the labels are consistent again, or when other changes occur in this star. The *worst-case round complexity* of a distributed fixing algorithm is the maximum number of rounds such a fixing process may take.

In our algorithms, it could be that it takes a while to fix some star, but we can argue that this is because other stars are being fixed. We measure this progress with a definition of the amortized round complexity.

When studying *centralized* algorithms for dynamic graphs, the amortized complexity measure is typically defined by an *aggregate analysis*, i.e., considering the time when the fixing process ends, and dividing the number of computation steps taken so far by the number of changes that occurred. The natural generalization of this definition to the distributed setting could be to take a time when the graph labeling is globally correct, and divide the number of rounds occurred so far by the number of changes the network had undergone. The first and most eminent problem in such a definition is that it requires a time when the *global solution* is correct, which is something that we cannot demand in a highly-dynamic environment. The second problem with it is that the adversary can fool this complexity measure, by doing nothing for some arbitrary number of rounds in which the graph is correct, while the algorithm still gets charged for these rounds.

To overcome the above problems, we define the amortized round complexity as follows. Starting from round 0, in which the labeling is consistent for all stars, we consider the situation in each round i. We denote by $\texttt{incorrect}(i)$ the number of rounds until round i in which there exists at least one inconsistent star. These are the computation rounds for which we charge the algorithm. Notice that we do not count only communication rounds in order to prevent an algorithm that cheats by doing nothing.[4] We denote by $\texttt{changes}(i)$ the number of changes which occurred until round i. We say that an algorithm has an *amortized round complexity* k if for every i with $\texttt{changes}(i) > 0$, we have $\texttt{incorrect}(i)/\,\texttt{changes}(i) \leq k$. This definition captures the *rate* at which changes are handled, in a way that generalizes the sequential definition.

Guarantees of our algorithm. Our algorithms have an $O(1)$ amortized fixing time, and in addition, they have additional desired progress properties. First, our algorithms guarantee a worst-case complexity of $O(n)$, which implies that repeated changes far from a given star will not postpone it from being fixed for too long. Moreover, if a labeled star is consistent and no topology change touches its neighborhood, then it remains consistent. Thus, our algorithm has strong guarantees also for intermediate solutions.

3 An $O(1)$ amortized dynamic algorithm for maximal matching

The solution to the maximal matching problem at any given time is determined according to the labels of the nodes. A label of a node v can be either `unmatched` or `matched-to-`u, indicating that v is unmatched, or is matched to u, respectively. Each node starts with the label `unmatched`. Alternatively, one can assume any graph as the initial graph, as long as the nodes start with a legal maximal matching solution. We prove the following.

▶ **Theorem 2.** *There is a deterministic dynamic distributed fixing algorithm for maximal matching which handles edge insertions/deletions in $O(1)$ amortized rounds.*

Proof. First, we assume that all nodes start with an initial globally consistent solution.

The setup: We denote $\gamma = 5$.

Let F_i be a set of edge changes (insertions/deletions) that occur in round $i \geq 0$ (for convenience, the first round is round 0). With each change in F_i, we associate two *timestamps* such that a total order is induced over the timestamps as follows: for an edge $e = \{u, v\}$ in F_i, we associate the timestamp $ts = (i, u, v)$ with node u, and the timestamp (i, v, u) with node v. Since u and v start round i with an indication of e being in F_i, both can deduce their timestamps at the beginning of round i. We say that a node v is the *owner* of the timestamps that are associated with it. In each round, a node only stores the largest timestamp that it owns, and omits the rest.

