Improved Distributed Approximate Matching

ZVI LOTKER, Ben Gurion University
BOAZ PATT-SHAMIR, Tel Aviv University
SETH PETTIE, University of Michigan

We present distributed network algorithms to compute weighted and unweighted matchings with improved approximation ratios and running times. The computational model is a network of processors exchanging $O(\log n)$-bit messages (the CONGEST model). For unweighted graphs, we give an algorithm providing $(1-\epsilon)$-approximation in $O(\log n)$ time for any constant $\epsilon > 0$, improving on the classical $\frac{1}{2}$-approximation in $O \log n)$ time of Israeli and Itai [1986]. The time complexity of the algorithm depends on $\frac{1}{\epsilon}$ exponentially in the general case, and polynomially in bipartite graphs. For weighted graphs, we present another algorithm which provides $(\frac{1}{2} - \epsilon)$ approximation in general graphs in $O(\log \epsilon^{-1} \log n)$ time, improving on the previously known algorithms which attain $(\frac{1}{4} - \epsilon)$-approximation in $O(\log n)$ time or $\frac{1}{2}$-approximation in $O(n)$ time. All our algorithms are randomized: the complexity bounds hold both with high probability and for the expected running time.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: CONGEST model, network algorithms

ACM Reference Format:
Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2015. Improved distributed approximate matching. J. ACM 62, 5, Article 38 (October 2015), 17 pages.
DOI: http://dx.doi.org/10.1145/2786753

1. INTRODUCTION

Computing a set of disjoint edges in a graph, also called a *matching*, is one of the fundamental tasks in combinatorial optimization and in distributed systems. Matching computation routines are often used as building blocks in complex distributed algorithms, and a matching may be the target application in its own right. As a concrete example, consider a communication switch (like an Internet router), whose basic task is to transfer packets from input ports to output ports through an internal network called the "switch fabric" (see Figure 1). In most switch architectures, the switch fabric can deliver in each cycle at most one packet from each input and at most one packet to each output port, and an internal scheduling routine decides which ports will be connected in each cycle. The scheduling routine tries to maximize throughput, which

A preliminary version of this article appeared in Lotker et al. [2008].
This work is supported by the Israel Science Foundation. S. Pettie is supported by NSF grants CCF-1217338 and CNS-1318294 and a grant from the US-Israel Binational Science Foundation.
Authors' addresses: Z. Lotker (correspondig author), Department of Communication Systems Engineering, Ben Gurion University; email: zvilo@cse.bgu.ac.il; B. Patt-Shamir, School of Electrical Engineering, Tel Aviv University; S. Pettie, Department of Electrical Engineering and Computer Science, University of Michigan.

DOI: http://dx.doi.org/10.1145/2786753

Fig. 1. Schematic switch structure. The fabric can realize a matching between the input ports and the output ports.

is usually interpreted as finding the largest possible matching between the input ports (see, e.g., [McKeown 1999]).

One significant generalization of the matching problem is the case where edges of the graph have intrinsic positive *weights*, and a matching of the largest possible weight is sought. A simple example for this case is the following scenario. There is a set of different servers and a set of jobs, and for each job, there is some benefit to be gained if the job is executed on one of a given subset of the servers. Assuming that each server can execute at most one job, maximizing the total gain is equivalent to computing a maximal weight matching.

Note that in both the switch and the job-server examples, as well as in many other important applications, the underlying graph for the matching problem is bipartite, that is, the nodes can be partitioned into two subsets such that all edges have one endpoint in each subset.

A Brief History of Distributed Matching. Due to its prominent role, much research has been invested in developing efficient and effective algorithms for maximum matching, in bipartite and general graphs and in the weighted and unweighted. Let us first try to quickly summarize the situation from the distributed algorithm point of view (for a summary of results in the sequential settings, see the following). First we note that finding an optimal matching distributively, even for unweighted graphs, may require time proportional to the diameter of the graph regardless of the message size.[1]

As a result, most research focuses on fast approximation algorithms. To discuss approximations, let us introduce some notation first. For $0 \leq \delta \leq 1$, we use δ-MCM to denote a matching that is a δ-approximation to the maximum *cardinality* matching (i.e., whose size is at least a δ fraction of the size of the largest possible matching for that graph). Similarly, we use δ-MWM to denote a *weighted* matching whose total weight is at least a δ fraction of the best possible.

It is straightforward to observe that the greedy algorithm (that repeatedly adds the heaviest remaining edge to the matching and removes all its incident edges from the graph) finds a $\frac{1}{2}$-MWM (or $\frac{1}{2}$-MCM if the graph is unweighted). Even such seemingly trivial guarantees are not very easy to obtain in the distributed setting (unless one is willing to pay time proportional to the number of edges [Hoepman 2004]). The basic result in this area is the elegant algorithm of Israeli and Itai [1986], called II henceforth. II is a randomized algorithm in the CONGEST model (i.e., using small messages, see Section 2) which is guaranteed to produce a $\frac{1}{2}$-MCM in $O(\log n)$ time in n-nodes graphs. (The running time is both expected and holds with high probability.) Ideas from II are the basis of the PIM algorithm used in the high-speed AN2 switch of DEC [Anderson et al. 1993]. PIM was later refined to the deterministic iSLIP algorithm [McKeown 1999], which is the algorithm of choice in many of today's routers. From the worst-case complexity viewpoint, PIM and iSLIP algorithms are no better than II. To date, II has

[1]To see that, consider C_{2n}, that is, a ring even size. There are only two maximum matchings in C_{2n} (either all even edges or all odd edges). Intuitively, computing a maximum matching requires global coordination that cannot be achieved in sublinear time. For a formal argument, note that finding a maximum matching in C_{2n} is equivalent to 2-coloring the nodes of C_{2n} (because the line graph of C_{2n} is C_{2n}), which cannot be done in less than $n-1$ time units [Linial 1992], Theorem 2.2.

not been completely subsumed. Barenboim et al. [2012] gave a randomized maximal matching algorithm in the CONGEST model running in $O(\log \Delta + (\log \log n)^4)$ time with high probability, which has optimal dependence on Δ [Kuhn et al. 2006]. Building on Hańćkowiak et al. [1999], a deterministic algorithm that computes a $\frac{2}{3}$-MCM using large messages in $O(\log^4 n)$ time is presented by Czygrinow et al. [2004]. Czygrinow and Hańćkowiak [2003], it is shown that in bipartite graphs, a $(1-\epsilon)$-MCM can be computed deterministically in $(\log n)^{O(1/\epsilon)}$ time, again using large messages.

For weighted graphs, Lotker et al. [2009] give a randomized algorithm for $(\frac{1}{4}-\epsilon)$-MWM using $O(\log n)$ time (expected and with high probability), and based on the approach of Wattenhofer and Wattenhofer [2004], they explain how to get $(\frac{1}{2}-\epsilon)$-MWM in $O(\log^2 n)$ time. In the special case of trees, it is shown in Hoepman et al. [2006] that a $(\frac{1}{2}-\epsilon)$-MCM can be computed in expected constant time. Concurrently with the preliminary version of this work, Nieberg [2008] presented an algorithm to compute $(1-\epsilon)$-MWM in $O(\log^2 n)$ time using large messages.

On the negative side, it is known that any distributed algorithm, randomized or deterministic, requires $\Omega(\sqrt{\log n})$ (expected) time to compute a $\Theta(1)$-approximate maximum cardinality matching [Kuhn et al. 2006]. The lower bound holds even if messages have unbounded size, so long as messages may progress only one hop at a time step.

