DISTRIBUTED APPROXIMATE MATCHING*

ZVI LOTKER†, BOAZ PATT-SHAMIR‡, AND ADI ROSÉN§

Abstract. We consider distributed algorithms for approximate maximum matching on general graphs. Our main result is a randomized $(4+\epsilon)$-approximation distributed algorithm for maximum weighted matching, whose running time is $O(\log n)$ for any constant $\epsilon > 0$, where n is the number of nodes in the graph. This is, to the best of our knowledge, the first log-time distributed algorithm that achieves constant approximation for maximum weighted matching on general graphs.

In addition, we consider the dynamic case, where nodes are inserted and deleted one at a time. For unweighted dynamic graphs, we give a distributed algorithm that maintains a $(1+\epsilon)$-approximation in $O(1/\epsilon)$ time for each node insertion or deletion, for any constant $\epsilon > 0$. For weighted dynamic graphs we give a constant-factor approximation distributed algorithm that runs in constant time for each insertion or deletion.

Key words. Graph algorithms, distributed algorithms, maximum matching, dynamic algorithms, distributed approximation algorithms

AMS subject classifications. 68Q25, 68W05, 68W15, 68W20, 68W25, 68W40

1. Introduction.

The *maximum matching* problem is undoubtedly one of the most basic problems in computer science and graph theory [12]. In its *unweighted* version we are given an unweighted graph, and the goal is to find a matching (a set of disjoint edges) of maximum cardinality. In the *weighted* version, the edges of the graph have positive weights, and the goal is to find a matching in the graph which maximizes the sum of its edge-weights. This version is usually referred to as the *maximum weighted matching* problem.

For the centralized setting of the problem, Edmonds gave more than forty years ago the first centralized polynomial time algorithms for maximum matching, and for maximum weighted matching, on general graphs [3, 4]. More than twenty years ago, a randomized distributed algorithm for *maximal* unweighted matching (and thus a 2-approximation for maximum unweighted matching) was given in [8]. The expected running time of this algorithm is $O(\log n)$ rounds (and this occurs also with high probability), where n is the number of nodes in the graph.[1] Only much more recently, Wattenhofer and Wattenhofer [14] presented distributed algorithms for computing approximate maximum *weighted* matchings: For trees, they give a 4-approximation algorithm that runs in constant time, and for general graphs, they give a randomized algorithm that runs in $O(\log^2 n)$ time, and with high probability achieves approximation factor 5. Subsequently, in [9] it was proved that any (possibly randomized) distributed algorithm that approximates maximum matching to within a constant must have running time $\Omega(\sqrt{\log n/\log\log n})$. This lower bound holds regardless of the size of the messages used by the algorithm.

In this paper, we give a distributed algorithm for general weighted graphs that (with high probability) approximates the maximum weighted matching to within a

*A PRELIMINARY VERSION OF THIS PAPER APPEARED IN PROC. OF THE 26TH ANN. ACM SYMP. ON PRINCIPLES OF DISTRIBUTED COMPUTING (PODC), 2007.

†Dept. of Communication Systems Engineering, Ben-Gurion University, P.O.Box 653. Beer Sheva 84105, Israel.

‡Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Supported in part by the Israel Science Foundation grant 664/05.

§ CNRS and University of Paris 11, LRI Bât. 490, Université Paris-Sud, 91405 Orsay, France.

[1]The algorithm in [8] is presented as a PRAM algorithm, but it readily works in the distributed message passing model.

factor of $4 + \epsilon$ and has (deterministic) running time of $O(\epsilon^{-1} \log \epsilon^{-1} \log n)$, for any given $\epsilon > 0$. Our result is, to the best of our knowledge, the first log-time distributed algorithm that gives constant-approximation for weighted matching on general graphs. Our algorithms for this case use messages of constant size.

We also consider the model of dynamic graphs. In this model, nodes (with all their incident edges) are inserted and deleted one at a time. We present a distributed algorithm which maintains a $(1 + \epsilon)$-approximate unweighted matching in $O(1/\epsilon)$ time per insertion or deletion, for any given $\epsilon > 0$. For weighted graphs we present a distributed algorithm that maintains a constant-approximation weighted matching, and runs in constant time per insertion or deletion. Our algorithms for the dynamic case are deterministic.

Related work. Maximum matching is a classical optimization problem that was the target of extensive research (see [1, 12]). A number of papers studied the problem from the distributed algorithms perspective. As mentioned above, a distributed algorithm for unweighted maximal matching in general graphs was given in [8]. More recently, a distributed approximation algorithm for weighted matching in general graphs was given in [14]. In addition, Hoepman [5] gave a deterministic distributed $O(n)$-time algorithm that achieves 2-approximation for weighted matching on general graphs. Hoepman et al. [6] gave an (expected) $(2 + \epsilon)$-approximation distributed algorithm for unweighted matching on trees, that runs in $O(1/\epsilon)$ time. An $O(\log^4 n)$ time deterministic distributed algorithm for 1.5-approximation of maximum unweighted matching was given in [2]. Finally, we note that subsequent to the initial publication of our work [11], further improvements to the approximation factors of logarithmic-time distributed algorithms for weighted and unweighted matching were presented in [10]. Specifically, [10] gives a distributed algorithm for unweighted matching that achieves approximation ratio of $1 + \epsilon$ for any $\epsilon > 0$, in time logarithmic in n (but exponential in $1/\epsilon$), and a distributed algorithm for weighted matching with approximation ratio $2 + \epsilon$ (for any $\epsilon > 0$) in time logarithmic in n (and polynomial in $1/\epsilon$). Furthermore, a distributed algorithm for weighted matching with approximation ratio of $1 + \epsilon$ (for any $\epsilon > 0$) in time polylogarithmic in n (and polynomial in $1/\epsilon$) is given; however this algorithm may use messages of linear size.

Organization. The remainder of this paper is organized as follows. In Section 2 we describe our model. In Section 3 we present algorithms for the static case, and in Section 4 we give algorithms for the dynamic model. Some conclusions appear in Section 5.

2. Model.

We consider the standard synchronous message passing distributed model of computation (cf., [13]). The system is modeled as an undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$, where nodes represent processors and edges represent bidirectional communication links. Time progresses in synchronous rounds, where in each round each processor may send (possibly different) messages to its neighbors. All messages sent are then received and processed in the same round by their recipient nodes. Processors may have unique identifiers of $O(\log n)$ bits.

For the purpose of defining weighted matching, edges may have weights, where the minimum possible weight is defined to be 1. We denote by $w(e)$ the weight of edge e.