Notice that timestamps are of unbounded size, which renders them impossible to fit in a single message. To overcome this issue we borrow a technique of [3], and we invoke a deterministic hash function H over the timestamps, which reduces their size to $O(\log n)$

[4] One could count also rounds in which the labeling is globally correct if the algorithm chooses to communicate in these rounds. Our algorithm never communicates in such rounds, so such a definition would not change our amortized complexity.

bits, while retaining the total order over timestamps. The reason we can do this is that not every two timestamps can exist in the system concurrently. To this end, we define $h(i) = i \bmod 3\gamma n$ and $H(ts) = (h(i), u, v)$ for a timestamp $ts = (i, u, v)$, and we define an order $\prec_H$ over hashed timestamps as the lexicographic order of the 3-tuple, induced by the following order $\prec_h$ over values of h. We say that $h(i) \prec_h h(i')$ if and only if one of the following holds:

- $0 \le h(i) < h(i') \le 2\gamma n$, or
- $\gamma n \le h(i) < h(i') < 3\gamma n$, or
- $2\gamma n \le h(i) < 3\gamma n$ and $0 \le h(i') < \gamma n$.

If two timestamps $ts = (i, v, u)$, $ts' = (i', v', u')$ are stored in two nodes v, v' at two times i, i', respectively, it holds that $ts < ts'$ (by the standard lexicographic order) if and only if $H(ts) \prec_H H(ts')$. The reason that this holds despite the wrap-around of hashed timestamps in the third bullet above, is the following property that we will later prove: for every two such timestamps, it holds that $i' - i \le \gamma n$. This implies $h(i) \prec_h h(i')$ whenever $i < i'$ despite the bounded range of the function h.

The algorithm: In the algorithm, time is chopped up into *epochs*, each consisting of γ consecutive rounds, in a non-overlapping manner. That is, epoch j consists of rounds $i = \gamma j, \ldots, \gamma(j+1) - 1$. For every epoch $j \ge 0$, we consider a set $D_j \subseteq V$ of *dirty* nodes at the beginning of each epoch, where initially no node is dirty ($D_0 = \emptyset$). Some nodes in D_j may become *clean* by the end of the epoch, so at the end of the epoch the set of dirty nodes is denoted by D'_j, and it holds that $D'_j \subseteq D_j$. At the beginning of epoch $j + 1$, all nodes that receive any indication of an edge in F_i in the previous epoch are added to the set of dirty nodes, i.e., $D_{j+1} = D'_j \cup I_j$, where I_j is the set of nodes that start round i with any indication about F_i, for any $\gamma j \le i \le \gamma(j+1) - 1$.

Intuitively, the algorithm changes the labels so that *the labels at the* ***end*** *of the epoch are consistent with respect to the topology that was at the* ***beginning*** *of the epoch*, unless they are labels of dirty nodes or of neighbors of dirty nodes.

The algorithm works as follows. In epoch $j = 0$, the nodes do not send any messages, but some of them enter I_0 (if they receive indications of edges in F_i, for $0 \le i \le \gamma - 1$).

Denote by N_v^i the neighborhood of v in round i, denote by L_v^i the label of v at the beginning of round i, before the communication takes place, and denote by $\hat{L}_v^i$ the label at the end of the round. Unless stated otherwise, the node v sets $\hat{L}_v^i \leftarrow L_v^i$ and $L_v^{i+1} \leftarrow \hat{L}_v^i$. Now, consider an epoch $j > 0$. On round γj every node $v \in D_j$ may locally change its label to indicate that it is unmatched, in case the edge between v and its previously matched neighbor is deleted:

$$L_v^{\gamma j} = \begin{cases} \texttt{matched-to-}u, & \text{if } \hat{L}_v^{\gamma j - 1} = \texttt{matched-to-}u \text{ and } u \in N_v^{\gamma j} \\ \texttt{unmatched}, & \text{otherwise} \end{cases} \tag{1}$$

where $\hat{L}_v^{\gamma j-1}$ is the label that v has at the *end* of round $\gamma j - 1 = \gamma(j-1) + 4$, which, as we describe below, may be different from its label $L_v^{\gamma j - 1}$ at the beginning of the round.[5] Then, the node v sends $L_v^{\gamma j}$ to its neighbors. These are the labels for the graph $G_{\gamma j}$ which the fixing addresses. We stress that the new labels $L_v^{\gamma j}$ might not form consistent stars. Instead, the nodes update $L_v^{\gamma j}$ and send it to all neighbors in order to maintain a common graph, with respect to which we show local consistency. As an example, consider a triangle w, v, u,

[5] We stress that one can describe our algorithm with labels that can only change at the beginning of a round, but we find the exposition clearer this way.

undergoing the deletion of the edge $\{w, v\}$ and of another edge connecting u with some other node (see Figure 1). Suppose that w immediately tries to fix the labels in its star, according to the fact that the edge $\{w, v\}$ does not exist, while u is selected to fix its own star before v, without knowing of the deletion of the edge $\{w, v\}$. Both nodes then simultaneously try to change the label of u, and it could not be clear what u should do, and which neighborhood of u will be corrected.