Our Results. In this article, we give substantially improved distributed algorithms for approximate maximum matchings. Building on some known techniques [Hopcroft and Karp 1973; Fischer et al. 1993; Pettie and Sanders 2004; Lotker et al. 2009], and adding a few new ideas, we obtain the following results. For unweighted graphs and any constant $\epsilon > 0$, we present the first $(1-\epsilon)$-MCM in $O(\log n)$ time using messages of $O(\log n)$ bits. (All our algorithms are randomized, and their stated running time is with high probability.) Our derivation consists of three steps. First, we describe a generic algorithm that requires, in the general case, messages of linear size. We then show how to implement the algorithm in the special case of bipartite graphs using messages of $O(\log n)$ bits, and the resulting time complexity is $O(\epsilon^{-3} \log n)$. Finally, we give a randomized reduction from general graphs to bipartite graphs. The running time of the last algorithm is $2^{O(1/\epsilon)} \log n$. For general *weighted* graphs, we give a $(\frac{1}{2}-\epsilon)$-MWM algorithm whose running time is $O(\log n)$ using $O(\log n)$-bit messages. The idea is to show that if a δ-MWM can be computed in time T for some constant $\delta > 0$, then it is possible to compute a $(\frac{1}{2}-\epsilon)$-MWM in time $O(T \log \frac{1}{\epsilon})$. Using the $(\frac{1}{4}-\epsilon)$-MWM algorithm of Lotker et al. [2009], we obtain the result.

More Related Work. Maximum matching is a classic optimization problem that is the target of extensive research for both the bipartite and general case. We refer the reader to Lovász and Plummer [1986] for a comprehensive account on matching theory, and to more recent [Mucha and Sankowski 2004; Harvey 2006; Cygan et al. 2012; Vazirani 2012; Duan et al. 2015] on the problem. The best sequential algorithm for approximate MWM guarantees $(1-\epsilon)$-MWM in time $O(\epsilon^{-1} \log \epsilon^{-1} m)$, where m is the number of edges in the graph [Duan and Pettie 2014]. Refer to Drake and Hougardy [2003] and Pettie and Sanders [2004] for earlier linear-time sequential algorithms for computing $(\frac{2}{3}-\epsilon)$-MWM.

Matching is also studied in the context of parallel computation. For example, Karp et al. [1985] give a randomized NC algorithm for maximum cardinality matching, and Fischer et al. [1993] give a PRAM algorithm for computing approximate maximum cardinality matchings. In their algorithm, $n^{\Theta(1/\epsilon)}$ processors produce a $(1-\epsilon)$-MCM in $O(\log^3 n)$ time. In Hougardy and Vinkemeier [2006], the algorithm from Fischer et al.

[1993] is extended to a $(1-\epsilon)$-MWM PRAM algorithm, with similar processor and time bounds.

We conclude this brief survey with a few results that were published following the publication of the preliminary version of this work [Lotker et al. 2008]. One direction was to extends matching to a more general edge packing problem called c-matching, where the goal is to select subsets of the edges with the greatest possible size or weight, subject to capacity constraints in the nodes (in a simple matching, the constraint is that each node touches at most one selected edge). This problem is treated (with a general framework for packing and covering) in Koufogiannakis and Young [2011], where an $O(\log n)$-time distributed algorithm with approximation ratio $\frac{1}{2}$ is presented.

Our algorithm is used as a building block in some distributed applications. One example is Patt-Shamir et al. [2012], where our matching algorithm serves as a key component in a distributed procedure that finds an assignment of mobile nodes to base stations in 4G cellular networks.

Another interesting new research direction is that of *local computation algorithms abbreviated* (LCAs) [Rubinfeld et al. 2011]. In the context of the matching problem, an LCA is an algorithm which consistently answers queries as to whether a given edge belongs to some (fixed, unknown) approximate matching. The twist is that the sequential time complexity of processing any query should be sublinear, so computing the complete matching is infeasible. LCAs are relevant to our work due to the observation that distributed algorithms can be transformed into sublinear-time algorithms [Parnas and Ron 2007]. And indeed, the matching LCAs [Mansour and Vardi 2013; Even et al. 2015] are based in part on our algorithm.

Organization. In Section 2, we describe the model and introduce some notation. In Section 3, we give our $(1-\epsilon)$-MCM algorithms. Section 4 covers our $(\frac{1}{2}-\epsilon)$-MWM algorithm. We conclude with some open problems in Section 5.

2. PRELIMINARIES AND NOTATION

Computational Model. Throughout this article, we assume the standard synchronous network model, where in each time step, processors send (possibly different) messages to neighbors, receive messages from neighbors, and perform some local computation. We consider both the LOCAL model where message size may be unbounded, and the CONGEST($\log n$) model where messages contain $O(\log n)$ bits. We assume that nodes have unique identifiers of $O(\log n)$ bits. See Peleg [2000] for more details.[2]

Graphs, Matching Etc. The input is an undirected graph $G = (V, E)$, which needs not be simple. If the graph is *weighted*, there is also a weight function $w : E \to \mathbb{R}^+$. If the graph is *unweighted*, we assume an implicit weight function $w(e) = 1$ for all $e \in E$. We use n to denote $|V|$, $\deg(v)$ to denote the degree (number of incident edges) of a node $v \in V$, and Δ to denote the maximal node degree in G.

A *matching* $M \subseteq E$ is a subset of the edges such that no two edges in M share an endpoint. Throughout the article, we use M to denote a matching in general, and M^* to denote the matching of maximum cardinality (or weight). A vertex is said to be *free with respect to* M if it is not incident with any edge in M, and *matched* otherwise. An *augmenting path with respect to* M is a simple path whose endpoints are free with respect to M and whose edges alternate between $E \backslash M$ and M (we usually omit the qualification "w.r.t. M" when it is clear from the context). Given an augmenting path

[2]Within these models, the assumption that the network is synchronous can be made without loss of generality (using, say, the α synchronizer of [Awerbuch 1985]), because we do not consider faults and because we are not interested in the message complexity of algorithms.

ALGORITHM 1: Abstract Algorithm (The input is a graph $G = (V, E)$ and $\epsilon > 0$)

1: $M \leftarrow \emptyset$ ▷*M ranges over sets of edges*
2: $k \leftarrow \lceil 1/\epsilon \rceil$
3: **for** $\ell \leftarrow 1, 3, \ldots, 2k-1$ **do**
4: Construct the conflict graph $C_M(\ell)$ ▷*See Def. 3.1*
5: Compute an MIS $\mathcal{I}$ of $C_M(\ell)$
6: Let $\mathcal{P}$ be the union the of augmenting paths corresponding to $\mathcal{I}$
7: $M \leftarrow M \oplus \mathcal{P}$
8: **end for**
9: Output M ▷*M is a $(1 - \frac{1}{k+1})$-approximate MCM*

A, we will abuse notation slightly and use A also to denote the set of edges in A. For sets A and B, we denote $A \oplus B \stackrel{\text{def}}{=} (A \cup B) \backslash (A \cap B)$.

Distributed Matching. As is customary in the distributed model, we assume that the input graph is also the underlying computational platform. The output is distributed in the following sense. Each node maintains an output register which either points to an incident edge (this is a matching edge) or points to NULL if that node is unmatched. In the case of weighted graphs, we assume that every node knows the weights of all its incident edges. We also assume that all nodes have a common upper bound on the maximal identifier value $W_{\max}$, such that $\log W_{\max} = O(\log n)$. Finally, we assume, for ϵ-dependent approximations, that all nodes know the value of ϵ.

