Distributed Maximum Matching Verification in CONGEST

Mohamad Ahmadi
University of Freiburg, Germany
mahmadi@cs.uni-freiburg.de

Fabian Kuhn
University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Abstract

We study the maximum cardinality matching problem in a standard distributed setting, where the nodes V of a given n-node network graph $G = (V, E)$ communicate over the edges E in synchronous rounds. More specifically, we consider the distributed CONGEST model, where in each round, each node of G can send an $O(\log n)$-bit message to each of its neighbors. We show that for every graph G and a matching M of G, there is a randomized CONGEST algorithm to *verify* M being a maximum matching of G in time $O(|M|)$ and disprove it in time $O(D + \ell)$, where D is the diameter of G and ℓ is the length of a shortest augmenting path. We hope that our algorithm constitutes a significant step towards developing a CONGEST algorithm to *compute* a maximum matching in time $\tilde{O}(s^*)$, where s^* is the size of a maximum matching.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed matching, distributed graph algorithms, augmenting paths

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.37

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07649`.

1 Introduction and Related Work

A matching $M \subseteq E$ of a graph $G = (V, E)$ is a set of pairwise disjoint edges and the maximum matching problem asks for a matching M of maximum cardinality (or of maximum weight if the edges are weighted). Matchings have been at the center of attention in graph theory for more than a century (see, e.g., [38]). Algorithmic problems dealing with the computation of matchings are among the most extensively studied problems in algorithmic graph theory. The problem of finding a maximum matching is on the one hand simple enough so that it can be solved efficiently [16, 17], on the other hand the problem has a rich mathematical structure and led to many important insights in graph theory and theoretical computer science. Apart from work in the standard sequential setting, the problem has been studied in a variety of other settings and computational models. Exact or approximate algorithms have been developed in areas such as online algorithms (e.g., [18, 32]), streaming algorithms (e.g., [41]), sublinear-time algorithms (e.g., [40, 47]), classic parallel algorithms (e.g., [23, 31]), as well as also the recently popular massively parallel computation model (e.g., [5, 12, 25]). In this paper, we consider the problem of verifying whether a given matching is a maximum matching in a standard distributed setting, which we discuss in more detail next.

Distributed maximum matching. In the distributed context, the maximum matching problem is mostly studied for networks in the following synchronous message passing model. The network is modeled as an undirected n-node graph $G = (V, E)$, where each node hosts a distributed process and the processes communicate with each other over the edges of G. We assume in a distributed context that matching M is given as an input to or computed

as an output by G when every node of V knows its adjacent edges in M. As it is common practice, we identify the nodes with their processes and think of the nodes themselves as the distributed agents. Time is divided into synchronous rounds and in each round, each node $v \in V$ can perform some arbitrary internal computation, send a message to each of its neighbors in G, and receive the messages of the neighbors (round r is assumed to start at time $r-1$ and end at time r). If the messages can be of arbitrary size, this model is known as the LOCAL model [35, 44]. In the more realistic CONGEST model [44], in each round, each node can send an arbitrary $O(\log n)$-bit message to each of its neighbors.

Our contribution. As the main result of our paper, we give a distributed maximum matching verification algorithm.

▶ **Theorem 1.** *Given an undirected graph $G = (V, E)$ and a matching M of G, there is a randomized distributed* CONGEST *model algorithm to test whether M is a maximum matching. If M is a maximum matching, the algorithm verifies this in time $O(|M|)$, otherwise, the algorithm disproves it in time $O(D + \ell)$, where D is the diameter of G and ℓ is the length of a shortest augmenting path.*

Note that the size of a maximum matching is always $\Omega(D)$, so we are not trying to perform in sub-diameter time. Our main technical contribution is a distributed algorithm that, given a matching M and a parameter x, determines if there is an augmenting path of length at most x in $O(x)$ rounds of the CONGEST model. If there is an augmenting path of length at most x, the algorithm identifies two free (i.e., unmatched) nodes u and v between which such a path exists. We note that if the algorithm can be extended to also *construct* an augmenting path of length at most x between u and v in time $\tilde{O}(x)$, it would directly lead to an $\tilde{O}(s^*)$-round algorithm for computing a maximum matching, where s^* is the size of a maximum matching. The reason for this follows from the classic framework of Hopcroft and Karp [29]. It is well-known that if we are given a matching M of size $s^* - k$ for some integer $k \geq 1$, there is an augmenting path of length less than $2s^*/k$ [29]. Hence, if we can find such a path and augment along it in time linear in the length of the path, we get a total time of $O(s^* \log s^*)$ by summing over all values of k from 1 to s^*. This approach has been used in [2] to compute a maximum matching in time $O(s^* \log s^*)$ in bipartite graphs in the CONGEST model and it could also be employed without further difficulties for general graphs if we assume that we could actually construct the detected shortest augmenting paths. While finding a shortest augmenting path is quite straightforward in bipartite graphs, getting an efficient CONGEST model algorithm for general graphs turns out to be much more involved. We therefore hope that our algorithm for finding the length and the endpoints of some shortest augmenting path provides a significant step towards also efficiently constructing a shortest augmenting path in the CONGEST model and therefore to obtaining an $\tilde{O}(s^*)$-time CONGEST algorithm to find a maximum matching. Next we give a brief summary of the history of the distributed maximum matching problem.

Distributed maximal matching algorithms. While except for [2, 10], there is no previous work on exact solutions for the distributed maximum matching problem, there is a very extensive and rich literature on computing approximate solutions for the problem. The most basic way to approximate maximum matching is by computing a *maximal matching*, which provides a 1/2-approximation for the maximum matching problem. The work on distributed maximal matching algorithms started with the classic randomized parallel maximal matching and maximal independent set algorithms from the 1980s [3, 30, 39]. While these

algorithms were originally described for the PRAM setting, they directly lead to randomized $O(\log n)$-round algorithms in the CONGEST model. It was later shown by Hańćkowiak, Karoński, and Panconesi [26, 27] that maximal matching can also be solved deterministically in polylogarithmic time in the distributed setting. The current best deterministic algorithm in the CONGEST model (and also in the LOCAL model) is by Fischer [21] and it computes a maximal matching in $O(\log^2 \Delta \log n)$ rounds, where Δ is the maximum degree of the network graph G. At the cost of a higher dependency on Δ, the dependency on n can be reduced and it was shown by Panconesi and Rizzi [43] that a maximal matching can be computed in $O(\Delta + \log^* n)$ rounds. The best known randomized algorithm is by Barenboim et al. [9] and it shows that (by combining with the result of [21]) a maximal matching can be computed in $O(\log \Delta) + O(\log^3 \log n)$ rounds in the CONGEST model. The known bounds in the CONGEST model are close to optimal even when using large messages. It is known that there is no randomized $o\big(\frac{\log \Delta}{\log \log \Delta} + \sqrt{\frac{\log n}{\log \log n}}\big)$-round maximal matching algorithm in the LOCAL model [34]. A very recent result further shows that there are also no randomized $o\big(\Delta + \frac{\log \log n}{\log \log \log n}\big)$-round algorithm and no deterministic $o\big(\Delta + \frac{\log n}{\log \log n}\big)$-round algorithms to compute a maximal matching [7].