3. Weighted Matchings in Static Graphs.

In this section we define a distributed algorithm that computes (with high probability) a weighted matching in general graphs whose approximation factor is arbitrarily close to 4. For any fixed

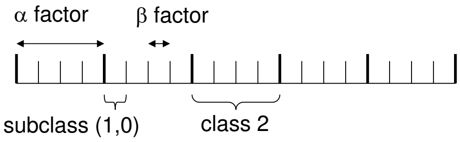

Fig. 3.1. *Example of a partition of edge weights on a logarithmic scale. Classes are demarcated by bold lines. Each class includes weights in the range w to $w(1+1/\epsilon)$ for some w, and each subclass includes weights in the range w to $w(1+\epsilon)$ for some w.*

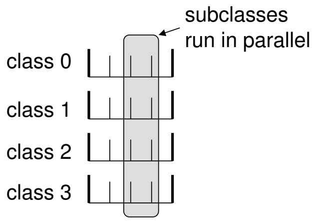

Fig. 3.2. *The weight classes rearranged. Vertically aligned subclasses are run in parallel: starting with the heaviest subclasses $(\star, k-1)$ and ending with the lightest subclasses $(\star, 0)$.*

approximation ratio the running time of our algorithm is $O(\log n)$. For completeness we give at the end of Section 4 a simpler algorithm that also runs in $O(\log n)$ time, but achieves an approximation ratio of 27.

Preliminaries. Suppose that we wish to find a $(4+\epsilon')$-approximation to the maximum weighted matching, for some given $\epsilon' > 0$. For reasons of convenience of notation, we actually give below an algorithm whose approximation ratio is $4+5\epsilon$, for $\epsilon = \frac{\epsilon'}{5}$. We henceforth use ϵ exclusively.

Given ϵ, we define the following parameters:

$$\alpha \stackrel{\text{def}}{=} 1 + \tfrac{1}{\epsilon} \qquad\qquad \beta \stackrel{\text{def}}{=} \tfrac{\alpha}{\alpha-1} \;=\; \epsilon + 1$$

In what follows we assume, without loss of generality, that $1/n \le \epsilon \le 1/2$: if we are given $\epsilon > 1/2$, we run the algorithm with $\epsilon = 1/2$; and if we are given $\epsilon < 1/n$, we run the algorithm of Hoepman [5] that runs in $O(n) = O(\frac{1}{\epsilon})$ time, getting approximation factor 2.

We partition the edge set according to edge weights by a two-level hierarchy as follows (see Figure 3.1). We define weight *classes*, where for $i \ge 0$, class i includes all edges e with $w(e) \in [\alpha^i, \alpha^{i+1})$. Let E_i denote the set of all edges of weight class i. Each class is further divided into $k \stackrel{\text{def}}{=} \lceil \log_\beta \alpha \rceil$ subclasses, where the jth subclass of class i, for $0 \le j < k$, is denoted subclass (i,j). Subclass (i,j) contains all edges of class i whose weights are in $[\alpha^i \cdot \beta^j \,,\, \alpha^i \cdot \beta^{j+1})$.[2] Let $E_{i,j}$ denote the set of edges of subclass (i,j). We denote the highest non-empty class number in the graph by W.

3.1. The Algorithm. Overview. Our approach is to reduce the weighted case to multiple instances of the unweighted case. Roughly speaking, the idea is to run a black-box distributed (possibly Monte Carlo) algorithm for computing maximal

[2]Note that since subclass $(i, k-1)$ is contained in class i, its weight range is $[\alpha^i \cdot \beta^j \,,\, \min\{\alpha^i \cdot \beta^{j+1},\, \alpha^{i+1}\})$.

matching on each subclass separately. In what follows we call this black-box algorithm UWM (for UnWeighted Matching). It is not hard to see that if we run UWM on the subclasses sequentially, from the heaviest to the lightest, deleting all matched nodes from consideration after every invocation of UWM, we get an approximation factor of $2\beta = 2 + 2\epsilon$, but the running time of such an algorithm is linear in the number of subclasses (times the running time of UWM). On the other hand, if we run many instances of UWM in parallel, the result is not necessarily a matching. In our algorithm, we balance the serial and the parallel invocations of UWM as follows.

We run a series of iterations, where in each iteration we run many instances of UWM (see Figure 3.2). In the first iteration, we run, in parallel, a set of independent instance of UWM, one instance for each of the heaviest subclass of each class; in the second iteration, we run in parallel an independent instance of UWM on each of the second-heaviest subclasses; etc. Nodes matched in an execution of UWM on class (i, j) are removed from subsequent invocations of UWM on all subclasses of class i (but not from other classes, see below). This ensures that by the end of all UWM invocations, the collection of edges selected from each class is a matching (in fact, it is a 2β-approximation of the maximum weighted matching of that class). It then remains to resolve conflicts between edges selected from different classes. To this end, we run a distributed algorithm that repeatedly selects heaviest edges and deletes their incident edges. This is run for a logarithmic number of iterations. We prove that the weight of the resulting matching is at least a $(1/4 - \epsilon)$-fraction of the weight of the maximum weighted matching.

Detailed description. To formally specify the algorithm, we conceptually decompose the graph as follows. We define the *component graph* $G_{i,j}$ to consist of all edges from subclass (i, j), and all their endpoint nodes. Note that a node in the original graph may be replicated in many component graphs.

The algorithm consists of two stages. In the first stage (specified in Figure 3.3), we run $k = \lceil \log_\beta \alpha \rceil$ iterations (recall that k is the number of subclasses in a class). In iteration $\ell \geq 1$, we compute an unweighted maximal matching in each of the component graphs $G_{\star,k-\ell}$. That is, an iteration consists of multiple instances of UWM running in parallel, where instance i of UWM in iteration ℓ computes an unweighted maximal matching in $G_{i,k-\ell}$. Note that we can run independent instances of the algorithm in parallel because component graphs have disjoint edge sets. After all instances of UWM in iteration ℓ terminate, we proceed as follows. For every i and every $j < k - \ell$, we remove from all component graphs $G_{i,j}$ all nodes that were matched in iteration ℓ in $G_{i,k-\ell}$. We then proceed to iteration $\ell + 1$. It is important to note that the algorithm processes the subclasses in *decreasing* order within each class.

Let $\mathcal{A}$ denote the set of all edges that are selected by the algorithm at the end of the first stage. Note that $\mathcal{A}$ is not necessarily a matching in the original graph, because an original node may be replicated several times in various component graphs. In the second stage we distill a matching from $\mathcal{A}$ as follows. Partition the edges of $\mathcal{A}$ according to weight classes. Let A_i denote the set of edges from weight class i in $\mathcal{A}$. Observe that in $\mathcal{A}$, a node may have at most one incident edge from each A_i. This is because two edges from the same class are either in the same subclass, in which case the correctness of UWM ensures that they are not both selected to $\mathcal{A}$, or else they are from different subclasses, in which case the first one to be selected eliminates the other. Suppose now that a node v has more than one incident edge in $\mathcal{A}$. Naturally, we would like its heaviest incident edge from $\mathcal{A}$, say (v, u), to be

```
Algorithm STAGE 1
for all i, j do                                                   initialization
   let the component graph G_{i,j} consist of
        the edges in E_{i,j} and their endpoints;
A ← ∅;
for ℓ ← 1 to ⌈log_β α⌉ do                                          iteration ℓ
   for all i do in parallel
        M_{i,k−ℓ} ← UWM(G_{i,k−ℓ})
        for all j < k − ℓ do
             remove from G_{i,j} all nodes matched in M_{i,k−ℓ} (together with their incident edges)
   A ← A ∪ ⋃_i M_{i,k−ℓ};
```