General Conventions. All logarithms in this article are to base 2. We use the phrase "with high probability," abbreviated "w.h.p.," as a synonym of "with probability $1 - n^{-c}$, for any desired constant $c > 0$."

3. UNWEIGHTED MATCHINGS

In this section, we present a $(1-\epsilon)$-approximation for maximum cardinality matching, which runs in $O(\log n)$ time for any constant $\epsilon > 0$ and uses messages of $O(\log n)$ bits. Our development consists of three steps: in Section 3.1, we give a generic algorithm that requires messages whose size may be linear in the graph size; in Section 3.2, we show how to reduce the message size to $O(\log n)$ bits in the case of bipartite graphs, and finally, in Section 3.3, we give a randomized reduction of general graphs to bipartite graphs.

3.1. Generic Algorithm

In this section, we present a generic distributed matching algorithm which uses large messages and runs in time polynomial in ϵ^{-1} and $\log n$. In the following sections, we show how to implement it using small messages.

The basic idea of the algorithm is the following. We proceed in phases, where in phase ℓ, for $\ell = 1, \ldots, \lceil \frac{1}{\epsilon} \rceil$, we identify a *maximal* set of augmenting paths of length $2\ell - 1$; we augment the matching from phase $\ell - 1$ along these paths and obtain the matching of phase ℓ. The key to the analysis of the algorithm is that the matching obtained in phase ℓ is a $(1 - \frac{1}{\ell+1})$-approximation to the maximum matching. We note that a similar idea, for the PRAM model, appeared in Fischer et al. [1993]. The only part of our generic algorithm which is slightly less standard is how to distributively identify a maximal set of nonconflicting augmenting paths. We implement this part by constructing a certain "conflict graph" and emulating an execution of a maximal independent set (MIS) algorithm on that graph.

To specify the algorithm in detail, we formalize the concept of a conflict graph.

Definition 3.1. Let $G = (V, E)$ be a graph, let $M \subseteq E$ be a matching, and let $\ell > 0$ be an integer. The *ℓ-conflict graph* w.r.t. M in G, denoted $C_M(\ell)$, is an undirected graph defined as follows. The nodes of $C_M(\ell)$ are all augmenting paths w.r.t. M of length at most ℓ, and two nodes in $C_M(\ell)$ are connected by an edge if and only if their corresponding augmenting paths intersect in G.

Observe that two augmenting paths intersect if and only if they have a common node.

Algorithm 1 contains pseudocode of an abstract algorithm that uses conflict graphs. We note that the algorithm is well-defined in the sense that no two augmentations applied in Step 7 are conflicting: if they were, then their paths intersect, contradicting the independence of their corresponding nodes in the conflict graph. The correctness and approximation ratio guaranteed by Algorithm 1 is a consequence of the following two facts, proved in Hopcroft and Karp [1973] (see also Karzanov [1973]).

LEMMA 3.2. *If the shortest augmenting path w.r.t. M has length ℓ and P is a maximal set of augmenting paths of length ℓ, then the shortest augmenting path w.r.t. $M \oplus P$ has length strictly greater than ℓ.*

LEMMA 3.3. *If the shortest augmenting path w.r.t. M has length $2k - 1 \geq 1$ for some integer $k > 0$, then $|M| \geq (1 - \frac{1}{k})|M^*|$, where $|M^*|$ is the maximum cardinality of a matching in G.*

Next, we explain how to implement the abstract algorithm over the physical graph and analyze its time complexity. The conflict graph construction (Step 1 of Algorithm 1) is implemented by Algorithm 2. The idea there is that each node explores G to distance ℓ, and a deterministic rule assigns each augmenting path to a *leader node*. Exploration is done essentially by flooding the adjacency information for ℓ rounds. Paths are assigned to their endpoint of the smaller ID. When Algorithm 2 terminates, each leader knows of all the intersections its augmenting paths are involved in.

Regarding the complexity of Algorithm 2, we note that since messages of Algorithm 2 contain descriptions of portions of the graph, message size can be bounded by the length of the description of G, namely, $O((|V| + |E|) \log n)$ bits, and we therefore have the following immediate result.

LEMMA 3.4. *Step 4 of Algorithm 1 can be implemented in the LOCAL model in $O(\ell)$ time using $O((|V| + |E|) \log n)$-bit messages.*

After Step 4 of Algorithm 1, each node P in $C_M(\ell)$ (where P is an augmenting path) is associated with a physical node $\text{leader}(P) = v \in V$, and furthermore, each node $v \in V$ which is a leader of an augmenting path knows where its neighbors are in the conflict graph. This information allows for emulating a distributed algorithm of $C_M(\ell)$ over G. More formally, if $v = \text{leader}(P)$, $u = \text{leader}(P')$, and P, P' are neighbors in $C_M(\ell)$, then u and v know of each other, so that when a message needs to be sent in $C_M(\ell)$, say from P to P', that message is routed from v to u over G along P and P', which takes at most 2ℓ steps. Therefore we can emulate any distributed algorithm for $C_M(\ell)$ over G, as stated in the following straightforward lemma.

LEMMA 3.5. *Let $\mathcal{A}$ be a distributed algorithm for the conflict graph $C_M(\ell)$ that runs in time $t(N)$, where N is the number of nodes in $C_M(\ell)$. Then $\mathcal{A}$ can be implemented on G in $O(t(N) \cdot \ell)$ time.*

ALGORITHM 2: Find the Conflict Graph $C_M(\ell)$ (Step 1 of Algorithm 1), Run at Node v

1: In time step $i = 1, \ldots, 2\ell$, v sends to all its neighbors a description of its neighborhood to distance $i-1$ as obtained from incoming messages in the previous step. Let $G_v(\ell)$ and $G_v(2\ell)$ be the local views to distance ℓ and 2ℓ, respectively.
2: Let $\mathcal{P}_v(\ell)$ be the set of all augmenting paths in $G_v(\ell)$, and similarly $\mathcal{P}_v(2\ell)$ in $G_v(2\ell)$ (local computation).
3: For all $P \in \mathcal{P}_v(\ell)$: v assigns leader$(P) \leftarrow v$ if v is an endpoint of P whose ID is smaller than the ID of the other endpoint of P. Similarly, v computes the identity of leaders of paths in $\mathcal{P}_v(2\ell)$.
4: v sends out full path description (node identifiers) along all paths P with $v =$ leader(P).

More specifically, we need the following result.

COROLLARY 3.6. *Step 5 of Algorithm 1 can be implemented, with probability* $1-n^{\Omega(1)}$, *in* $O(\ell^2 \log n)$ *time.*

PROOF. We invoke either the algorithm of Luby [1986] or Alon et al. [1986] to do the MIS computation. The distributed version of these algorithm runs in time $t(N) = O(\log N)$ (w.h.p.), where N is the number of nodes in the graph, and finds an MIS with probability $1-N^{-\Omega(1)}$. In our case, the number of nodes in $C_M(\ell)$ is $N = n^{O(\ell)}$. The result therefore follows from Lemma 3.5. □

We summarize the properties of the generic algorithm as follows.

THEOREM 3.7. *For any given n-node graph* $G = (V, E)$ *and any* $\epsilon \geq 1/n$, *a* $(1-\epsilon)$*-MCM can be computed w.h.p. in the LOCAL model in* $O(\epsilon^{-3} \log n)$ *time.*

PROOF. Let us first bound the probability that the algorithm fails due to the MIS computation (which is the only randomized step in Algorithm 1). By Corollary 3.6, for any given constant c_1, there exists a constant c_2 such that if we implement Step 5 in $c_2 \log n$ steps, the probability that it fails is at most n^{-c_1}. Since Step 5 is executed $O(1/\epsilon)$ times, the total probability of failure is at most $O(n^{-c_1}/\epsilon)$ by the union bound. Since $\epsilon \geq 1/n$, w.h.p. the algorithm does not fail.