Distributed maximum matching approximation algorithms. There is a series of papers that target the distributed maximum matching problem directly and that provide results that go beyond the 1/2-approximation achieved by computing a maximal matching. Most of them are based on the framework of Hopcroft and Karp [29]: after $O(1/\varepsilon)$ iterations of augmenting along a (nearly) maximal set of vertex-disjoint short augmenting paths, one is guaranteed to have a $(1-\varepsilon)$-approximate solution for the maximum matching problem. The first distributed algorithms to use this approach are an $O(\log^{O(1/\varepsilon)} n)$-time deterministic LOCAL algorithm for computing a $(1-\varepsilon)$-approximation in graphs of girth at least $2/\varepsilon - 2$ [13] and an $O(\log^4 n)$-time deterministic LOCAL algorithm for computing a 2/3-approximation in general graphs [14]. The first approximation algorithms in the CONGEST model are by Lotker et al. [36], who give a randomized algorithm to compute a $(1-\varepsilon)$-approximate maximum matching in time $O(\log n)$ for every constant $\varepsilon > 0$. For bipartite graphs, the running time of the algorithm depends polynomially on $1/\varepsilon$, whereas for general graphs it depends exponentially on $1/\varepsilon$.[1] The algorithm was recently improved by Bar Yehuda et al. [8], who give an algorithm with time complexity $O\big(\frac{\log \Delta}{\varepsilon^3 \log \log \Delta}\big)$ for computing a $(1-\varepsilon)$-approximation. As in [36], the time depends polynomially on $1/\varepsilon$ in bipartite graphs and exponentially on $1/\varepsilon$ in general graphs. Note that the time dependency on Δ in [8] matches the lower bound of [34]. In [2], Ahmadi et al. give a deterministic $O\big(\frac{\log \Delta}{\varepsilon^2} + \frac{\log^2 \Delta}{\varepsilon}\big)$-round CONGEST maximum matching algorithm that has an approximation factor of $(1-\varepsilon)$ in bipartite graphs and an approximation factor of $(2/3-\varepsilon)$ in general graphs. Unlike the previous algorithms, the algorithm of [2] is not based on the framework of Hopcroft and Karp. Instead, the algorithm first computes an almost optimal fractional matching and it then rounds the fractional solution to an integer solution by adapting an algorithm of [21]. There also exist deterministic distributed algorithms to $(1-\varepsilon)$-approximate maximum matching in polylogarithmic time [19, 22, 24], these algorithm however require the LOCAL model. The algorithms of [2, 22, 24] directly also work for the maximum weighted matching problem. Other distributed algorithms that compute constant-factor approximations for the weighted

[1] In the LOCAL model, the algorithm can be implemented in time $O(\log(n)/\operatorname{poly}(\varepsilon))$ also for general graphs. This was independently also shown in a concurrent paper by Nieberg [42].

maximum matching problem appeared in [8, 21, 28, 36, 37, 46]. We note that none of the existing approximation algorithms can be used to solve the exact maximum matching problem in time $o(|E|)$ in the CONGEST model.

Additional related work. Our result can also be seen in the context of verification and the results established regarding decision problems in distributed settings. As an example, in [15], lower bounds for decision problems like connectivity, spanning connected subgraph, and $s-t$ cut verification are presented, and the applications of these results in deriving strong unconditional time lower bounds on the hardness of distributed approximation are shown. Another example can be seen in [33], where tight bounds are presented to verify whether a given subgraph is an MST of the network. For further study in this regard, see [4, 20]. Another context in which our result can be seen is the recent interest in the complexity of computing exact solutions to distributed optimization problems. In particular, it was recently shown that several problems that are closely related to the maximum matching problem have near-quadratic lower bounds in the CONGEST model. In [11], it is shown that computing an optimal solution to the maximum independent set and the minimum vertex cover problem both require time $\tilde{\Omega}(n^2)$ in the CONGEST model. In [6], similar $\tilde{\Omega}(n^2)$ lower bounds are proven for other problems, in particular for computing an optimal solution to the minimum dominating set problem and for computing a $(7/8+\varepsilon)$-approximation for maximum independent set. Consequently, for maximum independent set, minimum vertex cover and minimum dominating set, the trivial $O(|E|)$-time CONGEST model algorithm is almost optimal. If our result can be extended to actually find the maximum matching in almost linear time, it would show that this is not true for the maximum matching problem.

Mathematical notation. Before giving an outline of our algorithm in Section 2, we introduce some graph-theoretic notation that we will use throughout the remainder of the paper. A walk W from node u to a node v in a graph $G=(V,E)$ is a sequence of nodes $\langle u=v_1, v_2, \ldots, v_k = v\rangle$ such that for all $j<k$, $\{v_j, v_{j+1}\} \in E$. A path P is a walk that is cycle-free, i.e., a walk where the nodes are pairwise distinct. Let $\mathbb{V}(W)$ denote the multi-set of the nodes in a walk W and let $|W|$ denote the length of the walk W, i.e., $|W| = |\mathbb{V}(W)| - 1$. For simplicity, we write $v \in W$ if $v \in \mathbb{V}(W)$. Moreover, we say an edge e is on walk W and write $e \in W$ if e is an edge between two consecutive nodes in W. For two walks $W_1 = \langle u_1, \ldots, u_s\rangle$ and $W_2 = \langle v_1, \ldots, v_t\rangle$ with $u_s = v_1$, we use $W_1 \circ W_2$ to denote the concatenation of the walks W_1 and W_2. Further, for a path $P = \langle u_1, u_2, \ldots, u_i, \ldots, u_j, \ldots, u_k\rangle$, we use $P[u_i, u_j]$ to denote the consecutive subsequence of P starting at node u_i and ending at node u_j, i.e., the subpath of P from u_i to u_j. We use parentheses instead of square brackets to exclude the starting or ending node from the subpath, e.g., $P(u_i, u_j]$, $P[u_i, u_j)$ or $P(u_i, u_j)$.

2 Outline of Our Approach

For a graph G, it is well-known that a matching M is a maximum matching of G if and only if there is no augmenting path in G w.r.t. M. By performing a broadcast/convergecast, the size of the given matching can be learnt by all nodes in the graph in time linear in D, the diameter of G. After all nodes learn the size of the given matching, the algorithm looks for an augmenting path of length at most r in phases for exponentially increasing guesses r, i.e., where r is initially set to D and it doubles from each phase to the next. The algorithm stops as soon as either $r > 4|M|$ or it detects a shortest augmenting path of length at most r. Note that the length of an augmenting path cannot be more than $2|M|+1$. Therefore, if the

algorithm does not find a shortest augmenting path, then there is no augmenting path. The efficiency of the algorithm depends on how fast one can detect the existence of a shortest augmenting path of length at most ℓ in G for an integer ℓ. Note that although the algorithm looks for a shortest augmenting path, detecting an approximately shortest augmenting path suffices here to efficiently verify the matching. The following lemma states that this central challenging task can be done efficiently.

▶ **Lemma 2.** *Given an arbitrary graph G and a matching M of G, there is a randomized algorithm to detect whether there exists an augmenting path of length at most ℓ in $O(\ell)$ rounds of the* CONGEST *model, with high probability.* [2]

Theorem 1 now follows directly from Lemma 2 and the above algorithm outline. If ℓ is the maximum length for which we apply Lemma 2, the time for the algorithm is $O(D+\ell)$ (as we use exponentially increasing values for ℓ, the time is dominated by the time for the largest length ℓ for which we look for an augmenting path). If M is not maximum, ℓ is at most twice the length of the shortest augmenting path, if M is maximum, we have $\ell = O(|M|)$ and we thus get a time of $O(D + |M|) = O(|M|)$.