FIG. 3.3. *The first stage of the algorithm.*

```
Algorithm COMBINE (for node v) (Stage 2)
Mark all v's incident edges in A as "eligible."                  initialization
for r ← 1 to 3⌈log_α n⌉ do                                        iteration r
     Let e_v be the highest-weight "eligible" edge incident to v.
     Send "request" on e_v.
     if received "request" on e_v then
          (1) Output e_v as part of the matching.
          (2) Send "not eligible" on all other eligible edges.
          (3) Halt (locally).
     for each incident edge e do
          if "not eligible" message was received on e then
               mark e as "not eligible".
```

FIG. 3.4. *Algorithm COMBINE for the second stage of the algorithm.*

in the final output. We say in this case that (v, u) 'dominates' the other edges from $\mathcal{A}$ incident to v. In algorithm COMBINE, specified in Figure 3.4, the idea is to find edges that dominate all their incident edges, at both their endpoints (this fact can be established in constant time). However, in the case of a long 'dominance chain,' (where an edge dominates on one of its endpoints, and is dominated on the other endpoint) only the last (heaviest) edge in the chain is selected, while every other edge could be selected. To extract more weight form such chains, we iterate the procedure: after selecting the edges that are dominating on both their endpoints, we delete them together with their adjacent nodes, and again select edges that are dominating on both endpoints. We repeat this procedure a logarithmic number of times.

3.2. Analysis. We now proceed to analyze the algorithm. The following concept is useful.

DEFINITION 3.1. *Let A be a set of edges, let $M \subseteq A$ be a set of disjoint edges, and let $\delta \geq 1$. M is said to be δ-*greedy maximal with respect to *A if for each edge $e \in A$ we have either*

- *$e \in M$, or*
- *there is an edge $e' \in M$ such that e and e' share an endpoint, and $w(e') \geq w(e)/\delta$.*

Note that for unweighted graphs, i.e., graphs where all edge-weights are the same, a δ-greedy maximal matching, for any $\delta \geq 1$, is just any maximal matching.

The following lemma states the crucial property of Stage 1. Recall that E_i is the set of all class-i edges, and that $A_i = \mathcal{A} \cap E_i$.

LEMMA 3.2. *Assume that all UWM instances in Stage 1 output a maximal matching. Then by the end of Stage 1 of the algorithm, for each class i, A_i is β-greedy maximal with respect to E_i.*

Proof: Let A_i^ℓ denote the set of class-i edges that were added to $\mathcal{A}$ by the end of iteration ℓ, $\ell \geq 1$. We show for each class i, by induction on ℓ, that A_i^ℓ is β-greedy maximal with respect to the edges of $B_i^\ell \stackrel{\text{def}}{=} \bigcup_{j \geq k-\ell} E_{i,j}$. The base case $\ell = 1$ follows from the correctness of the UWM algorithm: A_i^1 is just a maximal matching in the component graph $G_{i,k-1}$, and B_i^ℓ is exactly the edge set of $G_{i,k-1}$. For the inductive step, assume that the invariant holds for $\ell - 1$, and consider ℓ. First we claim that A_i^ℓ does not contain intersecting edges. This is true for edges in $A_i^{\ell-1}$ by induction; edges added in iteration ℓ do not intersect each other by the correctness of UWM; and edges added in iteration ℓ do not intersect edges in $A_i^{\ell-1}$ due to the removal, at the end of each iteration, of edges intersecting with edges picked during that iteration. We now prove that A_i^ℓ is β-greedy maximal w.r.t. B_i^ℓ. Let $e \in B_i^\ell$. If $e \in A_i^\ell$ we are done. Otherwise, by the correctness of UWM, it must be the case that either (1) e was not present in the graph on which the UWM ran in iteration ℓ, or else (2) e intersects one of the edges added to A_i in iteration ℓ. In both cases, e must intersect an edge $e' \in B_i^\ell$ whose subclass is not smaller than the subclass of e. The result follows: if e' is in a higher subclass, then $w(e') > w(e)$, and if e' is in the same subclass as e, then $w(e') \geq w(e)/\beta$ (in fact, $w(e') > w(e)/\beta$). ∎

Lemma 3.2 establishes the relation induced by the sequential nature of the iterations. We now proceed to analyze Algorithm COMBINE, which manages the results of the parallel executions within an iteration. First note that by the code, for any edge $e = (u, v)$ and any time, either (a) e is "eligible" on both its endpoints v and u, or (b) one of its endpoints has halted, and e is "not eligible" on the other endpoint, or (c) both its endpoints have halted. We can therefore talk about "not eligible" edges (situations (b) and (c)), and "eligible" edges (situation (a)). This is formalized in the following definition.

DEFINITION 3.3. *For any given time, an edge e is* eligible *if the nodes on both its endpoints are active, and edge e is marked in both of them as eligible. Otherwise, edge e is* not eligible.

We note that when the algorithm starts all edges are eligible. We now have the following straightforward property.

CLAIM 3.4. *If an edge e of class i becomes "not eligible" at some point in Algorithm COMBINE, then there is another edge e' such that (1) e' is incident to e, (2) e' is of class $i' > i$, and (3) e' is in the output of COMBINE.*

Proof: Directly from the code of COMBINE. Assertions (1) and (3) follow since an edge e becomes "not eligible" only when an incident edge e' is chosen to the output. An edge e' is chosen to the output when both its endpoints send a "request" message. In particular, the node where e and e' intersect sends a "request" message on e' when e was eligible, which means that e' has higher weight than e, proving assertion (2). ∎

We now state the main loop invariant of COMBINE.

LEMMA 3.5. *Let W be the highest non-empty class in the graph, and let $r \geq 1$.*

After iteration r of Algorithm COMBINE, for all $i > W - r$, every edge $e \in A_i$ is either output by COMBINE, or e intersects another edge e' output by COMBINE, and $e' \in A_{i'}$ for some $i' > i$.

Proof: We proceed by induction on r. We prove the following slightly stronger invariant: Let M be the output of COMBINE. After iteration $r \geq 0$ the following hold:

(1) There is no eligible edge of class $i > W - r$; and
(2) for any edge $e \in \bigcup_{i=W-r+1}^{W} A_i$, either $e \in M$, or it intersects an edge $e' \in M$ s.t. e' is in a weight class higher than the weight class of e.

The basis of the induction, $r = 0$, is trivial: by assumption, (1) the maximum weight class in the graph is W, and (2) $\bigcup_{i=W+1}^{W} A_i$ is empty. For the inductive step, let $r \geq 1$. We first claim that if there is an eligible edge $e \in A_{W-r+1}$ when iteration r starts, then e is added to the output M during iteration r. This follows since by the induction hypothesis (applied to $r-1$) there are no eligible edges of classes higher than $W - r + 1$, and hence by the code of COMBINE, both endpoints of e will select e as their heaviest eligible edge. Thus e is added to the output M. It follows immediately that e becomes "not eligible" by the end of iteration r, and this, together with the induction hypothesis, proves that Part (1) holds after iteration r.