Now, assuming that all MIS computations succeed, we prove the correctness of the algorithm by induction on the number of iterations of the main loop (lines 3–8 in Algorithm 1). We claim that before iteration $2\ell - 1$, the length of the shortest augmenting path in G is at least $2\ell - 1$. The base case, $\ell = 1$, is trivial. The inductive step follows from Lemma 3.2 and the maximality of the MIS computed in Step 5 of Algorithm 1. Therefore, by Lemma 3.3, when the algorithm terminates, the size of M is at least $(1 - 1/k) \geq (1 - \epsilon)$ times the optimum.

Finally, we consider time complexity. Each execution of Step 5 takes $O(1/\epsilon)$ time by Lemma 3.4 and each execution of Step 5 takes $O(\epsilon^{-2} \log n)$ time by Corollary 3.6. Step 7 is implemented distributively by letting leader nodes send messages along the paths chosen to the MIS in Step 5, which takes $O(\ell)$ time. Therefore an execution of an iteration is completed in $O(\epsilon^{-2} \log n)$ time, and the time bound follows from the fact that there are $O(\epsilon^{-1})$ iterations. □

3.2. Bipartite Graphs

In this section, we show how to reduce the message size to $O(\text{poly}(\epsilon^{-1}) \log n)$ in the case of bipartite graphs. There are two ideas underlying our algorithm.

The first idea, due to Hopcroft and Karp [1973], is that given a bipartite graph $G = (X, Y, E)$ and a matching $M \subseteq E$, it is easy to find augmenting paths by orienting all matching edges from Y to X and all edges in $E \backslash M$ from X to Y. This way, any directed path from a free X node to a free Y node is an augmenting path, and finding

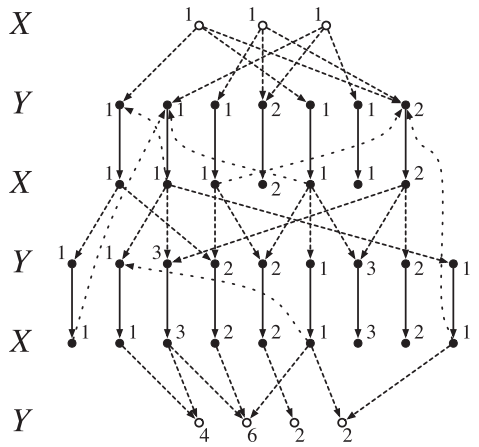

Fig. 2. Illustration of Algorithm 3. Hollow and filled circles represent free and matched vertices, resp. Dashed and solid arcs represent nonmatching and matching edges, resp. The algorithm starts at the top layer and progresses one layer at a time. Numbers next to nodes are the sum of numbers received from the previous level.

the shortest ones is easily done by a breadth-first search. (In the case of general graphs orienting the edges does not achieve this effect.)

The second idea is how to select a maximal set of shortest augmenting paths efficiently in a distributed way. Implementing Steps 4 and 5 of Algorithm 1 directly is expensive. Our approach is to combine these steps by constructing the conflict graph "on the fly," while performing the breadth-first search for the shortest augmenting paths. This results in a dramatic improvement in the complexity due an simple insight into the way the MIS algorithm works, which implies that it is sufficient for each node to know only the number of augmenting paths it is a member of. Briefly, the MIS algorithm adds a node v to the MIS with probability $1/(\deg(v) + 1)$; our algorithm mimics this random experiment by counting, for each node (in our case, for each augmenting path), its number of neighbors.

More specifically, the implementation is as follows (see Figure 2 for an example, and Algorithm 3 for pseudocode). Suppose that we are given a bipartite graph $G = (X, Y, E)$, where X and Y are the node sets and E is the set of edges connecting them. We count the number of augmenting paths of length ℓ by initiating a breadth-first search (BFS) from all free X nodes simultaneously. As the BFS progresses, each edge records the number of partial augmenting paths it leads to. A node sends only one message, immediately after the first round in which it received messages. All messages received later are discarded (these are the back arrows in Figure 2). The algorithm starts with each unmatched X-node sending 1 to all neighbors. Thereafter, when a node receives messages (i.e., numbers arriving from edges) for the first time, it records the received messages, and sums their numbers up. A receiving Y node sends the sum it obtained only if it is matched, and only to its mate. (A message arriving before round ℓ to an unmatched Y node indicates an augmenting path of length less than ℓ.) A receiving X node, which is necessarily matched (the message has arrived from its mate), sends the received number to all its unmatched neighbors. Messages arriving to visited Y nodes indicate an augmenting path that intersects a shorter one. This forwarding procedure is executed ℓ rounds. The last round is slightly different: X nodes send their value to all their neighbors as usual, but in this case, we care only for the values recorded by free Y nodes.

ALGORITHM 3: At Node v: Counting Half-Augmenting Paths of Length ℓ in Bipartite Graphs (*Precondition: there are no augmenting paths of length less than* ℓ)

1: $c_v[i] \leftarrow 0$ for all $1 \leq i \leq \deg(v)$ $\triangleright c_v[i]$ *is the number of paths from edge* i *at* v
2: **if** $v \in X$ **and** v is free **then**
3: send 1 to all neighbors and **halt**
4: **end if**
5: **wait** until a message is received; let $t(v)$ denote the message arrival time
6: $c_v[i] \leftarrow$ the number received on edge i
7: $n_v \leftarrow \sum_{i=1}^{\deg(v)} c_v[i]$
8: **if** $v \in X$ **then**
9: send n_v to all neighbors and **halt**
10: **end if**
11: **if** $v \in Y$, v is matched to a node u, and time $< \ell$ **then**
12: send n_v to u and **halt**
13: **end if**

Call a sequence of edges a *half-augmenting path* if it starts with a free X node and alternates between nonmatching and matching edges, with no restrictions on the last edge. Let $d(v)$ be the minimal length of any half-augmenting path which ends at v.

LEMMA 3.8. *Suppose that G and M are such that there are no augmenting paths shorter than ℓ. Consider a node v in the execution of Algorithm 3. Let n_v be as computed in line 7. Then there are n_v half-augmenting paths of length $d(v)$ that end at v. Furthermore, $n_v \leq \Delta^{\lceil d(v)/2 \rceil}$.*

PROOF. Let $t(v)$ be as defined in line 5. We prove, by induction on $t(v)$, that $t(v) = d(v)$ and that there are n_v shortest half-augmenting paths of length $t(v)$ that end at v. The base case is when $t(v) = 0$. In this case, v is a free X node. The assertion then follows from lines 2–3 of the code. For the inductive step, let $t(v) = t + 1$, and suppose that the hypothesis holds for $t(v) = t$. We distinguish between the cases where t is even and t odd.

Suppose first that t is odd. Then v (that receives a first message at time $t + 1$) is an X node. At time t, only Y nodes send messages, and these messages are sent only along matching edges. Suppose that node $u \in Y$ is matched to v. Then by definition, the number of half-augmenting paths of length $t+1$ that end with v is equal to the number of half-augmenting paths of length t that end with u. Since v, in this case, receives in round $t + 1$ only a message from its mate u, the inductive claim holds for $t + 1$.