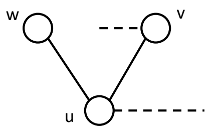

Figure 1 Dashed lines represent edges that were deleted.

We continue describing the algorithm. On rounds $\gamma j + 1$ to $\gamma j + 3$ the nodes propagate the hashed timestamps owned by dirty nodes. That is, on round $\gamma j + 1$, each node in D_j broadcasts its hashed timestamp, and on the following two rounds all nodes broadcast the smallest hashed timestamp that they see (with respect to the order $\prec_H$). Every node v in D_j which does not receive a hashed timestamp that is smaller than its own becomes *active*.

On the last round of the epoch, $\gamma j + 4$, every active node v computes the following candidate for a new label, denoting by $N_v^{\gamma j} = \{u_1, \ldots, u_d\}$ the neighborhood it had at round γj.

$$\ell_v = \begin{cases} \texttt{matched-to-}u_i, & \text{if } L_v^{\gamma j} = \texttt{matched-to-}u_i \\ \texttt{unmatched}, & \text{if } L_v^{\gamma j} = \texttt{unmatched} \text{ and for every } 1 \le i \le d, L_{u_i}^{\gamma j} \ne \texttt{unmatched} \\ \texttt{matched-to-}u_i, & \text{if } L_v^{\gamma j} = \texttt{unmatched} \text{ and } 1 \le i \le d \text{ is the smallest index} \\ & \text{for which } L_{u_i}^{\gamma j} = \texttt{unmatched} \end{cases} \tag{2}$$

Notice that v has the required information to compute the above, even if additional topology changes occur during the rounds in which timestamps are propagated. Yet, we need to cope with the fact that topology changes may occur also throughout the current epoch and, for example, make active nodes suddenly become too close. For this, we denote by $T_j \subseteq I_j$ the set of *tainted* nodes who received an indication of a topological change for at least one of their edges during the epoch j.

Now, only an active node v which is not in T_j sets $\hat{L}_v^{\gamma j+4} \leftarrow \ell_v$ and sends this new label to each neighbor u. Otherwise, an active node v that is tainted (i.e., is in T_j) aborts and remains dirty for the next epoch. Of course, if nodes u and v are neighbors at the beginning of an epoch but not when v sends the computed label, then u does not receive this information.

Finally, every active node $v \notin T_j$, if $\hat{L}_v^{\gamma j+4} = \texttt{matched-to-}u$ then u updates $\hat{L}_u^{\gamma j+4} = \texttt{matched-to-}v$ (note that such u has the required information since it receives ℓ_v, as otherwise, if by the time that ℓ_v is computed it holds that u and v are no longer neighbors, then v must be tainted). At the end of round $\gamma j + 4 = \gamma(j + 1) - 1$, node v becomes *inactive* and, unless it aborts, is not included in D'_j, i.e., we initialize $D'_j = D_j \setminus \{v \mid v \notin T_j \text{ is active in epoch } j\}$ at the end of epoch j.

Correctness. For correctness we claim the following invariant holds at the end of round $i = \gamma j + 4 = \gamma(j + 1) - 1$: For every two nodes u, v that are clean at the end of the epoch and for which $\{u, v\}$ is an edge in $G_{\gamma j}$, it holds that (1) at least one of $\hat{L}_u^{\gamma j+4}$ and $\hat{L}_v^{\gamma j+4}$ is not `unmatched` and (2) if $\hat{L}_u^{\gamma j+4} = \texttt{matched-to-}v$ then $\hat{L}_v^{\gamma j+4} = \texttt{matched-to-}u$.