For Part (2), let $e \in \bigcup_{i=W-r+1}^{W} A_i$. We consider three cases. First, if $e \in \bigcup_{i=W-r+2}^{W} A_i$, we are done by the induction hypothesis. Second, if $e \in M$, Part (2) of the invariant clearly holds. The only remaining case is when $e \in A_{W-r+1}$, and e is not chosen to the output before or at iteration r. As proved above, if e were "eligible" at the beginning of iteration r, it would have been chosen at iteration r. Hence, e is not eligible when iteration r starts. By claim 3.4, e intersects an edge e' which is in class $i' > W - r + 1$ and is in M. ∎

We can now prove the approximation factor of our algorithm.

Theorem 3.6. *Assume that for all $i \geq 0$, A_i is β-greedy maximal with respect to E_i. Let M be the matching that algorithm COMBINE outputs, and let $\mathcal{M}$ be any matching in the graph. Then*

$$w(\mathcal{M}) \leq (4 + 5\epsilon) w(M) .$$

Proof: We first bound from above the total weight of edges from $\mathcal{M}$ in "light" classes, and then bound from above the weight of the edges of $\mathcal{M}$ is "heavy" classes.

Let W be the highest non-empty class in the graph. Let X be the total weight of edges in $\mathcal{M}$ which are in the top $3\lceil \log_\alpha n \rceil$ classes, i.e, in classes $i > W - 3\lceil \log_\alpha n \rceil$. Let Y denote the weight of the remaining edges in $\mathcal{M}$, so $X + Y = w(\mathcal{M})$. Note that $Y \leq \frac{n}{2} \cdot \alpha^{W - 3\lceil \log_\alpha n \rceil + 1}$. Now, $w(M) \geq \alpha^W$ because M includes at least one edge from the top weight class, and since $\epsilon > \frac{1}{n}$ we have that

$$Y \leq \frac{1}{n} \cdot w(M) \leq \epsilon \cdot w(M) . \tag{3.1}$$

We now turn to bound X, the weight of edges $e \in \mathcal{M}$ which are in classes $i > W - 3\lceil \log_\alpha n \rceil$. To this end, we construct the following charging scheme, that maps each such edge $e \in \mathcal{M}$ to an edge $f(e) \in M$. Suppose $e \in \mathcal{M}$ belongs to class i. Then $f(e) \in M$ is defined as follows.

(1) If $e \in A_i$:
   (1a) If $e \in M$, then $f(e) = e$.

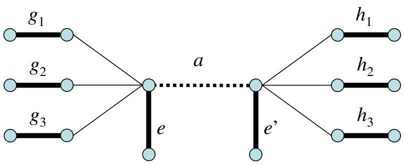

FIG. 3.5. *Example of the charging scheme. All thick edges are from the matching $\mathcal{M}$ and are mapped to the dashed edge a from M. The edges e and e' (case 1b) cannot be from a subclass higher than a, and the edges g_i and h_i (case 2b) must belong to classes strictly lower than a's. Furthermore, no two g_i edges belong to the same class, and no two h_i edges belong to the same class.*

(1b) If $e \notin M$, then by Lemma 3.5, at least one of the endpoints of e is matched in M with an edge e_1 of class $i' > i$ (if there are two such edges, pick one arbitrarily). In this case, we define $f(e) = e_1$.

(2) If $e \notin A_i$, then, since A_i is a maximal matching in E_i, it must be the case that at least one of the endpoints of e is shared with an edge $e_2 \in A_i$ (if there are two such edges, pick one arbitrarily). In this case, we define $f(e)$ similarly to Case 1 with e_2 playing the role of e:

(2a) If $e_2 \in M$, then $f(e) = e_2$.

(2b) If $e_2 \notin M$, then by Lemma 3.5, at least one of the endpoints of e is matched in M with an edge e_3 of class $i' > i$ (if there are two such edges, pick one arbitrarily). In this case, we define $f(e) = e_3$.

The situation is summarized in Figure 3.5, showing which edges of $\mathcal{M}$ are mapped to an edge of M.

Let us now bound from above the weight assigned to each edge $a \in M$, as a function of the weight of that edge, $w(a)$. By definition of the mapping f, an edge $a \in M$ is assigned the weight of an edge $e \in \mathcal{A}$ only if they share an endpoint (Case 1). An edge a is assigned the weight of an edge $e \notin \mathcal{A}$ from class i (Case 2) only if they share an endpoint, or if there is an "intermediate" edge $e' \in A_i$ which intersects both a and e.

Each endpoint of $a \in M$ is additionally the endpoint of at most a single edge $e \in \mathcal{M}$. Thus an assignment from a neighboring edge can add, for each endpoint of a, at most the weight $\beta \cdot w(a)$: the assigned weight is at most $w(a)$ if the edge e is the same as a (Case 1a) or the class of a is higher than the class of e (Case 1b). The assigned weight is at most $\beta \cdot w(a)$ if e and a are not the same edge and are in the same class (Case 2a) since A_i is β-greedy maximal by Lemma 3.2. Note that the class of a cannot be smaller than the class of e by Lemma 3.5.

We now consider the "indirect" assignments of Case 2b. Note that for each endpoint of a there can be at most one edge from each A_i that contains this endpoint. All edges e mapped to a by Case 2b must be from classes strictly lower than the class of a. Therefore the total weight mapped to a by Case 2b, per endpoint, is at most

$$w(a) \cdot \sum_{s=0}^{\infty} \alpha^{-s} = w(a) \cdot \frac{\alpha}{\alpha - 1} = w(a)\beta \ .$$

We can therefore conclude that the total weight assigned to edge a by edges of $\mathcal{M}$ that intersect a is at most $2\beta \cdot w(a)$, and that the total weight assigned to edge a by edges of $\mathcal{M}$ that do not intersect a is at most $2\beta \cdot w(a)$. The total is at most $4\beta \cdot w(a)$. Summing over all edges, and combining with Eq. (3.1), we get that $w(\mathcal{M}) = X + Y \leq \epsilon \cdot w(M) + 4\beta \cdot w(M) = (4 + 5\epsilon)w(M)$. ∎

The next theorem states the running time of our algorithm.