2.1 Detecting a Shortest Augmenting Path: The Challenges

Let $G = (V, E)$ be a graph, let $M \subseteq E$ be a matching of G, and let $f \in V$ be a free node (i.e., an unmatched node). Assume that we want to find a shortest augmenting path P connecting f with another free node f'. If the graph G is bipartite, such a path can be found by doing a breadth first search (BFS) along alternating paths from f. This works because in bipartite graphs, for every node v on a shortest augmenting path P connecting f with another free node f', the subpath $P[f, v]$ connecting f and v is also a shortest alternating path between f and v. In [45], Vazirani calls this property, which holds in bipartite graphs, the *BFS-honesty property*. If this property holds, to find a shortest alternating path from a free node f to a node v, it suffices to know shortest alternating paths from f to all the nodes along this path. The BFS-honesty property does not hold in general graphs. A simple example that shows this is given in Figure 1. The shortest alternating path connecting node f with u is of length 3. The shortest alternating path connecting f with v is of length 5 and it contains node u, however the subpath connecting f with u on the alternating path to v is of length 4.

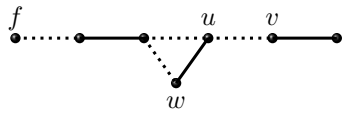

Figure 1 The BFS-honesty property does not hold in general graphs (solid lines depict edges in the matching, and dotted lines depict edges not in the matching).

To show the use of the BFS-honesty property in the distributed setting more clearly, we next sketch the algorithm of [2] for finding a shortest augmenting path in a bipartite graph. The algorithm essentially works as follows. Every free node $f \in V$ in parallel starts its own BFS exploration of G along alternating paths. The exploration of a free node f is done by propagating its ID (i.e., f) along alternating paths from f, where the ID is propagated by one more hop in each synchronous round. Whenever a node u receives the IDs of two

[2] An event is said to happen with high probability (w.h.p.) in graph G of size n if it happens with probability at least $1 - (1/n)^c$ for some constant $c > 0$.

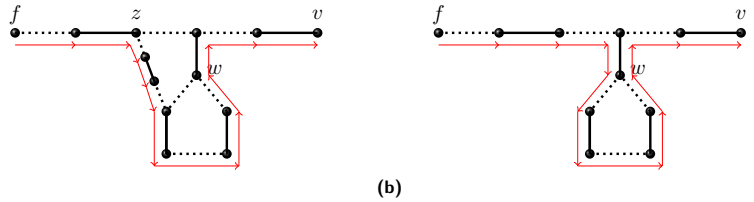

Figure 2 Main challenge: Nodes need to be able to distinguish whether the BFS exploration reaches them on an alternating path or only on alternating walks.

different free nodes f and f' in the same round, it only forwards the ID of one of them. Note that each node only forwards a single free node ID and it only forwards this ID once (in the round after it first receives it). This is sufficient if the BFS-honesty property holds. Moreover, this guarantees that IDs only traverse alternating paths and avoid traversing cycles. Assume that the shortest augmenting path in the graph is of length $\ell = 2k+1$. Let P be such a path and assume that $\{u, v\}$ is the middle edge of P. Note that this implies that the shortest alternating paths of nodes u and v are both of length k. Hence, u and v receive the ID of a free node exactly in round k and they will both forward that ID along edge $\{u, v\}$ in round $k+1$. When this happens, u and v learn about the fact that they are in the middle of an augmenting path of length $2k+1$ and that path can be constructed simply by following back the edges on which the alternating BFS traversals reached nodes u and v.

Let us now discuss some of the challenges when adapting this ID dissemination protocol to general graphs. For simplicity, assume that we are only doing the BFS exploration from a single free node f. Consider again the example in Figure 1. We have seen that the shortest alternating path from f to v passes through node u, however the subpath from f to u is not the shortest alternating path from f to u. In fact, while the shortest alternating path from f to u reaches u on an unmatched edge, in order to reach node v, we have to use the shortest one of the alternating paths from f to u that reaches u on a matched edge. This suggests that each node v should keep track of both kinds of shortest paths from node f and that v should forward f twice. A natural generalization of the protocol would thus be the following: After receiving f on a shortest alternating path ending in an unmatched edge of v, v forwards f on its matched edge and after receiving f on a shortest alternating path ending in the matched edge of v, v forwards f on its unmatched edges. One would hope that this lets each node detect both kinds of shortest alternating paths from node f. However, as Figure 2 shows, this is not necessarily true. While in the Figure 2a, when v receives f over its matched edge, the ID was indeed forwarded on a shortest alternating path from f to v. However, in Figure 2b, the exploration passes through an odd cycle and node v is only reached on an alternating walk instead of an alternating path. In the example of Figure 2b, node w should detect that the BFS traversal passed through the odd cycle and w should therefore not forward f over its matched edge. However, it is not clear how w should distinguish between the cases in Figure 2a and Figure 2b. Note that in the BFS traversal of Figure 2a, f is not only forwarded on the alternating path to w, but also through the odd cycle as in Figure 2b. In fact, the example of Figure 2 is still a relatively simple case as odd cycles can be nested, and closed odd walks can look much more complicated than just passing through a single odd cycle. Detecting whether and when to forward the ID of a free node is the main algorithmic challenge that we face.

A second challenge comes from the fact that we need to do alternating BFS explorations from all free nodes and it is not obvious how to coordinate these parallel BFS explorations while keeping the message size small. In the bipartite case, it was enough for each node v

to only participate in the BFS exploration of a single free node f and to discard all other BFS traversals that reach node v. It is not clear whether the same thing can also be done in general graphs. Luckily it turns out to still be sufficient if each node v only participates in the BFS exploration of the first free node that reaches v.

On a very high level, our approach resembles the basic underlying idea of the classic maximum matching algorithm of Edmonds [17]. As in Edmonds algorithm, we grow a forest of alternating trees from the unmatched (free) nodes. However, in order to find a shortest (or almost shortest) augmenting path, we have to grow these trees in a parallel BFS-like fashion. Note however that in a non-bipartite setting, each node of a graph might have an odd and an even-length alternating path to each free node and the shortest paths of both kinds might be of very different lengths. It is therefore not immediately clear how to grow forests in parallel such that in the end, one finds a *short* augmenting path. The following free node clustering (which we think might be of independent interest) shows that there is a relatively straightforward parallel alternating BFS construction that allows to indeed find a shortest augmenting path. The main technical challenge of our paper will be to construct this free node clustering in a distributed way. Note that unlike in Edmonds algorithm, we cannot just contract odd cycles when we find them. In fact, to be efficient, we cannot even detect odd cycles explicitly, but we have to deal with them in an implicit way.

2.2 The Free Node Clustering

We start the outline of our algorithm to detect a shortest augmenting path by describing the required outcome of the alternating BFS exploration in general graphs in more detail. We intend to perform BFS explorations starting from all the free nodes $f_1, \dots, f_\rho$ in parallel. We will show that it is sufficient for each node $v \in V$ to participate in the BFS exploration for exactly one free node f_i. This implies that at each point in time, the BFS explorations of the different free nodes $f_1, \dots, f_\rho$ induce a clustering of the nodes in V. There is a cluster for each free node f_i, and each node $v \in V$ is either contained in exactly one of the ρ clusters or it is not contained in any cluster (i.e., has not been reached by any of the explorations). We call this induced clustering the *free node clustering*. The clustering is computed in synchronous rounds and we will guarantee that it satisfies the following properties.