We prove the above by induction on the epochs. The base case holds trivially as during the first epoch the labels do not change, and we assume that the nodes start with a legal maximal matching for the initial graph. Now, assume the above invariants hold for epoch $j-1$.

For every two nodes u, v that are clean at the end of the epoch and for which $\{u, v\}$ is an edge in $G_{\gamma j}$, if their labels do not change during the epoch, then the invariant follows from the induction hypothesis.

If only one of their labels changes, say that of v, then either v is active and not tainted or there is a (single) neighbor w of v which is active and not tainted and makes v change its label. In the former case, since the label ℓ_v of v changes compared to $L_v^{\gamma j}$, it does not remain `unmatched` and does not remain `matched-to-`x for some node x. So the new label ℓ_v must be `matched-to-`y, for some node y. Since the label of u does not change, we have that $u \neq y$, and so if the label of u is not `unmatched` then it cannot be `matched-to-`v (as otherwise $L_v^{\gamma j}$ would be `matched-to-`u and so ℓ_v would also be `matched-to-`u, thus did not change). In the latter case, if v changes its label because of the new label ℓ_w that is sent to it by a neighbor w, then $\ell_w =$ `matched-to-`v and hence the new label of v is set to `matched-to-`w.

Finally, if both of their labels change, then without loss of generality v is active and not tainted and computes $\ell_v =$ `matched-to-`u, making u update its label to `matched-to-`v. The crucial thing to notice here is that it cannot be the case that a node w_v changes the label of v and a different node w_u changes the label of u at the same time, because this implies that the distance between w_v and w_u is at most 3, in which case either at least one of them aborts due to an edge insertion, or the edge $\{u, v\}$ is inserted (maybe immediately after being deleted), but then v and u are not clean.

Since the invariant holds, we conclude that whenever $D_j = \emptyset$, it holds that the labeling is that of a maximal matching for $G_{\gamma j}$. Further, what the invariant implies is that some correctness condition holds even for intermediate rounds: at the end of every epoch j, the entire subgraph induced by the set of nodes that are clean and have all of their neighborhood clean consists of nodes with locally consistent labels.

Round complexity. We now prove that the algorithm has an amortized round complexity of $O(1)$, by proving `incorrect`$(i) \leq 2\gamma \cdot$ `changes`(i) for all i. First, note that the algorithm communicates in each round where the graph is incorrect, and these communication rounds can be split into epochs, implying `incorrect`$(i) \leq \gamma \cdot$ `epochs`(i), where `epochs`(i) denotes the number of epochs of computation done by the algorithm until round i (if round i is the middle of an epoch then it does not affect the asymptotic behavior, so we can safely ignore this partial epoch). On the other hand, the node with minimal timestamp at the beginning of the j-th epoch becomes active during the epoch, and its timestamp is handled – even if it becomes tainted by a change, the old timestamp is replaced by the new one. So, in each epoch at least one timestamp disappears from the system. Now, since each topology change creates at most two timestamps, we have that the number of timestamps created until round i is at most $2 \cdot$ `changes`(i), implying `epochs`$(i) \leq 2 \cdot$ `changes`(i), and the claim follows.

Finally, we show that the timestamps can be represented by $O(\log n)$ bits. First, we claim that for every two timestamps $ts = (i, v, u)$ and $ts' = (i', v', u')$ such that $ts < ts'$, that are simultaneously owned by nodes at a given time, it holds that $i' - i \leq \gamma n$. Assume otherwise, and consider the first time when this condition is violated by a timestamp ts', with respect to a previous timestamp $ts < ts'$. This means that the owner v of ts does not become active for more than n epochs. Since up to this point in time there were no violations, in each epoch at least one timestamp was handled, and this was done in the desired order, i.e., all these labels where smaller than ts. So, v not becoming active for more than n epochs can only happen if at round i there were more than n timestamps which were then not yet handled, stored in various nodes. But there are at most n nodes and each one stores at most one timestamp so the above is impossible. Since $i' - i \leq \gamma n$, we have that $H(ts) \prec_H H(ts')$, because $h(i) \prec_h h(i')$, as argued earlier.