THEOREM 3.7. *The running time of the algorithm is* $O(\log n/\log\frac{1}{\epsilon}+(\frac{1}{\epsilon}\log\frac{1}{\epsilon})T_{\mathrm{UWM}})$, *where* T_{UWM} *is the running time of each invocation of UWM.*

Proof: The number of iterations in the first stage is $k=\lceil\log_\beta\alpha\rceil$, and each iteration takes T_{UWM} rounds. Now, since $0<\epsilon\le 1/2$, we have that

$$\log_\beta\alpha=\frac{\ln\alpha}{\ln\beta}=\frac{\ln(1+\frac{1}{\epsilon})}{\ln(1+\epsilon)}\le\frac{2\log\frac{1}{\epsilon}}{\epsilon}\ .$$

It follows that the total running time of the first stage is $O((\frac{1}{\epsilon}\log\frac{1}{\epsilon})T_{\mathrm{UWM}})$, for $0<\epsilon\le 1/2$. The number of iterations in the second stage is $\lceil 3\log_\alpha n\rceil=O(\log n/\log\frac{1}{\epsilon})$ for $0<\epsilon\le 1/2$, and each iteration takes constant time. ∎

As mentioned above, the algorithm that computes maximal unweighted matching (denoted UWM) is treated as a black-box. To be useful within our algorithm, this black-box algorithm must have a deterministic upper bound on its running time. We denote this upper bound T_{UWM}: after T_{UWM} rounds, all nodes output which of their incident edges is in the maximal matching, and the next iteration can start. As for its correctness, it is sufficient that UWM computes a maximal matching with high probability

Finally, to get our result we plug the algorithm of [8] as the black-box implementation of UWM, where T_{UWM} is $O(\log n)$, and the probability of success of each invocation is set to at least $1-\frac{1}{n^4}$ (see Section 3.3 below). Note that the total number of invocations of UWM is upper-bounded by $\binom{n}{2}$, because there can be at most $|E|$ non-empty subclasses. Hence we have from the union bound that the probability that no UWM invocation fails is at least $1-1/n^2$. We thus arrive at the top-level result of this section.

COROLLARY 3.8. *Let* $\epsilon>0$. *Our algorithm for weighted matching, using the unweighted maximal matching algorithm of [8] as an implementation of UWM, runs in time* $O(\epsilon^{-1}\log\epsilon^{-1}\log n)$, *and with high probability finds a matching whose weight is at least* $1/(4+\epsilon)$ *of the maximum weighted matching.*

We note that the algorithm uses messages of constant size.

Remark: For completeness, we note that an extension of the algorithm of [14] yields (with high probability) approximation factor $2+\epsilon$ in $O(\epsilon^{-1}\log^2 n)$ deterministic time, for any constant $\epsilon>0$, assuming that the largest edge weight is known. Without loss of generality we assume that $1/n\le\epsilon\le 1/2$ (cf. comments above). Briefly, the idea is to divide the edges into weight classes of the form $[(1+\epsilon)^i,(1+\epsilon)^{i+1})$, and then run a UWM algorithm on each class in descending order of weights. The algorithm starts with the highest non-empty class, and goes down for $3\lceil\log_{1+\epsilon}n\rceil$ classes. After each execution of UWM, all edges selected by UWM are output, and they are removed along with all their incident edges. We use the algorithm of [8] as an implementation of UWM by running it for $O(\log n)$ time, ensuring that each invocation computes a maximal matching with probability at least $1-\frac{1}{n^4}$ (see Section 3.3 below). We get a randomized algorithm with (deterministic) running time of $O(\log_{1+\epsilon}n\cdot\log n)=O(\epsilon^{-1}\log^2 n)$. Since the number of nonempty edge classes is at most $\binom{n}{2}$, using the Union Bound we get that the probability that all invocations of UWM compute a maximal matching is at at least $1-\frac{1}{n^2}$. In this case, the weighted matching computed by the algorithm is within a factor of $2+3\epsilon$ of the maximum weighted matching: it is a $2(1+\epsilon)$ approximation with respect to the maximum weighted matching of the highest $3\log_{1+\epsilon}n$ edge classes; the weight lost in the remaining classes represents at most an ϵ fraction of the weight of the computed matching (since $1/n\le\epsilon$).

3.3. Implementation of UWM. We implement UWM using the randomized algorithm of [8], which we call below "Algorithm RMM" (for Randomized Maximal Matching). To complete the analysis, we argue about using Algorithm RMM in our algorithm. The following proposition is a direct consequence of the main argument of [8].

PROPOSITION 3.9. *Let G be a graph with at most m edges. There exists a constant $c > 0$ such that for any $x \geq 1$, if Algorithm RMM is run for $x \cdot c \log m$ rounds, then it outputs a maximal matching with probability at least $1 - m^{-x}$.*

Proposition 3.9 proves the following lemma.

LEMMA 3.10. *Let G be a graph with at most n nodes. There exists a constant $\gamma > 0$ such that if we implement UWM as algorithm RMM with $T_{\mathrm{UWM}} = \gamma \log n$, then the output of UWM is a maximal matching with probability at least $1 - \frac{1}{n^4}$.*

Proof: In our algorithm, all invocations of UWM are run on component graphs. Obviously, the number of edges in any component graph is at most $\binom{n}{2} < n^2/2$. Therefore, when implementing UWM by Algorithm RMM, we have from Proposition 3.9 (by taking $m = n^2$, and $x = 2$), that there exists a constant $\gamma > 0$ such that if we run Algorithm RMM for $\gamma \log n$ rounds, the output is correct (i.e., the output is a maximal matching) with probability at least $1 - 1/n^4$. ∎

4. Dynamic graphs. In this section we consider dynamic graphs. In this model, the input is a sequence of topological changes; without loss of generality, we assume that each topological change is either the insertion or the deletion of a single node along with its incident edges (edge insertion and deletion can be simulated by deleting a node and re-inserting it with the new set of incident edges). After each topological change, the system makes a computation, and outputs a local indication for each edge, whether it is in the matching or not. We consider both the unweighted and weighted graphs cases, and present distributed deterministic approximation algorithm for each case.

4.1. Unweighted Dynamic Graphs. In this section we prove the following result.

THEOREM 4.1. *Let $\epsilon > 0$. There exists a distributed algorithm whose running time per topological change is $O(1/\epsilon)$, and whose output, after processing the change, is at least $\frac{1}{1+\epsilon}$ times the size of the maximum matching.*

We need the following standard concept.

DEFINITION 4.2. *Let $G = (V, E)$ be a graph, let $M \subseteq E$ be a set of non-intersecting edges in E, and let $k \geq 1$. A path $v_0, v_1, \ldots, v_{2(k-1)}, v_{2k-1}$ is an* augmenting path of length $2k - 1$ *with respect to M if for all $1 \leq i \leq k - 1$, $(v_{2i-1}, v_{2i}) \in M$, for all $1 \leq i \leq k$ $(v_{2(i-1)}, v_{2i-1}) \notin M$, and both v_0 and v_{2k-1} are not endpoints of any edge in M.*

Our algorithm relies on the following graph-theoretic proposition (cf., for example, [7]).

THEOREM 4.3. *Let $G = (V, E)$ be a graph, and let $M \subseteq E$ be a set of non-intersecting edges. Let k be a positive integer. If there is no augmenting path of length $2k - 1$ or less w.r.t. M, then the size of the largest matching in G is at most $\frac{k+1}{k} \cdot |M|$.*

The idea behind our algorithm is to maintain the invariant that the output never contains augmenting paths shorter than $\lfloor 2/\epsilon \rfloor$. This invariant implies, by Theorem 4.3, that the size of the output matching is close to the size of the best possible matching, as stated in Theorem 4.1. It remains to show how to maintain that invariant.