Suppose now that t is even. Then v is a Y node. At time t only X nodes send messages. By induction, the message received by v at time $t + 1$ from any neighbor u is exactly the number of half-augmenting paths of length t that end with u. Since the prefix of length t of each half-augmenting path of length $t + 1$ that ends with v must be a half-augmenting path of length t that ends with some neighbor u of v, the assignments of lines 6–7 guarantee that the inductive claim holds in this case as well.

Clearly, messages arriving after time t correspond to non-shortest augmenting paths, and thus need not be counted.

Finally, we note that $n_v \leq \Delta^{\lceil d(v)/2 \rceil}$ because the maximum number sent at time $t + 2$ is at most Δ times the maximum sent at time t. □

It follows from Lemma 3.8 that if there are no augmenting paths of length less than ℓ, then, for every unmatched node $y \in Y$, after ℓ time steps n_y is the number of augmenting paths ending at y.

Computing a Maximal Set of Augmenting Paths. We adapt the MIS algorithm of Luby [1986] to our setting. The algorithm works in iterations as follows. In each iteration,

each node in the conflict graph (representing an augmenting path of length ℓ) chooses a random number, and that node is added to the MIS if and only if its number is larger than all numbers chosen by its neighbors. The numbers are chosen uniformly at random from $[1, N^4]$, where N is the number of nodes in the conflict graph., which in our case is bounded by $n\Delta^{(\ell+1)/2}$. (If n and Δ are not known to the nodes, we can use $W_{\max}$, the maximal possible node identifier value, as a substitute for both: all we need is an upper bound on N.) After $O(\log N)$ iterations of this procedure, an MIS is found with probability $1 - n^{-\Omega(1)}$.

We emulate an iteration distributively as follows. First, we change Algorithm 1 slightly by stipulating that the leader of each augmenting path is its free Y endpoint (we do not need identifiers to break symmetry in the bipartite case). Suppose that node y is the leader of n_y paths (as determined by Algorithm 3). First, instead of each leader selecting a number for each of its paths explicitly, we let each leader y select a single number w_y which is the "winner" of all paths which y leads. This number is chosen according to the probability distribution of the maximum of n_y uniform random variables over $\{1, \ldots, N^4\}$. Since the probability that such a variable gets a value at most m is exactly $\frac{m}{N^4}$, it follows that the probability that the maximum of n_y such i.i.d. variables is at most m is given by the following function:

$$\Pr[\max \leq m] = \begin{cases} 0, & \text{if } m < 1, \\ \left(\frac{m}{N^4}\right)^{n_y}, & \text{if } 1 \leq m \leq N^4, \\ 1, & \text{if } m > N^4. \end{cases}$$

Once we determine the winning number, we select the winning path by constructing it stochastically link by link. We work backwards (bottom up in Figure 2) as follows. The first node of the path is the leader node. Now, at any point in the construction, if we are at a node $y \in Y$, the next edge is a nonmatching edge i, chosen randomly with probability $c_y[i]/n_y$ (using the c array computed in Step 6 of Algorithm 3); and if we are at a node $x \in X$, the next edge is the unique matching edge incident with x.

In the algorithm, each leader y sends a token message carrying their chosen number w_y. The path traversed by the token is selected stochastically as previously described, and the path is recorded by the nodes. Whenever tokens meet, only the token carrying the largest number w_y continues with constructing its path, and the other colliding tokens disappear. From the way the paths were constructed, tokens may arrive at a node only at a single round. Finally, when a token arrives at the last node of its path, it traces that path back, and while doing so, it also performs augmentation along that path, that is, it flips the status of matching edges to nonmatching edges and vice-versa.

We can now prove the following.

LEMMA 3.9. *In bipartite graphs with no augmenting paths shorter than ℓ, a maximal independent set of augmenting paths of length ℓ can be found (w.h.p.) in $O(\ell \log N) = O(\ell^2 \log \Delta + \ell \log n)$ time using messages of $O(\log n)$ bits.*

PROOF. Consider the algorithm just describe, where routing the messages is implemented as follows. By Lemma 3.8, the numbers w_y to be routed are $O(\ell \log \Delta) \leq O(\ell \log n)$ bits long, and they need to traverse paths of ℓ hops. To send a number of $j \log n$ bits over an edge, we break it into j *chunks* of $O(\log n)$ bits each, and send the chunks one by one in a pipelined fashion as follows. Initially, all incident edges are marked as *qualifying* to be a possible source of the largest number. The chunks are sent in decreasing order of significance. In each routing step, only chunks from qualifying edges are examined. Of them, only the chunk encoding the largest number is

ALGORITHM 4: Given $G = (V, E)$ and $k > 2$, find w.h.p. a $(1 - \frac{1}{k})$-MCM

1: $M \leftarrow \emptyset$
2: **for** $2^{2k+1}(k+1)\ln k$ iterations **do**
3: Each node independently colors itself red or blue with probability 1/2.
4: Let $\hat{G} \leftarrow (\hat{V}, \hat{E})$, where
$\hat{V} = \{u \mid u \in V$ is free, or $\exists (u, v) \in M$ s.t. (u, v) is bichromatic$\}$, and
$\hat{E} = \{(u, v) \mid (u, v) \in E, u, v \in \hat{V}$ and (u, v) is bichromatic$\}$.
5: $\mathcal{P} \leftarrow \text{Aug}(\hat{G}, M, 2k-1)$ $\triangleright \mathcal{P}$ *is a maximal set of disjoint augmenting paths of length* $\leq 2k-1$
6: $M \leftarrow M \oplus \mathcal{P}$
7: **end for**
8: **return** M

transmitted in the next step, and the sources of other chunks are disqualified. Once a single qualifying edge remains, it is recorded as the source of the maximal value token. This way, an iteration of the MIS algorithm on the conflict graph $C_M(\ell)$ is emulated in $O(\ell)$ steps in G, and therefore an MIS of the conflict graph is found (w.h.p.) in $O(\ell \log N) = O(\ell^2 \log \Delta + \ell \log n)$ time. □

Lemma 3.9 suffices for the next step of our development, as described in Section 3.3. But since bipartite graphs are an important special case in their own right, we state the following result explicitly.

THEOREM 3.10. *In the CONGEST model, a* $(1 - \frac{1}{k})$*-approximation to the maximum matching in bipartite graphs can be found in* $O(k^3 \log \Delta + k^2 \log n)$ *time.*

The proof follows from applying Lemma 3.9 in the context of Algorithm 1.

3.3. Matching in General Graphs Revisited

In this section, we give a randomized reduction of general graphs to bipartite graphs. The idea is to repeatedly sample a bipartite subset of the edges and find a maximal set of short augmenting paths in the resulting bipartite subgraph. Given an integer $k > 2$, after a constant number of iterations of this procedure (depending only on k) we will, with high probability, obtain a $(1 - \frac{1}{k})$-approximation to the maximum cardinality matching. The analysis must now handle the fact that instead of choosing shortest augmentations as before, we now use augmentations that are only "not too long," which means that we cannot apply Lemmas 3.2 and 3.3 directly.

Pseudocode is given in Algorithm 4. The subroutine $\text{Aug}(H, M, \ell)$, takes a matching M in a bipartite graph H and an integer $\ell \geq 1$, and returns a maximal set of disjoint augmenting paths w.r.t. M, each of length at most ℓ.

We start the analysis with two straightforward observations.