Using the above we can also see that the worst case running time of our algorithm is $O(n)$. To see this, fix some node v with an inconsistent star which does not experience topology changes touching its 1-hop neighborhood for $(\gamma + 1)n$ rounds. This guarantees that its timestamp does not change throughout these rounds, and after γn rounds its timestamp must become a local minima. In the following epoch, if no changes occur within its 1-hop neighborhood then its star becomes consistent, which matches the definition of having a worst-case complexity of $O(n)$. Further, once a node v successfully invokes a fixing of its star, the star remains consistently labeled as long as no topology changes touch the 1-hop neighborhood of v, thus we obtain strong guarantees for intermediate solutions. ◀

For node insertions and deletions, a direct application of the algorithm of Theorem 2 increases the amortized complexity if all neighbors of a changed node (inserted or deleted) become dirty and $O(\Delta)$ timestamps are associated with this topology change. However, notice that when an edge is inserted, it suffices that *only one* of its endpoints becomes dirty in the algorithm and gets matched to the other endpoint if needed. Hence, if a node is inserted, it suffices that the inserted node becomes dirty, and we do not need all of its neighbors to become so. An only slightly more subtle rule for deciding which nodes become dirty upon a node deletion gives the following.

▶ **Theorem 3.** *There is a deterministic dynamic distributed fixing algorithm for maximal matching which handles edge/node insertions/deletions in $O(1)$ amortized rounds.*

Proof. We modify the algorithm of Theorem 2 as follows. Upon an insertion of a node v, the node v becomes dirty. Upon a deletion of a node v with neighbors $\{u_1, \ldots, u_d\}$, only the node u_i, for $1 \leq i \leq d$, that is matched to v (if there exists such a node) becomes dirty.

The $O(1)$ amortized round complexity remains, as every topology change induces at most two new timestamps. Correctness still holds because it is not affected by a node insertion, which can be viewed as multiple edge insertions (in terms of correctness, but without paying this cost for the amortized time complexity), and it is not affected by a node deletion because for any other node $u_j \in \{u_1, \ldots, u_d\}$ such that $j \neq i$ it holds that the deletion of v does not influence its local consistency. ◀

4 An $O(1)$ amortized dynamic algorithm for MIS

One can use a similar approach in order to obtain an MIS algorithm. However, when a node v needs to be removed from the MIS due to an edge insertion, a neighbor u of v may need to join the MIS if none its other neighbors are in the MIS. One way to do this is to mark all of the neighbors of v as dirty, but this violates the amortized time complexity as a single topology change may incur too many dirty nodes. Another option is to have v's label include neighborhood information such that upon receiving this label, its neighbors know which of them should be moved into the MIS. This would give a simple $O(1)$ amortized rounds algorithm for MIS, but only if messages are allowed to be large. Instead, we present a labeling that does not contain all the neighborhood information and uses only *small* labels.

To handle the new subset of neighbors that needs to be added to the MIS in the aforementioned example, our approach is to have the active node simply indicate to all of its neighbors that they cannot remain clean and must check for themselves whether they need to change their labels. Of course, such a single topology change may now incur a number of dirty nodes that is the degree of this endpoint, which may be linear in n.

Yet, we make a crucial observation here: any node that becomes dirty in this manner, can be blamed on a previous topology change in which only one node becomes dirty. This implies a budget, to which we add 2 units for every topology change, and charge either 0, 1, 2, or d (current node degree) units for each invocation of the fixing function, in a manner that preserves the budget non-negative at all times. Note that due to the accounting argument, here we must start with an empty graph for the amortization to work, unlike previous problems, where we could start with any graph as long as the nodes have labels that indicate a valid solution. Roughly speaking, we rely on the fact that since all nodes that are in the graph start as MIS nodes because there are no edges, then a node switches from being an MIS node to being a non-MIS node only upon an insertion of an edge between two MIS nodes, and the other endpoint of the inserted edge safely remains in the MIS. Note that we impose the rule that an inserted node never makes a node switch from being an MIS node to being a non-MIS node, since the inserted node chooses to become an MIS node only if all of its neighbors are already non-MIS nodes.