We start with the event of node insertion. We use the following property.

Lemma 4.4. *Let $G = (V, E)$ be a graph, let $M \subseteq E$ be a matching, and suppose that there are no augmenting paths in G of length at most ℓ w.r.t. M. Let $v' \notin V$ be a new node, and let $E' \subseteq \{v'\} \times V$ be its incident edges. Let $G' = (V \cup \{v'\}, E \cup E')$ be the resulting graph. Then any augmenting path of length at most ℓ in G' has v' as one of its endpoints.*

Proof: Let P be an augmenting paths of length at most ℓ in G' (if there are no such paths we are done). First, note that P must contain v'. Otherwise, P is an augmenting path in G of length at most ℓ, contradicting the assumption. Next, note that v' cannot be in the middle of P, because no edge incident to v' is in M, and any augmenting path w.r.t. M alternates between edges in M and edges not in M. ∎

In our algorithm, upon the insertion of a new node v', we search for all augmenting paths that start with v' and whose length is at most $2 \cdot \frac{1}{\epsilon} - 1$. If no such path exists, we are done. Otherwise, let P be the shortest such path. We *augment along* P, i.e., we switch the roles of the edges in P as follows. Let P_M be the set of edges in $P \cap M$, and let P_A be the set of edges in $P \setminus M$. We set $M' = M \cup P_A \setminus P_M$, and declare M' as the new output.

The correctness of our algorithm relies on the following key lemma, which may be of independent interest.

Lemma 4.5. *Let G be a graph, let M be a matching in G, and let ℓ be such that there are no augmenting paths of length ℓ or less w.r.t. M in G. Let G' be the graph obtained from G by adding a node v' and its incident edges. Let P be the shortest augmenting path in G', and suppose that $|P| \leq \ell$. Let M' be the matching obtained by augmenting M along P. Then in G' there are no augmenting paths of length ℓ or less with respect to M'.*

Proof: Denote the nodes in P by $v', v_1, v_2, \ldots, v_n$ (by Lemma 4.4, v' must be one of the endpoints of P). Let P' be any augmenting path w.r.t. M' in G'. If P' does not contain any node from P, then P' is an augmenting path w.r.t. M in G, and hence its length is more than ℓ, and we are done. Therefore suppose that P' shares at least one node with P. Observe that P and P' must share an edge, because that node is matched in M', and P (resp. P') contains all edges of M' incident to nodes in P (resp. P'). Fix an arbitrary orientation of P', and under that orientation, let v_i be the first node in P' which is also in P, and let v_j be the last node in P' which is also in P. Without loss of generality assume $i < j$ (otherwise reverse the orientation of P'; equality is ruled out by the observation above).

Consider P'. In general, P' may take a "detour" leaving P between v_i and v_j and later return to P. Let $\tilde{v}$ be the node after which P' leaves P for the first time (after v_i), and let $\hat{v}$ be the last node on that detour (i.e., P and P' coincide between $\hat{v}$ and v_j). We proceed by case analysis, depending on whether edge (v_i, v_{i+1}) is in M or not, and whether edge (v_{j-1}, v_j) is in M or not (as noted above, it must be the case that $j \geq i + 1$ and hence both edges must exist, but possibly they are the same edge). There are 4 cases to consider (see Figure 4.1). In each case, we construct from P and P' a new augmenting path w.r.t. M in G' , and using the fact that P is the shortest such path, we prove that $|P'| > \ell$.

Case 1: $(v_i, v_{i+1}) \notin M$, and $(v_{j-1}, v_j) \notin M$. P' goes from w to v_i, and from v_i it must turn to v_{i+1}, and eventually get to v_{j-1} (and then to v_j). Note that the last edge on P' before v_i is not in M; therefore, the path $v' \rightsquigarrow v_i \rightsquigarrow w$ is an augmenting path in G' w.r.t. M. Since P is a shortest augmenting path in G' w.r.t. M, then,

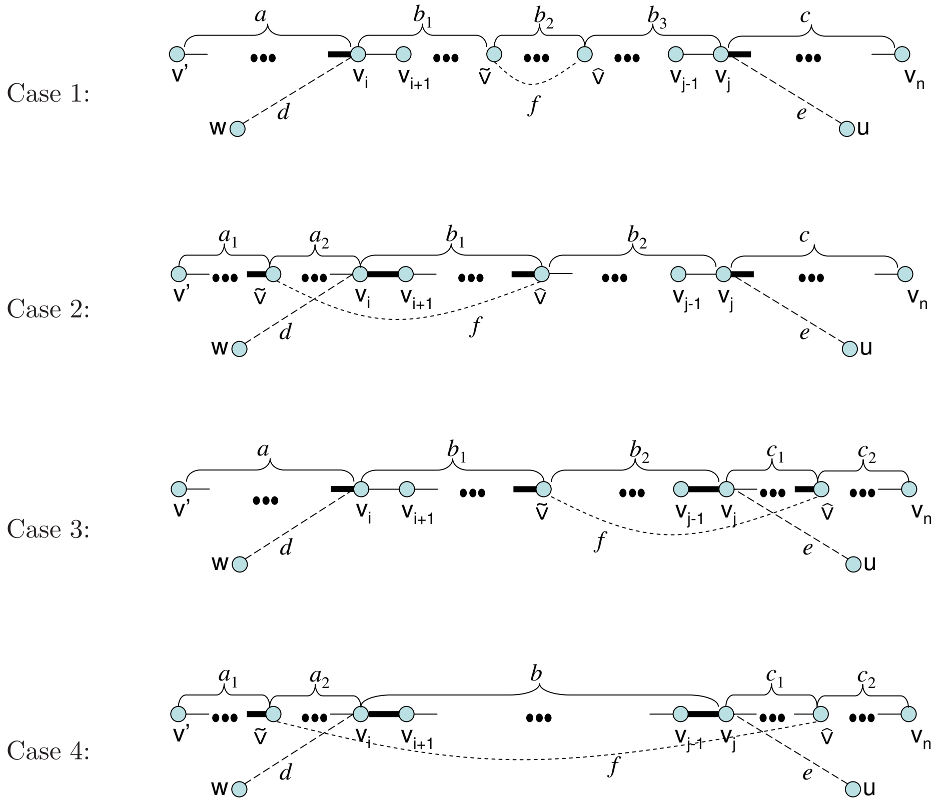

FIG. 4.1. *Schematic representation of the cases considered in the proof of Lemma 4.5. Thick lines denote edges in M, and dashed lines denote paths. The values a, b, c, etc., denote the number of edges in the sub-paths indicated by the curly brackets.*

using the notation in Figure 4.1, we have that it must hold that

$$a + d \geq a + b_1 + b_2 + b_3 + c \quad \Rightarrow \quad d \geq c \; . \tag{4.1}$$

Similarly, $u \rightsquigarrow v_j \rightsquigarrow v_n$ is an augmenting path in G w.r.t. M, and by assumption on the length of the shortest augmenting path in G w.r.t. M, we have

$$e + c > \ell \; , \tag{4.2}$$

and hence we have that

$$|P'| = d + b_1 + f + b_3 + e \geq c + e > \ell \; ,$$

which proves case 1.