(C1) Consider some node $v \in V$ and some round number $r \geq 1$. If v has not joined any cluster in the first $r-1$ rounds, if v has an alternating path P of length r to a cluster center f (i.e., a free node f), and if all nodes of P except node v are in the cluster of node f after $r-1$ rounds, then v joins the cluster of f or some other cluster with the same property. That is, v joins a cluster in round r such that afterwards, it has an alternating path P' of length r to its cluster center such that P' is completely contained in the cluster that v joined. Let C be the cluster that v joins and let $U \subseteq C$ be the set of neighbors u of v such that the cluster contains an alternating path of length r from v through u to the cluster center. The set U is called the *predecessors* of v. If v joins a cluster in round r, we say that v is r-*reachable* (i.e., v's shortest alternating paths to its cluster center that are completely contained in the cluster are of length r). If v's incident edge on the shortest alternating path of v is an unmatched edge (i.e., if r is odd), we say that v is r-0-*reachable* and otherwise, we say that it is r-1-*reachable*.

(C2) Assume that v is r_v-reachable and f is the center of the cluster to which v already belongs. Let $r > r_v$ be the first round after which the cluster of v contains an alternating path P of length r connecting v with f such that if v is r_v-0-reachable, P starts with a

matched edge at node v and if v is r_v-1-reachable, P starts with an unmatched edge at node v (if such a round r exists). Then, after r rounds of the construction, v is aware of the existence of such a path. If v is r_v-0-reachable, we say it is also r-1-*reachable* and if it is r_v-1-reachable, we say that it is also r-0-*reachable*.

To put it differently, a node in a cluster is called r-0-reachable (r-1-reachable) if there is a shortest odd-length (even-length) alternating path of length r from the cluster center to the node that is completely contained in the cluster. The clustering after r rounds of the construction will be called the *r-radius free node clustering*. For the precise definition of the clustering and of the related terminology, we refer to Section 3.1. We will see that the r-radius free node clustering can be constructed in r rounds in the CONGEST model. We give an outline of the distributed construction of the clustering in the following Section 2.3. The details of the distributed construction and its analysis appear in Section 3 and the full version [1]. Before discussing the distributed construction, we next sketch how the free node clustering can be used to detect an augmenting path and why it is sufficient for the detection.

Detecting augmenting paths. After computing the r-radius free node clustering for a sufficiently large radius r, we can use it to detect the existence of an augmenting path as follows. Let u and v be two neighbors in G such that u and v are in different clusters (say for free nodes f and f'). Assume that for two integers $\ell_u, \ell_v \geq 0$ one of the following conditions hold:

1. The edge $\{u, v\}$ is in the matching, u is ℓ_u-0-reachable (in its cluster), and v is ℓ_v-0-reachable (in its cluster).
2. The edge $\{u, v\}$ is not in the matching, u is ℓ_u-1-reachable (in its cluster), and v is ℓ_v-1-reachable (in its cluster).

In both cases the matching directly implies that there exists an augmenting path of length $\ell_u + \ell_v + 1$ between the free nodes f and f'. Further, after $\max\{\ell_u, \ell_v\} + 1$ rounds, u and v are aware of the existence of this path. Note that this only gives the existence of a shortest augmenting path between f and f', but it does not allow to construct such a path. To construct the path, nodes would need to know their next node on the path towards the free nodes. However, since the subpaths of a shortest alternating path of a node to some free node are not shortest alternating paths themselves (i.e., since the BFS-honesty property does not hold), the information provided by the clustering does not suffice to compute the path.

Detectability of a shortest augmenting path. It remains to show that the free node clustering allows to find some shortest augmenting path. However, note that detecting a relatively short augmenting path suffices to efficiently verify the matching. Assume that the length of a shortest augmenting path in G w.r.t. the given matching M is $2k + 1$ for some integer $k \geq 0$. For an augmenting path $P = \langle f = v_0, \ldots, v_\ell = f' \rangle$ of length $\ell = 2k + 1$ between two free nodes f and f', we let $i \geq 0$ and $j \geq 0$ be two integers such that i is the largest integer such that all nodes in $P[v_0, v_i]$ are in the cluster of f and such that j is the largest integer such that all the nodes in $P[v_{\ell-j}, v_\ell]$ are in the cluster of f'. We define the *rank* of the augmenting path P as $i + j$. Note that the path P is detectable if and only if it has rank $2k$. We thus need to show that there exists a shortest augmenting path of rank $2k$.

To prove that there is a shortest augmenting path of rank $2k$, we assume that P is a shortest augmenting path of maximal rank and that the rank of P is less than $2k$ and we show that this leads to a contradition: this either allows to construct an augmenting path of length less than $2k + 1$ or it allows to construct an augmenting path of length $2k + 1$ of larger rank. The actual proof is somewhat technical. It consists of two steps. If we assume,

w.l.o.g., that $i \leq j$, we first show inductively that all the nodes $v_{i+1}, \ldots, v_{\max\{i+1,j\}}$ are in the cluster of f'. If $j \geq \ell - j - 1$, we have proven that $v_{\ell-j-1}$ is in the cluster of f', which is a contradiction to the choice of j. Otherwise, node $v_{\ell-j-1}$ is in a cluster $f'' \neq f'$ (it is however possible that $f'' = f$). We can now derive the desired contradiction by a careful concatenation of parts of the paths connecting f'' with $v_{\ell-j-1}$, parts of the augmenting path between f and f', and parts of a path between f' and v_{i+1} that was constructed in the earlier inductive argument. The details of the arguments appear in Section 3.2.

2.3 Distributed Construction of the Free Node Clustering

We focus on a single step (round) of the the distributed construction of the free node clustering. To that end, consider graph G and matching M, and assume that the first $r-1$ rounds of the clustering construction have been done successfully and the introduced clustering properties (C1) and (C2) of Section 2.2 hold. Therefore, for all integers $t < r$ and $\vartheta \in \{0, 1\}$, every t-ϑ-reachable node correctly detects the fact that it is t-ϑ-reachable and knows its predecessors. Then, let us explain the outline of the approach towards implementing the r^{th} step of the distributed construction of the clustering.

Let us first focus on maintaining property (C1). To satisfy (C1), every $(r-1)$-ϑ-reachable node sends its cluster ID over its adjacent matched edge if $\vartheta = 0$ and over its adjacent unmatched edges if $\vartheta = 1$. This way, for every node that receives a cluster ID over its adjacent edge, by joining the corresponding cluster, there would be an alternating path of length r from the cluster center to the node such that the path is completely contained in the cluster. This maintains property (C1) for r steps of the clustering and it can be achieved in a distributed setting as explained. However, maintaining property (C2) is the main challenge as we try to elaborate in the sequel.

Let us consider a node v in the cluster centered at some free node f after $r-1$ steps of the clustering construction such that it is r'-ϑ-reachable for some integers $r' < r$ and $\vartheta \in \{0, 1\}$. Let us then assume that after the nodes have joined their corresponding clusters in the r^{th} step, there is an alternating path of length r from f to v that has completely fallen into the cluster centered at f such that the path contains an adjacent matched edge of v if $\vartheta = 0$ and an adjacent unmatched edge of v otherwise. Therefore, v should learn about the existence of such a path to maintain property (C2). Nodes like v can be reached within the cluster from their corresponding cluster center in two ways; first through an alternating path from the cluster center to v such that the path is completely contained in the cluster and contains a matched edge of v, and second through a similar path but containing an unmatched edge of v. Let us call these nodes that can be reached via both kinds of paths *bireachable nodes*.