▶ **Theorem 4.** *There is a deterministic dynamic distributed fixing algorithm for MIS, which handles edge/node insertions/deletions in $O(1)$ amortized rounds.*

Proof. We consider labels which are in $\{\texttt{true}, \texttt{false}\}$ and maintain that the set of nodes with the label `true` form an MIS. We start with an empty graph and all labels are `true`. We first define the assignments of L_v^i and ℓ_v as in assignments (1) and (2) in the algorithm for maximal matching in the proof of Theorem 2. Then, we explain how we modify the algorithm further in order to avoid large messages with neighborhood information.

First, the label for $L_v^{\gamma j}$ does not change from the previous round, i.e., assignment (1) is $L_v^{\gamma j} = \hat{L}_v^{\gamma(j-1)+4}$. For assignment (2) we set ℓ_v to be `false` if $L_{u_i}^{\gamma j} = \texttt{true}$ for some $u_i \in N_v^{\gamma j}$ and otherwise we set ℓ_v to be `true`. If v is active and not in T_j then it sets $\hat{L}_v^{\gamma j+4}$ to be ℓ_v and sends this label to all of its neighbors. Notice that this is insufficient for arguing that the labels at the end of the epoch form an MIS if all nodes are clean, for the same reason as in the tricky example above: if v leaves the MIS due to an edge insertion, its neighbors do not have enough information to decide which of them joins the MIS. To overcome this challenge, we consider an algorithm similar to the one of Theorem 2, with the following modifications.

(1) When an edge $e = \{v, u\}$ is deleted, if the labels of both u and v are `false` then neither of them becomes dirty, and if only one of them is `false` then only this node becomes dirty.
(2) When an edge $e = \{v, u\}$ is inserted, if at least one of the labels of u and v is `false` then neither of them becomes dirty, and if both are `true` then only the node with smaller ID becomes dirty.
(3) When a node v is inserted then only v becomes dirty.
(4) When a node v is deleted then a neighbor z becomes dirty only if its label is `false` and it has no neighbor with a label `true`.

In order for a node v to indicate that new labels may be needed for its neighbors, we add the following item:
(5) When an active node v changes its label to `false`, all of its neighbors marked `false` that do not have a neighbor marked `true` become dirty.

As we prove in what follows, this allows the correct fixing process that we aim for, but this has the cost of having too many nodes become dirty. However, the crucial point here is that not all nodes that become dirty in items (4) and (5) will actually utilize their timestamp

– some will drop their timestamp before competing for becoming active, and hence we will not need to account for fixing them. That is, we add the following item:

(6) When an active node v changes its label to `true`, all of its dirty neighbors become clean.

Correctness. The correctness follows the exact line of proof of the algorithm in Theorem 2, with the modification that making some neighbors dirty in item (5) compensates for not being able to assign them directly with good new labels. That is, at the end of the epoch, we still have the following guarantee: if all nodes are clean, then their labels induce an MIS; otherwise, for every two clean neighbors, either exactly one of them is in the MIS, or both have a neighbor in the MIS.

Amortized round complexity. The proof follows the same lines as the previous complexity proofs, with the addition of an accounting argument. This is used to prove that the cumulative number of epochs in which any node becomes active is at most twice the number of topology changes. This proves our claim of an amortized $O(1)$ round complexity.

First, as in the former algorithms, we note that $\texttt{incorrect}(i) \leq \gamma \cdot \texttt{epochs}(i)$, and at each epoch at least one timestamp is handled. Thus, we only need to upper bound the number of timestamps created by round i as a function of $\texttt{changes}(i)$. However, here we need to be much more careful and we can not simply account each change for two timestamps, as some changes create much more timestamps than others.