Case 2: $(v_i, v_{i+1}) \in M$, and $(v_{j-1}, v_j) \notin M$. As before, $P' = w \rightsquigarrow v_i \rightsquigarrow \tilde{v} \rightsquigarrow \hat{v} \rightsquigarrow v_j \rightsquigarrow u$. The path $v' \rightsquigarrow \tilde{v} \rightsquigarrow \hat{v} \rightsquigarrow v_i \rightsquigarrow w$ is an augmenting path in G' w.r.t. M, and since P is a shortest augmenting path in G' w.r.t. M, we have

$$a_1 + f + b_1 + d \geq a_1 + a_2 + b_1 + b_2 + c \quad \Rightarrow \quad f + d \geq c \; . \tag{4.3}$$

Similarly to Case 1, we have an augmenting path w.r.t. M in G $v_n \rightsquigarrow v_j \rightsquigarrow u$, and hence

$$e + c > \ell. \tag{4.4}$$

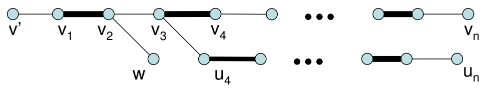

Fig. 4.2. *Example showing that augmenting along the shortest augmenting path is necessary. Bold lines denote edges in the matching.*

Therefore,

$$|P'| = d + a_2 + f + b_2 + e \geq a_2 + b_2 + c + e > \ell \ ,$$

which proves case 2.

Case 3: $(v_i, v_{i+1}) \notin M$, and $(v_{j-1}, v_j) \in M$. In this case the path $v' \rightsquigarrow v_i \rightsquigarrow w$ is an augmenting path in G' w.r.t. M, and by the minimality of P

$$a + d \geq a + b_1 + b_2 + c_1 + c_2 \quad \Rightarrow \quad d \geq b_2 + c_2 \ . \tag{4.5}$$

Furthermore, we have that $u \rightsquigarrow v_j \rightsquigarrow \tilde{v} \rightsquigarrow \hat{v} \rightsquigarrow v_n$ is an augmenting path in G w.r.t. M and hence

$$e + b_2 + f + c_2 > \ell \ . \tag{4.6}$$

Therefore,

$$|P'| = d + b_1 + f + c_1 + e \geq b_2 + c_2 + b_1 + f + c_1 + e > \ell \ ,$$

which proves case 3.

Case 4: $(v_i, v_{i+1}) \in M$, and $(v_{j-1}, v_j) \in M$. Then the path $v' \rightsquigarrow \tilde{v} \rightsquigarrow \hat{v} \rightsquigarrow v_n$ is an augmenting path in G' w.r.t. M, and by the minimality of P

$$a_1 + f + c_2 \geq a_1 + a_2 + b + c_1 + c_2 \quad \Rightarrow \quad f \geq b \ . \tag{4.7}$$

Also, the path $w \rightsquigarrow v_i \rightsquigarrow v_j \rightsquigarrow u$ is an augmenting path w.r.t. M in G, and therefore

$$d + b + e > \ell \ . \tag{4.8}$$

Again, we obtain

$$|P'| = d + a_2 + f + c_1 + e \geq d + a_2 + b + c_1 + e > \ell \ ,$$

which proves case 4. ∎

The algorithm. Based on the above observations, the algorithm is straightforward. We inductively assume that before any topological change there is no augmenting path or length at most $\lfloor 2/\epsilon \rfloor$. This is clearly true when the algorithm starts with an empty graph. Whenever a node v' is inserted, it initiates an exploration of the topology of the graph up to distance $2/\epsilon + 1$ from itself, so as to find any augmenting path (w.r.t. the current matching M) which starts with v', and is of length at most $\lfloor 2/\epsilon \rfloor$. If no such path is found the algorithm terminates. Otherwise, a shortest augmenting path is chosen (ties broken arbitrarily), and its edges flip their role: matching edge become non-matching edge and vice versa. Note that this algorithm uses messages that are large enough to encode neighborhoods of radius $O(1/\epsilon)$, whose size in general may be linear in the size of the graph.

When a node v is deleted, there are two cases. If v was not matched, then the algorithm terminates immediately. Otherwise, suppose that $(v, v') \in M$ for some node v'. In this case, the algorithm re-inserts v' using the insertion algorithm. Note that if there are no augmenting paths of length at most ℓ in G, then there are no such paths in $G \setminus \{v, v'\}$.

Importance of augmenting along the shortest path. We note that augmenting along an arbitrary augmenting path of length at most ℓ does not preserve the invariant that there are no augmenting paths of length ℓ or less. Consider the situation as depicted in Figure 4.2. If the lengths of the paths $v_3 \rightsquigarrow v_n$ and $v_3 \rightsquigarrow u_n$ are slightly more than $\ell/2$ edges, then the path $u_n \rightsquigarrow v_3 \rightsquigarrow v_n$ is not an augmenting path of length ℓ or less, but $v' \rightsquigarrow v_n$ is (unless ℓ is very small). If we augment along $v' \rightsquigarrow v_n$ instead of along $v' \rightsquigarrow w$, then $w \rightsquigarrow u_n$ becomes an augmenting path of length ℓ or less.

4.2. Weighted Dynamic Graphs. We now show how to maintain constant approximation weighted matching in dynamic graphs (when the edges are weighted). Following each topological change, our algorithm runs in constant time. In our algorithm and analysis below we do not attempt to optimize the constants.

Our algorithm is based on the idea to reduce the weighted case to the unweighted case, and apply a simplified version of the COMBINE algorithm. More formally, the algorithm is as follows.

Algorithm WeightedDynamic

(1) We partition the edges into disjoint classes, where all edges in class $i \geq 0$ have weight in $[3^i, 3^{i+1})$. For each such class our algorithm will maintain a 2-approximation maximum unweighted matching.
(2) When a node is inserted, it initiates the unweighted algorithm for each weight class, according to the weights of its incident edges. The algorithms are run with $\epsilon = 1$, which in fact means that each new edge is added to the output (of its class) greedily, i.e., if and only if both its endpoints are not matched in that class. After this step we again have for each class a 2-approximation maximum unweighted matching.
(3) After $O(1)$ time, all algorithms terminate, and each node may have at most one incident edge matched for every weight class. Each node then picks among these edges, as a candidate for the output, the matched incident edge having the highest weight class (if such edge exists).
(4) An edge is output if and only if it is chosen as the candidate by both its endpoints.

Note that each of the weight-class algorithms works only to distance $O(1/\epsilon) = O(1)$ from the location of the topological change, and therefore the only possible changes in the output are in that neighborhood. It follows that Steps 3 and 4 need to be carried out only at distance $O(1/\epsilon) = O(1)$ from the location of the change—more remote nodes in the graph do not change their output.