Let us define an *odd cycle* to be an alternating walk of odd length that is completely contained in a cluster and starts and ends at the same node. Node v that is the first and last node of an odd cycle is called the stem of the odd cycle. An odd cycle is said to be *minimal* if it has no consecutive subsequence that is an odd cycle. Note that a minimal odd cycle can still have a consecutive subsequence that is an even-length cycle. An odd cycle is moreover said to be *reachable* if either the stem is the cluster center or there is an alternating path from the cluster center to the stem of the odd cycle such that (1) it is completely contained in the cluster, (2) it is edge-disjoint from the odd cycle, and (3) it includes the matched edge of the stem of the odd cycle. You can see examples of reachable minimal odd cycles in Figure 2a, one with stem w and another with stem z. All the nodes of an odd cycle except the stem are said to be *strictly inside* the odd cycle. Then, one can show that a node is bireachable if and only if it is strictly inside a reachable minimal odd cycle. We only need this simple observation to explain the intuition behind our approach for maintaining property (C2), and

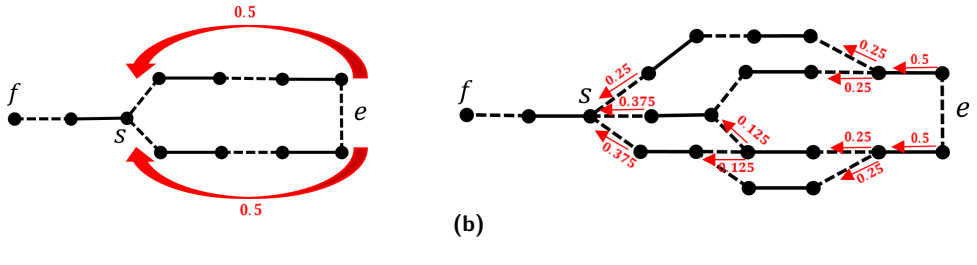

Figure 3 Flow circulation in reachable minimal odd cycles.

in the full version of the paper [1], we formally prove the correctness of the approach. As an example, node w is strictly inside the reachable minimal odd cycle with stem z in Figure 2a and hence bireachable, but w is not bireachable in Figure 2b.

To help the nodes to distinguish whether they are strictly inside a reachable minimal odd cycle or not, we define a flow circulation protocol throughout each cluster. Let us consider the very simple example of a reachable minimal odd cycle in Figure 3a. When the cluster ID is sent over the middle edge of this odd cycle (i.e., e) in both directions in the same round, we consider a flow generation of unit size over the edge and we call it the flow of e. Then, half of the generated flow is sent back towards the stem of the odd cycle on each of the two paths. When the stem receives the whole unit flow of edge e in a single round, it learns that it is the stem of an odd cycle for which the flow is generated and discards the flow (it avoids sending the flow further). Whereas all the other nodes inside the odd cycle receive a flow of value less than 1. They interpret this incomplete flow receipt as being strictly inside a reachable minimal odd cycle. Moreover, they interpret the round in which they receive an incomplete flow for the first time as the length of an existing alternating path from the cluster center. Therefore, to maintain property (C2), a node detects the length of its shortest alternating path through its matched (unmatched) edge by receiving an incomplete flow for the first time if its shortest alternating path contains its unmatched (matched) edge.

Many such odd cycles might share a common middle edge as the cycles in Figure 3b that share e as their middle edge. Then, it is enough that every node divides the value of the received flow of e and sends them backwards until the cycle's stem, i.e., node s, receives the whole unit flow of e. However, in case of having many interconnected and nested reachable minimal odd cycles that do not share a single middle edge, an edge might carry the flows of many different edges in the same round. This is a problem when implementing the idea in the CONGEST model as we cannot bound the number of flows that have to be sent concurrently over an edge. Instead of separately sending flows generated at different edges e, we therefore sum all flows that have to go over the same edge and only send aggregate values. Ideally, we would like to have the following desired differentiation; a node that receives an aggregated flow whose size is not an integer, learns that it is strictly inside at least one reachable minimal odd cycle, and a node that receives an aggregated flow whose size is an integer learns that it is the stem of at least one reachable minimal odd cycle but not strictly inside any such cycle and it discards the flow. To avoid that a set of fractional flows for different edges sums to an integer, we can use randomization. Instead of always equally splitting flow that has to be sent over several edges, we randomly split the flow. This guarantees that w.h.p., flows only sum up to an integer if they consist of all parts of all involved separate flows. Unfortunately, this is still not directly implementable in the CONGEST model because we might need to split a single flow a polynomial in n many times and $O(\log n)$ bits then are not sufficient to forward

the flow value with sufficient accuracy. In order to apply the idea in the CONGEST model, we instead use flow values from a sufficiently large (polynomial size) finite field. In [1], we show that this suffices to w.h.p. obtain the same behavior as if flows for each edge were sent separately. Aggregating flows thus allows to satisfy the congestion requirement, it however causes a number of further challenging problems, which we present and discuss next.

Let us consider the rather basic example of having only two nested reachable minimal odd cycles in Figure 4. Let C denote the odd cycle with stem s and C' denote the odd cycle with stem s'. Observe that e is the middle edge of C and e' is the middle edge of C'. The received flow of e by x must be sent to y_1 whereas the received flow of e' by x must be sent to y_2, which requires node x to treat the two flows differently. That is, node x must recognize that the received flow of e corresponds to the odd cycle containing the alternating path ending at edge $\{y_1, x\}$, and the received flow of e' corresponds to the odd cycle containing the alternating path ending at edge $\{y_2, x\}$. This cannot be achieved due to the flow aggregation enforced by the congestion restriction. Therefore, node x is not capable of correctly directing the flows along the right paths so that the flows of e only traverse the paths of cycle C and the flows of e' only traverse the paths of cycle C'.

To resolve this issue and be able to still aggregate the flows, nodes should be able to treat all flows in the same way. Therefore, since every node knows its predecessors, we would like to establish the generic regulation of always sending flows only towards all predecessors no matter what the flow is. However, by letting node x send the received flow of e' to its only predecessor y_1, the nodes in the alternating path between x and s' through y_2 do not anymore receive any flow of e'. To fix this and keep node x free of treating flows differently, we eliminate the flow generation over e' and simulate it by generating a flow over e''. That is, we shift the flow generation of cycle C' from e' to e''. Then, half of the flow of e'' is sent by y_2 to its predecessor y_4, and half of it is sent by x to its predecessor y_1.

Previously, we let the flow generation only occur over the middle edge of an odd cycle, that can be easily recognized when an edge carries the same cluster ID in opposite directions in the same round. Now by having flow generation over both middle edges like e as well as non-middle edges like e'', we need a more involved flow generation regulation. Let every node send its cluster ID in at most one round over its matched edge and in at most one round over its unmatched edges. A node sends its cluster ID over its matched edges in round r if it is $(r-1)$-0-reachable, and it sends its cluster ID over its unmatched edges in round r' if it is $(r'-1)$-1-reachable. We let a flow be generated over an edge when the two endpoints are not each other's predecessors and they both send the same cluster ID to each other, no matter if they are sent in the same round or not. Neither the endpoints of e nor those of e'' are each other's predecessors while the endpoints send f to each other over e and e''. Therefore, flow generation occur over both e and e'', where e is an example of a middle edge that the endpoints send cluster IDs in the same round, and e'' is an example of a non-middle edge for a shifted flow generation that the endpoints send cluster IDs in different rounds. Also note that since y_3 is the predecessor of y_6, a flow is not anymore generated over e' within this new regulation.