Consider a node v that is deleted in round i as in item (4) (or v is active and marked `false` as in item (5)), and a set $Z = \{z_1, \ldots, z_k\}$ of its neighbors that become dirty by satisfying the condition in item (4) (or item (5)) above, ordered by their timestamps (i, v, z_j) for $1 \leq j \leq k$, as induced by this topology change. For each $1 \leq j \leq k$, if a node z_j becomes active due to this timestamp, then by item (6), starting from round i none of its neighbors change their label to `true`. Consider the last round i' before round i in which the label of z_j is `true` (i' exists since this condition occurs initially when the graph is empty). We claim that the topology change whose associated active node changed the label of z_j to `false` in round $i' + 1$, is either an insertion of an edge $\{z_j, u\}$ that satisfies the condition of item (2) with $ID(z_j) < ID(u)$, or an insertion of the node z_j which connects it to at least one node whose label is `true`. The reason for this is that these are the only topology changes which cause z_j to be assigned the label `false`.

Finally, notice that these topology changes both induce only a single dirty node (thus a single active node and a single epoch), and therefore we can blame z_j becoming active on the corresponding topology change. This is an injective mapping, as any other node cannot blame these changes (they are changes that made z_j dirty), and z_j itself may become active again in the future due to satisfying the condition in item (4) (or item (5)) above only if its label is changed to `false` again in between.

In other words, this blaming argument implies $\texttt{epochs}(i) \leq 2 \cdot \texttt{changes}(i)$ here as well, completing the proof. ◀

References

1 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *STOC*, 2018.

2 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *SODA*, 2019.

3 Hagit Attiya, Danny Dolev, and Nir Shavit. Bounded polynomial randomized consensus. In *PODC*, 1989.

4 Yossi Azar, Shay Kutten, and Boaz Patt-Shamir. Distributed error confinement. *ACM Trans. Algorithms*, 2010.

5 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? *CoRR*, abs/1902.06803, 2019.

6 Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In *STOC*, 2018.

7 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. In *IPDPS*, 2019.

8 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

9 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed (Δ+1)-coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *PODC*, 2018.

10 Soheil Behnezhad, Mahsa Derakhshan, Mohammadtaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. *FOCS*, 2019.

11 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *SODA*, 2018.

12 Matthias Bonne and Keren Censor-Hillel. Distributed detection of cliques in dynamic networks. In *ICALP*, 2019.

13 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed lovász local lemma. In *STOC*, 2016.

14 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemyslaw Uznanski. LCL problems on grids. In *PODC*, 2017.

15 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *PODC*, 2016.

16 Keren Censor-Hillel, Victor I. Kolobov, and Gregory Schwartzman. Finding subgraphs in highly dynamic networks. In *submission*, 2020.

17 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM J. Comput.*, 2019.

18 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM J. Comput.*, 2019.

19 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. *FOCS*, 2019.

20 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

21 Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *CoRR*, abs/1804.08908, 2018. URL: `http://arxiv.org/abs/1804.08908`.

22 Michael Elkin. A near-optimal distributed fully dynamic algorithm for maintaining sparse spanners. In *PODC*, 2007.

23 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *STOC*, 2017.

24 Nabil Guellati and Hamamache Kheddouci. A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs. *J. Parallel Distrib. Comput.*, 2010.

25 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. `arXiv:1804.01823`.

26 Giuseppe F. Italiano. Distributed algorithms for updating shortest paths (extended abstract). In *WDAG*, 1991.

27 Ken-ichi Kawarabayashi and Gregory Schwartzman. Adapting local sequential algorithms to the distributed setting. In *DISC*, 2018.
28 Michael König and Roger Wattenhofer. On local fixing. In *OPODIS*, 2013.
29 Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *STOC*, 2010.
30 Shay Kutten and David Peleg. Fault-local distributed mending. *J. Algorithms*, 1999.
31 Shay Kutten and David Peleg. Tight fault locality. *SIAM J. Comput.*, 2000.
32 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 1995.
33 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *SODA*, 2016.
34 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, 2016.
35 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 2013.