OBSERVATION 3.11. *Consider* $\hat{G} = (\hat{V}, \hat{E})$ *as defined in Line 4 of Algorithm 4, and let* $\hat{M} = M \cap \hat{E}$*. If* P *is an augmenting path w.r.t.* $\hat{M}$ *in* $\hat{G}$*, then* P *is also an augmenting path w.r.t.* M *in* G*.*

OBSERVATION 3.12. *If* P *is an augmenting path of length* ℓ *w.r.t.* M *in* G*, then* $\Pr[P \subseteq \hat{E}] = 2^{-\ell}$.

Using Observations 3.11 and 3.12, we now analyze the overall behavior of the algorithm. Recall that M^* denotes an optimal matching in G.

LEMMA 3.13. *Let* $\alpha = (1-\frac{1}{k+1})|M^*|-|M|$. *In Line 5 of Algorithm 4,* $\Pr[|\mathcal{P}| \geq \frac{\alpha}{(k+1)2^{2k}}] = 1 - e^{-\Omega(\alpha)}$.

PROOF. The graph $M \oplus M^*$ consists of a set of paths and cycles whose edges alternate between M and M^*. Let $\mathcal{P}^*$ be the set of augmenting paths in $M \oplus M^*$ with length at most $2k-1$. We claim that $|\mathcal{P}^*| \geq \alpha$. To see this, consider how "far" M is from a $(1-\frac{1}{k+1})$-MCM. By definition, M is exactly α edges short of a $(1-\frac{1}{k+1})$-MCM in G, and hence at least α edges short of a $(1-\frac{1}{k+1})$-MCM in $M \oplus M^*$. However, if $|\mathcal{P}^*| < \alpha$, then one could eliminate all augmenting paths of length at most $2k-1$ by adding less than α edges to the matching, and then, by Lemma 3.3, the resulting matching is a $(1-\frac{1}{k+1})$-MCM in $M \oplus M^*$, a contradiction.

Now, let $\hat{\mathcal{P}}^* \subseteq \mathcal{P}^*$ be those augmenting paths appearing in $\hat{G}$. Since the augmenting paths in $\mathcal{P}^*$ are disjoint, each augmenting path in $\mathcal{P}^*$ is included in $\hat{\mathcal{P}}^*$ independently with probability at least 2^{-2k+1}, and therefore, by the Chernoff Bound (e.g., [Alon and Spencer 2000]), the probability that $|\hat{\mathcal{P}}^*| < |\mathcal{P}^*|/2^{2k}$ (i.e., half its expectation) is $\exp(-\Omega(\alpha))$.

Finally, consider $\mathcal{P}$. The call to $\text{Aug}(\hat{G}, M, 2k-1)$ returns a maximal set $\mathcal{P}$ of nonconflicting augmenting paths in $\hat{G}$. Each augmenting path in $\mathcal{P}$ can intersect at most $k+1$ paths in $\hat{\mathcal{P}}^*$, which implies that $|\mathcal{P}| \geq |\hat{\mathcal{P}}^*|/(k+1)$ and hence $|\mathcal{P}| \geq |\mathcal{P}^*|/((k+1)2^{2k})$ with probability $1-\exp(-\Omega(\alpha))$. □

LEMMA 3.14. *The matching returned by Algorithm 4 is a* $(1-\frac{1}{k})$*-MCM with high probability.*

PROOF. Let δ_i be the difference between $|M|$ and $(1-\frac{1}{k+1})|M^*|$ after i iterations of Line 2. Thus, $\delta_0 = (1-\frac{1}{k+1})|M^*|$. If M is not a $(1-\frac{1}{k})$-MCM after i iterations, then $\delta_i > \frac{1}{k(k+1)}|M^*|$, and then, by Lemma 3.13, $\delta_{i+1} \leq (1-\frac{1}{(k+1)2^{2k}})\delta_i$ with probability $1-\exp(-\Omega(|M^*|))$. Thus, after $2^{2k+1}(k+1)\ln k$ iterations we have, with high probability,

$$\begin{aligned}
\delta_{2^{2k+1}(k+1)\ln k} &\leq \delta_0 \left(1 - \frac{1}{(k+1)2^{2k}}\right)^{2^{2k+1}(k+1)\ln k} \\
&\leq e^{-2\ln k}\left(1 - \frac{1}{k+1}\right)|M^*| \\
&= \frac{1}{k^2}\left(\frac{k}{k+1}\right)|M^*| \\
&= \left(\frac{1}{k} - \frac{1}{k+1}\right)|M^*| \,.
\end{aligned}$$

Thus, the matching M returned by Algorithm 4 is a $(1-\frac{1}{k})$-approximation with probability $1-\exp(-\Omega(|M^*|))$. □

The procedure Aug is implemented by the algorithm of Section 3.2 with the following slight adjustment. In bipartite graphs, we may assume that the shortest augmenting path had a known length ℓ. In the general case, this is no longer true, but this is not a real problem: it only means that a free Y node may receive messages before time ℓ. Lemma 3.13 ensures that w.h.p., no augmenting path longer than $2k-1$ remains when the algorithm terminates.

The complexity of Algorithm 4 is dominated by the call to $\text{Aug}(\hat{G}, M, 2k-1)$, and hence, by repeated application of Lemma 3.9, $O(k^3 \log n)$ time with $O(\log n)$-bits

ALGORITHM 5: Returns a $(\frac{1}{2}-\epsilon)$-MWM of a weighted graph $G=(V,E,w)$

1: $M \leftarrow \emptyset$
2: **for** $(3/2\delta)\ln(2/\epsilon)$ iterations **do**
3: $\quad G' \leftarrow (V, E, w_M)$
4: $\quad M' \leftarrow \delta$-MWM$(G')$ $\quad \triangleright$*invoking a black-box δ-MWM algorithm*
5: $\quad M \leftarrow M \oplus (\bigcup_{e\in M'} \text{wrap}(e))$
6: **end for**
7: **return** M

messages suffice for the implementation. As a corollary, we arrive at our main result of this section.

THEOREM 3.15. *For any graph G and integer $k > 2$, a $(1-\frac{1}{k})$-MCM of G can, with high probability, be computed distributively in $O(2^{2k}k^4(\log k)\log n)$ time using messages of length $O(\log n)$ bits.*

4. WEIGHTED MATCHINGS

In this section, we present an algorithm for $(\frac{1}{2}-\epsilon)$-approximation of weighted matchings running in time $O(\log\frac{1}{\epsilon}\log n)$. While in the unweighted case, we reduced maximum matching to the maximal independent set, in the weighted case, we reduce $(\frac{1}{2}-\epsilon)$-MWM to δ-MWM, for any $\delta > 0$; in other words, we show how to obtain $(\frac{1}{2}-\epsilon)$-weighted matching using a δ-MWM.

Preliminaries. For an edge (u,v) in a matching M, we denote $M(u) \stackrel{\text{def}}{=} v$. For any edge $(r,s) \in E\backslash M$, let wrap(r,s) be the path consisting of the edges $(M(r),r)$, (r,s), and $(s,M(s))$. The length of wrap(r,s) may be one, two, or three, because the edges $(M(r),r)$ and $(s,M(s))$ may or may not exist, but wrap(e) is always defined. Given a set $\mathcal{P}$ of augmenting paths w.r.t. a matching M, define

$$g(\mathcal{P}) \stackrel{\text{def}}{=} w(M \oplus \mathcal{P}) - w(M)\ .$$

Intuitively, $g(\mathcal{P})$ is the resulting *gain* in weight if M was augmented along $\mathcal{P}$.

Algorithm. Our key tool is a new edge-weight function w_M, defined for each edge (u,v) by

$$w_M(u,v) = \begin{cases} g(\text{wrap}(u,v)) & \text{for } (u,v) \notin M, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, $w_M(u,v)$ is the gain in weight of M if we modify M by adding (u,v) and deleting edges incident to u and v (if such edges are in M). See Figure 3 for an example.