Now to see that shifting flow generation maintains the desired effects and avoids any side effects, let us compare the two cases of flow generation over e' and its simulation over e''. Each half of the flow of e' is sent towards s', one along the path between y_6 and s' through y_5 and one along the path between y_3 and s' through y_4 as depicted by arrows in Figure 4. In the simulation, each half of the flow of e'' is also sent towards s', one along the path between y_2 and s' through y_4 and one along the path between x and s' through y_1 again as depicted by arrows in Figure 4. We need the simulation to serve the purposes of the flow

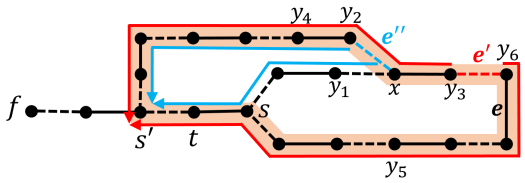

Figure 4 Nested odd cycles: flow simulation of edge e' on edge e''.

generation of e'. However, there are two crucial differences that might question the desired effects of flow e' if we run the simulation instead. To explain the first difference, consider the nodes inside odd cycle C. Nodes like y_1 do not receive flows of e' but receive flows of e'' in the simulation. Moreover, nodes like y_5 receive flows of e' but not flows of e'' in the simulation. The second difference is that node s' as the stem of C' receives the whole unit flow of e' in a single round as desired to perceive the fact that it is the corresponding stem and discards the flow. However, in the simulation since e'' is not the middle edge of C' and the flows are sent along paths with different lengths, s' does not receive the whole unit flow of e'' in a single round. This avoids node s' to perceive the fact that it is the stem of an odd cycle and the flow is further sent by s'. Let us see how crucial these differences are and how we can resolve them.

Regarding the first difference, the decisive observation is that whenever a node receives a proper fraction of a flow for the first time in round r, it detects the existence of an alternating path of length r. Therefore, only the first receipt of such flow is important and must be at the right time for a node. All those nodes in C that differ in receiving the corresponding flows in the flow circulation of e' and e'' already have received a proper fraction of flow e, and hence the receipt and the time of receiving later flows are irrelevant to them.

However, the second difference is crucial and needs to be resolved. Note that the half flow of e'' is sent along the path between y_2 and s' through y_4 that is the same path traversed by the half flow of e'. Therefore, if y_2 sends the half flow of e'' to y_4 immediately in the next round of receiving the cluster ID from x, it reaches s' at exactly the same time as the half flow of e' would have reached s'. Now assume that x would also have sent the other half flow of e'' along the path between x and s' through y_5. If x would have sent the flow in the very next round of receiving the cluster ID from y_2, then the flow of e'' would also have reached s' in exactly the same round as the flow of e' would have reached s'. However, since x actually sends this flow along the path between x and s' through y_1, it reaches s' sooner. This time difference is the difference of the length of the shortest alternating path between x and f ending in the matched edge of x and such a path ending in the unmatched edge of x. This difference is known by x, and x can therefore delay sending the flow by this number of rounds and repair the unwanted side effects of the simulation. Note that x cannot send the flow in the very next round of receiving the cluster ID from y_2 since it has not yet decided at that time to send its cluster ID to y_2 and hence cannot yet recognize the flow generation over e''. Therefore, it has to anyway send the flow along a shorter path, e.g., the path through y_1.

This discussed simple example inevitably abstracts away some details. In the example, flows are only sent over alternating paths. However, if nodes always send flows to their predecessors, flows do not necessarily traverse alternating paths and the paths along which flows are sent might have consecutive unmatched edges. Then, along a path that a flow is forwarded, every node that has two adjacent unmatched edges on the path delays forwarding the flow.

3 Shortest Augmenting Path Detection

In this section, we present the algorithm to detect a shortest augmenting path in time linear in the length of the path in the CONGEST model. The organization of this section is as follows. In Section 3.1, we formally define the free node clustering that was described in Section 2.2. We define the clustering by giving a deterministic sequential algorithm that constructs the clustering in a step-by-step manner. Note that this deterministic algorithm is only for the purpose of providing a precise definition of the clustering. Then, in Section 3.2, we show that given such a clustering, at least one shortest augmenting path can be detected in a single round of the CONGEST model. In Section 3.3, we provide a distributed algorithm to construct the free node clustering. Due to lack of space, the analysis of the distributed free node clustering algorithm appears in the full version of the paper [1]. For the sake of simplicity, we first consider no restriction on the size of sent messages when we describe the algorithm.

3.1 The r-Radius Free Node Clustering

To define the r-radius free node clustering of a graph G w.r.t. a given matching M of G, we introduce a deterministic sequential r-step algorithm, which we henceforth call the FNC algorithm. The free nodes $f_1, \dots, f_\rho$ are the cluster centers. For all i, let C_i denote the cluster that is centered at free node f_i. Initially every cluster C_i only contains f_i and during the execution of the algorithm more nodes potentially join the cluster. For consistency, we assume that there exists a step 0 in which every free node joins the cluster centered at itself, i.e., initially $\forall i \in [1, \rho] : C_i = \{f_i\}$. Then, in every step $t \geq 1$, every node that has not yet joined any cluster, concurrently joins the cluster centered at f_i if and only if f_i is the minimum-ID free node from which v has an alternating path P of length t such that $\mathbb{V}(P) \setminus \{v\} \subseteq C_i$. Note that there might be some nodes in G that never join any cluster in any step of the FNC algorithm. Throughout, let $C_i(t)$ denote the set of nodes in cluster C_i after t steps of the FNC algorithm. We define $\mathcal{C}(r) := \{C_1(r), \dots, C_\rho(r)\}$ to be the r-radius free node clustering of G.

To simplify the discussions, we introduce the following definitions and a simple observation in the next lemma. In the following definitions, v is an arbitrary node in G and ϑ is an arbitrary integer in $\{0, 1\}$. We say that P is a *path of* v or v *has a path* P if P is a path starting at a free node and ending at node v.

▶ **Definition 3** (Uniform Paths). *We say that a path P is uniform at time $t \geq 0$ if $\mathbb{V}(P) \subseteq C_i(t)$ for some $i \in \{1, \dots, \rho\}$. When the time t is clear from the context, we just say that P is uniform.*

▶ **Lemma 4.** *Let P be an alternating path of length r from any free node to any node. If there is any time (possibly larger than r) at which P is uniform, then P is uniform at time r.*

Proof. Due to lack of space, the proof is presented in the full version of the paper [1]. ◀

▶ **Definition 5** (Almost Uniform Paths). *We say that a path P of v is almost uniform (at time t) if $\mathbb{V}(P) \setminus \{v\}$ is uniform (at time t). Note that every uniform path of v is also almost uniform.*

▶ **Definition 6** (ϑ-Edges). *A free (i.e., an unmatched) edge is called a 0-edge, and a matched edge is called a 1-edge.*

▶ **Definition 7** (ϑ-Paths). *An alternating path P of v is called a ϑ-path of v if P contains a ϑ-edge adjacent to v.*

▶ **Definition 8** (Predecessors). *We say u is a predecessor of v if u is the neighbor of v on a shortest uniform alternating path of v.*

▶ **Definition 9** (Reachability). *For an integer $r \geq 0$, we say that v is r-ϑ-reachable if v has a shortest uniform ϑ-path of length r. Moreover, we say that v is r-reachable if it has a shortest uniform alternating path of length r.*

For the sake of consistency, we assume that every free node is 0-0-reachable and 0-1-reachable

3.2 Detecting a Shortest Augmenting Path

In this section, we show how the existence of a shortest augmenting path of length at most ℓ can be detected in a single round of the CONGEST model if the nodes of G are provided with the ℓ-radius free node clustering with respect to a given matching M of G and if in addition the ℓ-radius free node clustering is *well-formed* in the following sense.