Analysis. Let A_i be the output of the algorithm for weight class i, and let OPT denote the optimum weighted matching. We start with the following simple property.

LEMMA 4.6. $w(\mathrm{OPT}) < 6 \cdot \sum_i w(A_i)$.

Proof: Let OPT_i be the optimum weighted matching, if only edges of class i are considered. Then $|A_i| \geq |\mathrm{OPT}_i|/2$, and for all $e \in A_i$ and all $e' \in \mathrm{OPT}_i$ we have $w(e) > w(e')/3$. Therefore $w(\mathrm{OPT}_i) < 6w(A_i)$ for each i. Summing over all weight classes yields the claim. ∎

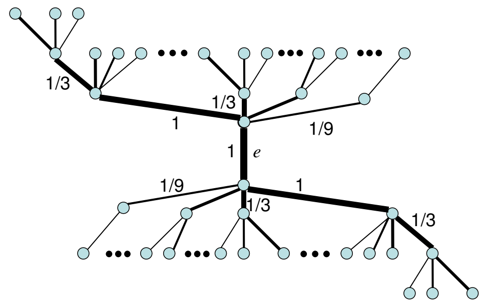

FIG. 4.3. *Trees of rejected A_i edges hanging on an edge $e \in M$ (see Lemma 4.7).*

Let M be the matching output by algorithm WeightedDynamic. We now give a lower bound on the weight of M as a function of the weights of the A_i's.

LEMMA 4.7. $\sum_i w(A_i) < \frac{9}{2} \cdot w(M)$.

Proof: We bound the total weight of the edges that are in the A_i's but are not in M by mapping each such rejected edge to an edge in M. The mapping is natural: an edge in A_i is not in M only if it is incident to an edge in A_j for some $j > i$. Transitively, we have a tree of rejected A_i edges hanging on each endpoint of each edge which is in the output matching M. We now bound the total weight in such a tree. Let $e \in M$, and denote the weight class of e by i_0. On each endpoint of e there may be at most one edge from each weight class smaller than i_0 that was rejected by e. Each such edge e' may in turn have rejected an edge from each class smaller than its own. By induction, it follows that the number of rejected edges from weight class $i_0 - j$, $j \geq 1$ (per endpoint of e) is at most j. The weight of the edges in class $i_0 - j$ is less than $w(e)/3^{j-1}$, and thus we have that the total weight of one tree of rejected edges hanging from e is strictly less than

$$\sum_{j=0}^{\infty} w(e) \cdot \frac{j+1}{3^j} = w(e) \left(\sum_{i=0}^{\infty} 3^{-j} \right)^2 = w(e) \cdot \frac{9}{4} \ .$$

Since there is one such tree for each endpoint of e, it follows that the total weight rejected by e is less than $w(e) \cdot 9/2$. Summing over all edges in M yields the claim. ▮

We can now summarize with the following theorem.

THEOREM 4.8. $w(\mathrm{OPT}) < 27 \cdot w(M)$.

Proof: Follows from Lemma 4.6 and Lemma 4.7. ▮

Note that all the algorithms for the various weight classes run in parallel in time $O(1/\epsilon) = O(1)$ (since we use $\epsilon = 1$). The combining stage runs in constant time. If follows immediately that the running time of our algorithm is constant per node insertion or deletion. We note that since we run the unweighted dynamic algorithms with $\epsilon = 1$, then each new edge is added to the unweighed matching of its class greedily, and thus messages of constant size are sufficient for the algorithm for dynamic weighted graphs.

Remark: We note that the above algorithm also suggests a logarithmic-time, constant-approximation distributed algorithm for static weighted graphs as follows. Partition the edges into classes $i \geq 0$, such that class i contains exactly all edges whose weight is in $[3^i, 3^{i+1})$. Run in parallel UWM within each class for $O(\log n)$ rounds. The output of each of these algorithms is, with probability at least $1 - \frac{1}{n^4}$, a 2-approximation unweighted matching for the relevant class (cf. Section 3.3). Since there are at most $\binom{n}{2}$ non-empty classes, all UWM executions terminate correctly with probability at least $1 - \frac{1}{n^2}$. To obtain a single weighted matching, combine the results of these algorithms in constant time as in algorithm WeightedDynamic (Steps 3 and 4). The analyses of Lemmas 4.6 and 4.7 apply as is, implying an approximation ratio of 27. The running time is obviously $O(\log n)$ rounds.

5. Conclusions. Distributed matching is a fundamental network problem, which has been the subject of active research for decades. Yet, determining the exact complexity of this problem remains an elusive target. In this paper we have narrowed the gap between the lower and upper bounds for weighted matching. In particular, we have given the first, to the best of our knowledge, log-time distributed algorithm that achieves constant approximation for maximum weighted matching on general graphs. We note that following our work, several improvements on logarithmic-time distributed algorithms for matching were presented in [10]. However, several very interesting questions remain open. For example, a long-standing open problem is to find a deterministic log-time distributed algorithm for maximal matching. This problem is in fact a special case of the maximal independent set problem, which has a log-time randomized distributed algorithm, but no deterministic log-time algorithm is known to work for general graphs. Another interesting question is whether maximum weighted matching can be approximated to within a factor of $1 + \epsilon$ for any constant $\epsilon > 0$ in polylogarithmic time using messages of $O(\log n)$ bits.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin. *Network Flows.* Prentice-Hall, Engelwood Cliffs, New Jersey, USA, 1993.

[2] A. Czygrinow, M. Hańćkowiak and E. Szymańska. A fast distributed algorithm for approximating the maximum matching. In *Proc. 12th Ann. European Symp. on Algorithms (ESA)*, pages 252–263, 2004.

[3] J. Edmonds. Path, trees and flowers. *Can. J. Math.* 17, pp. 449–467, 1965.

[4] J. Edmonds. Matching and a polyhedron with 0,l vertices. *J. Res. Nat. Bur. Standards* 69B, pp. 125–130, 1965.

[5] J.-H. Hoepman. Simple Distributed Weighted Matchings CoRR cs.DC/0410047, 2004.

[6] J.-H. Hoepman, S. Kutten and Z. Lotker. Efficient Distributed Weighted Matchings on Trees. In *Proc. SIROCCO 2006*, pages 115–129.

[7] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

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

[9] F. Kuhn, T. Moscibroda and R. Wattenhofer. The price of being near-sighted. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–989, 2005.

[10] Z. Lotker and B. Patt-Shamir and Seth Pettie, Improved distributed approximate matching, In *Proc. 22th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 129–136, 2008.

[11] Z. Lotker and B. Patt-Shamir and Adi Rosén, Distributed approximate matching, In *Proc. 26th Ann. ACM symposium on Principles of distributed computing (PODC)*, pages 167–174, 2007.

[12] L. Lovász and M. D. Plummer, *Matching Theory.* North Holland, 1986.

[13] D. Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[14] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. In *Proc. 18th International Conference on Distributed Computing (DISC)*, pages 335–348, 2004.