The idea of our approximate MWM algorithm is to apply a set of augmenting paths of length 3. Since different augmenting paths may conflict with each other, we select paths using a maximal-weight matching algorithm (see Algorithm 5). In each iteration, Algorithm 5 is given a matching M (starting with the empty matching). The algorithm finds a δ-MWM in the graph G modified to have edge weights defined by w_M, obtaining a matching M'. M is then augmented by all augmenting paths of length 3 centered at the edges of M', and the result fed into the next iteration.

Analysis. We first prove that after each iteration, M is a matching of increased weight. The idea is that we may add an edge e to the matching only if its weight is larger than the weight of matching edges e is adjacent to (those edges will be thrown out of the matching if we add e).

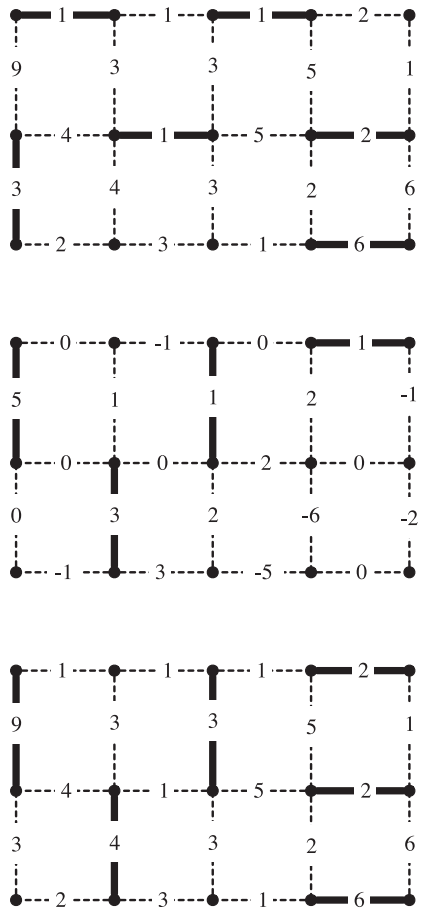

Fig. 3. Top: a matching M (bold edges matched, dashed edges unmatched) with weight 14 under the original weight function w. Middle: a matching M' with weight 10 under the weight function w_M. Bottom: the matching $M'' = M \oplus \bigcup_{e \in M'} \text{wrap}(e)$, having weight $w(M'') = 26 \geq w(M) + w_M(M')$.

LEMMA 4.1. *Let M and M' be two disjoint matchings, and let $M'' \stackrel{\text{def}}{=} M \oplus \bigcup_{e \in M'} \text{wrap}(e)$ (note that $\text{wrap}(e)$ is w.r.t M). Then M'' is also a matching, and furthermore, $w(M'') \geq w(M) + w_M(M')$.*

PROOF. We first show that M'' is a matching by contradiction. If M'' is not a matching then it must contain two adjacent edges $f \in M$ and $e \in \text{wrap}(e')$ for some $e' \in M'$. Since $e \notin M$, it must be the case that $e \in M'$ and then $f \in \text{wrap}(e)$, a contradiction to $f \in M''$. Turning to the second part, we have the following inequalities:

$$\begin{aligned} w(M'') - w(M) &= g\left(\bigcup_{e \in M'} \text{wrap}(e)\right) \\ &\geq \sum_{e \in M'} g(\text{wrap}(e)) = w_M(M'). \end{aligned}$$

The first and last equality follow immediately from the definitions. Notice that the short augmenting paths in $\bigcup_{e\in M'} \text{wrap}(e)$ could overlap, but only at M edges. Thus, adding the individual gains in $\sum_{e\in M'} g(\text{wrap}(e))$ is, if anything, an underestimate of $g\left(\bigcup_{e\in M'} \text{wrap}(e)\right)$. See Figure 3 for an example. □

We use the following fact from Pettie and Sanders [2004] (recall that M^* denotes an optimal matching).

LEMMA 4.2. *For all $k > 0$, there exists a collection $\mathcal{P}$ of* disjoint *augmenting paths and cycles, each having no more than k unmatched edges, such that $w(M \oplus \mathcal{P}) \geq w(M) + \frac{k+1}{2k+1}(\frac{k}{k+1} w(M^*) - w(M))$.*

Using Lemma 4.2 with $k = 2$, we obtain the following.

LEMMA 4.3. *Let M_i be the matching after i iterations. Then, $w(M_i) \geq \frac{1}{2}(1 - e^{-2\delta i/3}) \cdot w(M^*)$.*

PROOF. Consider iteration i. By Lemma 4.2, there exists a set $\mathcal{P}$ of vertex-disjoint augmentations (each with one unmatched edge) such that $g(\mathcal{P}) \geq \frac{2}{3}(\frac{1}{2} w(M^*) - w(M_{i-1}))$. By the definition of $w_{M_{i-1}}$ and the disjointness of the augmentations in $\mathcal{P}$, it follows that $w_{M_{i-1}}(\mathcal{P} \setminus M_{i-1}) = g(\mathcal{P})$. Lemmas 4.2 and 4.1 together imply that

$$\begin{aligned} w(M_i) &= w\left(M_{i-1} \oplus \bigcup_{e\in M'} \text{wrap}(e)\right) \\ &\geq w(M_{i-1}) + \frac{2\delta}{3}\left(\frac{1}{2} w(M^*) - w(M_{i-1})\right). \end{aligned}$$

Applying the argument inductively, we obtain that $w(M_i) \geq \frac{1}{2}(1 - (1 - \frac{2\delta}{3})^i) \cdot w(M^*)$. □

We also use the following fact from Lotker et al. [2009].

LEMMA 4.4. *In the CONGEST model, $\frac{1}{5}$-MWM can be computed in $O(\log n)$ time (w.h.p.).*

Now we conclude with the following theorem.

THEOREM 4.5. *In the CONGEST model, for any $\epsilon > 0$, $(\frac{1}{2} - \epsilon)$-MWM can be computed in $O(\log \epsilon^{-1} \log n)$ time (w.h.p.).*

PROOF. We use the algorithm of Lotker et al. [2009] in Line 4 with $\delta = 1/5$. Since in constant time we can find $\text{wrap}(e)$ for any edge e and apply augmentation, it follows from Lemma 4.4 that each iteration of Algorithm 5 takes $O(\log n)$ time. Applying Lemma 4.3 with $i = \frac{3}{2\delta} \cdot \ln \frac{2}{\epsilon}$ yields the result. □

We note that our algorithm cannot yield approximation better than 1/2 in general, as can be seen by a graph with three unit-weight edges in series, and the matching that consists of the middle edge. The gain of all edges is 0 in this case.

Remark. A $(1 - \epsilon)$-MWM can be obtained in $O(\epsilon^{-4} \log^2 n)$ time, using messages of linear size, by adapting the PRAM algorithm of Hougardy and Vinkemeier [2006] to the distributed setting using Algorithm 2. Since a similar result was reported independently in Nieberg [2008], we give here only a high-level overview of the algorithm. The idea is as follows. Using Algorithm 2, we look at all augmentations of length $O(\frac{1}{\epsilon})$ and calculate for each its "gain" (similar to the w_M weight). The augmentations are then partitioned into classes, where the gain of augmentations in class i is at least 2^{i-1} and less then 2^i. Then, an MIS algorithm is run repeatedly over the conflict graph, taking

into account only nodes (i.e., augmentations) of the highest remaining class: after each MIS invocation, the resulting node set and its neighbors are removed from the conflict graph and the process is repeated. After invoking the MIS algorithm on the top $O(\log n)$ classes, the matching is extended along all augmentations selected. In Hougardy and Vinkemeier [2006], it is shown that repeating this procedure $O(\frac{1}{\epsilon})$ times results in a $(1-\epsilon)$-MWM. The time complexity, using Lemma 3.5, is $O(\epsilon^{-4}\log^2 n)$, but messages may be $O((|V|+|E|)\log n)$ bits long.