▶ **Definition 10** (Well-Formed Clustering). *The r-radius free node clustering $\mathcal{C}(r)$ is said to be well-formed in a distributed setting if for all $r' \leq r$, $\vartheta \in \{0,1\}$ and i, every r'-ϑ-reachable node $v \in C_i(r)$, beyond knowing its cluster ID, knows its predecessors and the fact of being r'-ϑ-reachable.*

▶ **Definition 11** (Rank of an Augmenting Path). *For an integer ℓ, consider an arbitrary augmenting path P of length ℓ between any pair of free nodes f_s and f_t. Let us name the nodes of P as $\langle f_s = u_0, \ldots, u_\ell = f_t \rangle$ and let i and j be the largest integers such that the subpaths $P[f_s, u_i]$ and $P[u_{\ell-j}, f_t]$ are uniform. Then, we define the rank of P to be $i + j$.*

▶ **Lemma 12.** *Let all nodes of a given graph G be provided with the well-formed r-radius free node clustering with respect to a given matching M of G. If there is a shortest augmenting path of length $\ell \leq r$, then a shortest augmenting path can be detected in a single round of the* CONGEST *model.*

Proof. Due to lack of space, the proof is presented in the full version of the paper [1]. ◀

3.3 Distributed Free Node Clustering

In this section, we present a distributed deterministic algorithm whose r-round execution provides the well-formed r-radius free node clustering. This algorithm uses large messages. However, as we discussed in Section 2.3, we can use randomness to adapt this algorithm to the CONGEST model.

Distributed r-Radius Free Node Clustering: DFNC Algorithm. The algorithm is run for r rounds. Let the following variables be maintained by the nodes during the execution; $r_v^{(0)}$ and $r_v^{(1)}$ keep track of the v's reachabilities, cid_v holds the cluster ID of v, and $pred_v$ holds the set of v's predecessors. At the beginning of the execution, for every free node v, variables $r_v^{(0)}$ and $r_v^{(1)}$ are set to 0, variable cid_v is set to v, and $pred_v$ is set to $\emptyset$. Moreover, for every matched node, all these variables are initially undefined and set to $\perp$. Every node v participates in the token dissemination based on the following simple rule. For an arbitrary integer $t \geq 1$, in round t:

- If $r_v^{(0)} = t-1$, then v sends cid_v over its adjacent 1-edge (if any). Otherwise, if $r_v^{(1)} = t-1$, then v sends cid_v over all its adjacent 0-edges (if any).

Based on the above simple rule, every node sends tokens to its neighbors in at most two rounds, at most once over its 1-edge and at most once over its 0-edges. Therefore, in the first round of the execution, every free node sends its ID to all its neighbors. Let round t be the first round in which a node v receives tokens. Let $\tau_1, \dots, \tau_j$ be the tokens that v receives from its neighbors in round t. Then, v sets cid_v to $\min_i \tau_i$, and subsequently sets $pred_v$ to the set of all its neighbors that sent cid_v to v in round t. There are two types of messages sent by the nodes throughout the execution; tokens (i.e., free node IDs) and *flow messages*. Node v sets $r_v^{(1)}$ and $r_v^{(0)}$ based on the received tokens and flow messages. Before we explain how these variables are set by v, let us first define the flow messages by explaining flow generation and circulation throughout the network.

Flow Generation: A *flow* is a key-value pair, where the key is an edge and the value is a real number in $[0, 1]$. Any flow whose key is some edge e is simply called a *flow of edge* e. A *flow message* is then defined to be a set of flows (i.e., key-value pairs) that are sent by a node to its neighbor in a round. A flow generation is an event that can only happen over an edge for which both endpoints belong to the same cluster. Let us fix an arbitrary edge $e = \{u, w\}$ where both endpoints belong to the same cluster. Then, we say that a flow is generated over edge e if and only if (1) none of u or w is the other one's predecessor, and (2) both u and w send tokens to each other. Note that we only consider at most one flow generation for every edge throughout the whole execution. Let us assume that u and w are not each other's predecessors, u sends token to w in round r_u, and w sends token to u in round r_w. Then, the flow generation over e is defined as an event in which u receives a singleton flow message $\{(e, 1/2)\}$ over e in round r_w and w receives a singleton flow message $\{(e, 1/2)\}$ over e in round r_u. It is important to note that nodes u and w might send tokens to each other in different rounds, i.e., $r_u \neq r_w$. However, they cannot perceive the flow generation over e before they make sure that e carries tokens in both directions. In particular, if $r_w < r_u$, in round r_w, node u cannot perceive the flow receipt over e since it does not yet know whether it will send a token to w. Hence, u will perceive this flow receipt of round r_w later in round r_u in which it decides to send token over e and then knows that the edge carries tokens in both directions. However, we will see that u does not need to know about the flow receipt of round r_w before round r_u.

Flow Circulation: No matter if it receives a flow over its adjacent 0-edges or its adjacent 1-edge, every node always forwards the received flow to its predecessors by equally splitting the flow value among them. When the edge over which v receives a flow and the edges connected to its predecessors are not all 0-edges (see Figure 5a and 5b), v forwards the flow immediately in the next round after receiving the flow. Otherwise (see Figure 5c), it delays forwarding the flow for $r_v^{(1)} - r_v^{(0)}$ rounds. A node furthermore avoids forwarding the whole flow of a single edge e in a single round (i.e., a flow of value 1 of e). Let us see the details of the flow circulation in the following. Let $I_v(t)$ denote the set of all the flows that a node v receives in a round t, i.e., the union of all the received flow messages by v in round t. Moreover, let $O_v(t, e)$ denote the *output buffer* of a node v for its adjacent edge e in a round t, which is initially an empty set and eventually sent as a flow message over e by v in round t. Now let us fix an arbitrary node v, where E_v is the set of v's adjacent edges that are connected to its predecessors. Node v updates its output buffers in two ways; (1) it updates them with respect to the received flows and (2) it updates them to avoid forwarding the whole unit flow of a specific edge in a single round. Regarding the former case, fix an arbitrary round t in which v receives flows. If the edges over which v receives flows and the edges in E_v are all 0-edges (Figure 5c), let $t' := t + r_v^{(1)} - r_v^{(0)} + 1$.

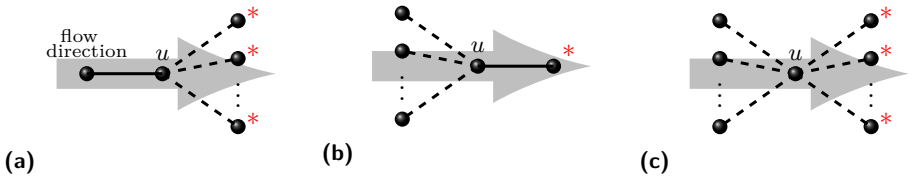

Figure 5 The 3 possibilities of flow forwarding. The predecessors of u are marked by asterisks.