5. CONCLUSION

In this article, we have greatly improved the quality of approximate (weighted and unweighted) matchings that can be computed by distributed algorithms. There is clearly still plenty of room for improvement in the weighted case: it may yet be possible to obtain weighted matchings that are arbitrarily close to optimal using small messages. For unweighted graphs, it is interesting to see whether there exists a $(1-\epsilon)$-approximation for general graphs, using small messages, with time complexity polynomial in $1/\epsilon$ and $\log n$. And of course, the long-standing open question: can maximal matching, or independent set, be computed deterministically in $O(\log n)$ time on general graphs?

REFERENCES

Noga Alon, Laszlo Babai, and Alon Itai. 1986. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms* 7 (1986), 567–583.

Noga Alon and Joel H. Spencer. 2000. *The Probabilistic Method* (2nd Ed.). Wiley.

Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. 1993. High speed switch scheduling for local area networks. *ACM Trans. Comput. Syst.* 11, 4 (Nov. 1993), 319–352.

Baruch Awerbuch. 1985. Complexity of network synchronization. *J. ACM* 32, 4 (Oct. 1985), 804–823.

Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2012. The locality of distributed symmetry breaking. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 321–330. (Full version available at arXiv:1202.) 1983v3.

Marek Cygan, Harold N. Gabow, and Piotr Sankowski. 2012. Algorithmic applications of Baur-Strassen's theorem: Shortest cycles, diameter and matchings. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 531–540.

Andrzej Czygrinow and Michał Hańćkowiak. 2003. Distributed algorithm for better approximation of the maximum matching. In *Proceedings of the 9th Annual on Computing and Combinatorics (COCOON)*. 242–251.

Andrzej Czygrinow, Michał Hańćkowiak, and Edyta Szymańska. 2004. A fast distributed algorithm for approximating the maximum matching. In *Proceedings of the 12th Annual European Symposium on Algorithms*. 252–263.

Doratha E. Drake and Stefan Hougardy. 2003. Improved linear time approximation algorithms for weighted matchings. In *Proceedings of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX)*. 14–23.

Ran Duan and Seth Pettie. 2014. Linear-time approximation for maximum weight matching. *J. ACM* 61, 1 (2014), 1. DOI: http://dx.doi.org/10.1145/2529989

Ran Duan, Seth Pettie, and Hsin-Hao Su. 2015. Scaling algorithms for weighted matching in general graphs. *CoRR* abs/1411.1919v2.

Guy Even, Moti Medina, and Dana Ron. 2015. Distributed maximum matching in bounded degree graphs. In *Proceedings of the International Conference on Distributed Computing and Networking (ICDCN'15)*. ACM, Article 18, 10 pages. DOI: http://dx.doi.org/10.1145/2684464.2684469

Ted Fischer, Andrew V. Goldberg, David J. Haglin, and Serge Plotkin. 1993. Approximating matchings in parallel. *Info. Proc. Lett.* 46, 3 (June 1993), 115–118.

Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. 1999. A faster distributed algorithm for computing maximal matchings deterministically. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing*. 219–228.

Nicholas J. A. Harvey. 2006. Algebraic structures and algorithms for matching and matroid problems. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science*. 531–542.

Jaap-Henk Hoepman. 2004. Simple distributed weighted matchings. CoRR cs.DC/0410047.

Jaap-Henk Hoepman, Shay Kutten, and Zvi Lotker. 2006. Efficient distributed weighted matchings on trees. In *Proceedings of the 13th International Colloquium on Structural Information and Communication Complexity (SIROCCO'06)*. 115–129.

John E. Hopcroft and Richard M. Karp. 1973. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* 2, 4 (Dec. 1973), 225–231.

Stefan Hougardy and Doratha E. Vinkemeier. 2006. Approximating weighted matchings in parallel. *Info. Proc. Lett.* 99, 3 (2006), 119–123.

Amos Israeli and Alon Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. *Info. Proc. Lett.* 22, 2 (1986), 77–80.

Richard M. Karp, Eli Upfal, and Avi Wigderson. 1985. Constructing a perfect matching is in random NC. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*. ACM, 22–32.

Alexander V. Karzanov. 1973. On finding maximum flows in networks with special structure and some applications [in Russian]. In *Mathematical Issues of Production Control*. Vol. 5. Moscow State University Press, 81–94. English translation available from the author's website.

Christos Koufogiannakis and Neal E. Young. 2011. Distributed algorithms for covering, packing and maximum weighted matching. *Distrib. Comput.* 24, 1 (2011), 45–63. DOI:http://dx.doi.org/10.1007/s00446-011-0127-7

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2006. The price of being near-sighted. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, 980–989. DOI:http://dx.doi.org/10.1145/1109557.1109666. Full version available at arXiv:1011.5470.

Nathan Linial. 1992. Locality in distributed graph algorithms. *SIAM J. Comput.* 21 (1992), 193–201.

Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2008. Improved distributed approximate matching. In *Proceedings of the 20th ACM Symposium on Parallellism in Algorithms and Architecture*. 129–136. DOI:http://dx.doi.org/10.1145/1378533.1378558

Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. 2009. Distributed approximate matching. *SIAM J. Comput.* 39, 2 (2009), 445–460.

Laszlo Lovász and Michael D. Plummer. 1986. *Matching Theory*. North Holland, Amsterdam.

Michael Luby. 1986. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.* 15, 4 (Nov. 1986), 1036–1053.

Yishay Mansour and Shai Vardi. 2013. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 260–273.

Nick McKeown. 1999. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.* 7, 2 (1999), 188–201.

Marcin Mucha and Piotr Sankowski. 2004. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual Symposium on Foundations of Computer Science*. 248–255.

Tim Nieberg. 2008. Local, distributed weighted matching on general and wireless topologies. In *Proceedings of the 5th International Workshop on Foundations of Mobile Computing (DIALM-POMC'08)*. ACM, 87–92. DOI:http://dx.doi.org/10.1145/1400863.1400880

Michal Parnas and Dana Ron. 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theo. Comput. Sci.* 381, 1–3 (2007), 183–196. DOI:http://dx.doi.org/10.1016/j.tcs.2007.04.040

Boaz Patt-Shamir, Dror Rawitz, and Gabriel Scalosub. 2012. Distributed approximation of cellular coverage. *J. Parallel Distrib. Comput.* 72, 3 (2012), 402–408. DOI:http://dx.doi.org/10.1016/j.jpdc.2011.12.003

David Peleg. 2000. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Seth Pettie and Peter Sanders. 2004. A simpler linear time $2/3 - \epsilon$ approximation to maximum weight matching. *Info. Proc. Lett.* 91, 6 (2004), 271–276.

Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. 2011. Fast local computation algorithms. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS)*. 223–238.

Vijay Vazirani. 2012. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR* abs/1210.4594.

M. Wattenhofer and R. Wattenhofer. 2004. Distributed weighted matching. In *Proceedings of the 18th International Conference on Distributed Computing (DISC'04)*. 335–348.

Received August 2014; revised March 2015; accepted May 2015