Otherwise (see Figure 5a and 5b), let $t' := t+1$. Then, for every $(e, f) \in I_v(t)$, v updates its output buffers as follows:

$$\forall e' \in E_v : \quad O_v(t', e') \leftarrow O_v(t', e') \cup (e, \frac{f}{|E_v|}) \ .$$

Now regarding the latter way of output buffers update, fix an arbitrary round t'. At the beginning of round t', let $O_v(t')$ be the set of all the flows in the output buffers of v for round t', i.e., $O_v(t') := \bigcup_{e' \in E_v} O_v(t', e')$. Then, let $S_v(e, t')$ be the sum of flow values of a specific edge e that are sent by v in round t', i.e., $S_v(e, t') := \sum_{(e,f) \in O_v(t')} f$. For every edge e, if $S_v(e, t') = 1$, node v removes all flows of edge e from all its output buffers of round t'. That is, v removes all flows of e and we say that v *discards* the flow of e. After discarding all such flows, for every $e' \in E_v$, v forwards $O_v(t', e')$ over edge e' in round t' if $O_v(t', e') \neq \emptyset$.

Setting Variables $r_v^{(1)}$ and $r_v^{(0)}$ (Reachability Detection): We say that round t is an *incomplete round* for v if node v sends flow in round $t+1$. Let t be the first round in which v receives tokens or the first incomplete round for v. If t is an even integer, v assigns t to $r_v^{(1)}$, otherwise, v assigns t to $r_v^{(0)}$. The first round that a node receives a token (if any) is before its first incomplete round (if any) since it receives flows from the nodes it has already sent tokens to.

References

1 M. Ahmadi and F. Kuhn. Distributed maximum matching verification in congest, 2020. `arXiv:2002.07649`.

2 M. Ahmadi, F. Kuhn, and R. Oshman. Distributed approximate maximum matching in the CONGEST model. In *Proc. 32nd Symp. on Distributed Computing (DISC)*, 2018.

3 N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

4 H. Arfaoui, P. Fraigniaud, and A. Pelc. Local decision and verification with bounded-size outputs. In *Symposium on Self-Stabilizing Systems*. Springer, 2013.

5 S. Assadi, M. Bateni, A. Bernstein, V. Mirrokni, and C. Stein. Coresets meet EDCS: Algorithms for matching and vertex cover on massive graphs. In *Proc. 30th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1616–1635, 2019.

6 N. Bachrach, K. Censor-Hillel, M. Dory, Y. Efron, D. Leitersdorf, and A. Paz. Hardness of distributed optimization. *CoRR*, abs/1905.10284, 2019.

7 A. Balliu, S. J. Hirvonen, D. Olivetti, M. Rabie, and J. Suomela. Lower bounds for maximal matchings and maximal independent sets. In *Proc. 60th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 481–497, 2019.

8 R. Bar-Yehuda, K. Censor-Hillel, M. Ghaffari, and G. Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 165–174, 2017.

9 L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *Proceedings of 53th Symposium on Foundations of Computer Science (FOCS)*, 2012.

10 R. Ben-Basat, K. Kawarabayashi, and G. Schwartzman. Parameterized Distributed Algorithms. In *33rd International Symposium on Distributed Computing (DISC 2019)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

11 K. Censor-Hillel, S. Khoury, and A. Paz. Quadratic and near-quadratic lower bounds for the CONGEST model. In *Proc. 31st Symp. on Distributed Computing (DISC)*, pages 10:1–10:16, 2017.

12 A. Czumaj, J. Lacki, A. Madry, S. Mitrovic, K. Onak, and P. Sankowski. ound compression for parallel matching algorithms. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, pages 471–484, 2018.

13 A. Czygrinow and M. Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. In *9th Annual International Computing and Combinatorics Conference (COCOON)*, pages 242–251, 2003.

14 A. Czygrinow, M. Hańćkowiak, and E. Szymanska. A fast distributed algorithm for approximating the maximum matching. In *Proceedings of 12th Annual European Symposium on Algorithms (ESA)*, pages 252–263, 2004.

15 A. Das Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proc. 43rd Symp. on Theory of Computing (STOC)*, pages 363–372, 2011.

16 J. Edmonds. Maximum matching and a polyhedron with 0, l-vertices. *J. of Res. the Nat. Bureau of Standards*, 69 B:125–130, 1965.

17 J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

18 Y. Emek, S. Kutten, and R. Wattenhofer. Online matching: haste makes waste! In *Proc. 48th Symp. on Theory of Computing (STOC)*, pages 333–344, 2016.

19 G. Even, M. Medina, and D. Ron. Distributed maximum matching in bounded degree graphs. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking (ICDCN)*, pages 18:1–18:10, 2015.

20 L. Feuilloley and P. Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 2016.

21 M. Fischer. Improved deterministic distributed matching via rounding. In *Proc. 31st Symp. on Distributed Computing (DISC)*, pages 17:1–17:15, 2017.

22 M. Fischer, M. Ghaffari, and F. Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *Proceedings of 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–191, 2017.

23 T. Fischer, A. V. Goldberg, D. J. Haglin, and S. Plotkin. Approximating matchings in parallel. *Information Processing Letters*, 46(3):115–118, 1993.

24 M. Ghaffari, D. G. Harris, and F. Kuhn. On derandomizing local distributed algorithms. In *Proc. 59th Symp. on Foundations of Computer Science (FOCS)*, pages 662–673, 2018.

25 M. Ghaffari and J. Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proc. 30th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1636–1653, 2019.

26 M. Hańćkowiak, M. Karoński, and A. Panconesi. On the distributed complexity of computing maximal matchings. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 219–225, 1998.

27 M. Hańćkowiak, M. Karoński, and A. Panconesi. A faster distributed algorithm for computing maximal matchings deterministically. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 219–228, 1999.

28 J.H. Hoepman, S. Kutten, and Z. Lotker. Efficient distributed weighted matchings on trees. In *Proceedings of 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 115–129, 2006.

29 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 1973.

30 A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Process. Lett.*, 22(2):77–80, 1986.

31 R. M. Karp, E. Upfal, and A. Widgerson. Constructing a perfect matching is in random NC. In *Proc. 17th ACM Symp. on Theory of Computing (STOC)*, pages 22–32, 1985.

32 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd ACM Symp. on Theory of Computing (STOC)*, pages 352–358, 1990.

33 L. Kor, A. Korman, and D. Peleg. Tight Bounds For Distributed MST Verification. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS11)*, 2011.

34 F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *J. of the ACM*, 63(2), 2016.

35 N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

36 Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 129–136, 2008.

37 Z. Lotker, B. Patt-Shamir, and A. Rosén. Distributed approximate matching. *SIAM Journal on Computing*, 39(2):445–460, 2009.

38 L. Lovász and M.D. Plummer. *Matching Theory*. North-Holand, 1986.

39 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

40 Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273, 2013.

41 A. McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181, 2005.

42 T. Nieberg. Local, distributed weighted matching on general and wireless topologies. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, pages 87–92, 2008.

43 A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.

44 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

45 V. V. Vazirani. A simplification of the mv matching algorithm and its proof. *CoRR*, abs/1210.4594, 2012.

46 M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. In *Proc. 18th Conf. on Distributed Computing (DISC)*, pages 335–348, 2004.

47 Y. Yoshida, M. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proc. 41st ACM Symp. on Theory of Computing (STOC)*, pages 225–234, 2009.