Improved Distributed Approximate Matching

Zvi Lotker
Department of Communication Systems Engineering
Ben Gurion University
Beer Sheva 84105, Israel
zvilo@cse.bgu.ac.il

Boaz Patt-Shamir*
School of Electrical Engineering
Tel Aviv University
Tel Aviv 69978, Israel.
boaz@eng.tau.ac.il

Seth Pettie
Dept. of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48104 USA
pettie@umich.edu

ABSTRACT

We present improved algorithms for finding approximately optimal matchings in both weighted and unweighted graphs. For unweighted graphs, we give an algorithm providing $(1-\epsilon)$-approximation in $O(\log n)$ time for any constant $\epsilon > 0$. This result improves on the classical $\frac{1}{2}$-approximation due to Israeli and Itai. As a by-product, we also provide an improved algorithm for unweighted matchings in bipartite graphs. In the context of *weighted* graphs, we give another algorithm which provides $(\frac{1}{2}-\epsilon)$ approximation in general graphs in $O(\log n)$ time. The latter result improves on the known $(\frac{1}{4}-\epsilon)$-approximation in $O(\log n)$ time.

Categories and Subject Descriptors

F.1.2 [**Modes of Computation**]: Parallelism and concurrency; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms

Algorithms, Theory

Keywords

Matching

1. INTRODUCTION

Computing a set of disjoint links in a graph, also called a *matching*, is one of the fundamental tasks in distributed computing. 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. In some systems, for example, a node may be engaged in a "conversation" with only one other node at a time, and having a large cardinality matching (over all potential conversations) increases overall communication throughput of the system. An important example is internal scheduling of a communication switch: The basic task of a switch is to transfer packets from input-port buffers to output-port buffers through an internal network called "switch fabric." In each cycle, the switch fabric can realize one partial permutation, and an internal scheduling routine decides which ports will be connected in each cycle. To increase throughput, the scheduling routine tries to find the largest possible matching between the input ports and the output ports. Note that in this important case (as well as numerous other applications), the underlying graph for the matching problem is bipartite.

One significant generalization of the matching problem is the case where edges of the graph have intrinsic positive "weights," and matching of the largest possible weight is sought. For example, in the switch model, packets may have weights representing their importance (or their utility), and the goal is to find a set of disjoint edges (packets) whose sum of weights is as large as possible.

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 settings (see below). Regarding distributed algorithms, it is easy to see that finding an optimal solution, even for unweighted graphs, may require time proportional to the diameter of the graph, and therefore algorithmic research seeks fast approximations. To discuss approximations, it is convenient to introduce some notation first. Let δ-MCM 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 optimum for that graph). Similarly, let δ-MWM 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 $\frac{1}{2}$-MCM or $\frac{1}{2}$-MWM. Approximation factor $\frac{1}{2}$ can be acheived in linear time in the centralized computational model [25, 6].

Even such seemingly trivial guarantees are not very easy to obtain in the distributed setting, where time is dominated by the distance that information has to travel. In the distributed model, the basic result is a randomized $\frac{1}{2}$-MCM algorithm, presented by Israeli and Itai in 1986 [15]. The expected (and high-probability) running time of the algorithm in [15] is $O(\log n)$, where n is the number of nodes in the graph. Ideas from the algorithm of [15] are the basis

*Supported in part by the Israel Science Foundation, grant 664/05.

SPAA'08, June 14–16, 2008, Munich, Germany.

of the PIM algorithm used in the high-speed AN2 switch of DEC [3]. PIM was later refined to iSLIP [23], 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 the algorithm of [15]. To date, the algorithm of [15] has not been completely subsumed. Building on [9], Czygrinow et al. [5] present a deterministic algorithm that computes a $\frac{2}{3}$-MCM in $O(\log^4 n)$ time. Czygrinow and Hańćkowiak [4] have shown that in bipartite graphs, a $(1-\epsilon)$-MCM can be computed deterministically in $(\log n)^{O(1/\epsilon)}$ time.

For weighted graphs, Lotker et al. [18] give a randomized algorithm for $(\frac{1}{4}-\epsilon)$-MWM using $O(\log n)$ time (expected and with high probability), and based the approach of Wattenhofer and Wattenhofer [26], they explain how to get $(\frac{1}{2}-\epsilon)$-MWM in $O(\log^2 n)$ time. In the special case of trees, Hoepman et al. [12] showed that a $(\frac{1}{2}-\epsilon)$-MCM can be computed in expected constant time [12]. Hoepman [11] has shown that a $\frac{1}{2}$-MWM can be computed deterministically in $O(n)$ time.

On the negative side, Kuhn et al. [17] proved that any distributed algorithm, randomized or deterministic, requires $\Omega(\sqrt{\log n/\log\log n})$ (expected) time to compute a $\Theta(1)$-approximate maximum cardinality matching. The lower bound holds even if messages have unbounded size, so long as messages may progress only one hop at each time step.

Our results.

In this paper we give substantially improved distributed algorithms for approximate maximum matchings. Building on some known techniques [13, 8, 24, 18], 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. Finally, we give a randomized reduction from general graphs to bipartite graphs. (The running time of the last algorithm is exponential in $1/\epsilon$.) 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 $O(\log\frac{1}{\epsilon}\cdot T)$. Using the $(\frac{1}{4}-\epsilon)$-MWM algorithm of [18], we obtain the result.

More related work.

Maximum matching is a classic optimization problem that was the target of extensive research for both the bipartite and general cases (see [19] for a fascinating account). For the bipartite graph case, let us mention the $O(|V|^{2.5})$ algorithm by Hopcroft and Karp [13] (see [19] for classical results and [21, 10] for some recent work on the problem). There are a few PRAM algorithms for maximum matching. For example, Karp et al. [16] give a randomized NC algorithm for maximum cardinality matching, and Fischer et al. [8] 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. Hougardy and Vinkemeier [14] have recently extended the algorithm from [8] to the weighted case, with similar processor and time bounds. Drake and Hougardy [7] and Pettie and Sanders [24] have developed linear time sequential algorithms for $(\frac{2}{3}-\epsilon)$-MWM.

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

Execution model.

Throughout the paper 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 model where message size may be unbounded, and the model where messages contain $O(\log n)$ bits. See [22] for more details.

Graphs, matchings etc.

The input is an undirected graph $G = (V, E)$, which may be associated with a weight function $w : E \to \mathbb{R}^+$. We use n to denote $|V|$, and Δ to denote the maximal node degree in G. Throughout the paper, $M \subseteq E$ denotes a matching, and M^* the matching of maximum cardinality (or weight). A vertex is *free w.r.t.* M if it is not incident to any edge in M. An *augmenting path* is a simple path whose endpoints are free and whose edges alternate between $E\backslash M$ and M. For sets A and B, we denote $A \oplus B \stackrel{\text{def}}{=} (A \cup B)\backslash(A \cap B)$.

3. UNWEIGHTED MATCHINGS

In this section we present a $(1-\epsilon)$-approximation for maximal 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 of potentially linear size; in Section 3.2 we show how to reduce the message size to logarithmic in the case of bipartite graph; and finally, in Section 3.3, we give a randomized reduction of general graphs to bipartite graphs. Each of the three algorithms is of possible independent interest.

3.1 Generic Algorithm

In this subsection we give a generic algorithm we use later, and prove the following concrete result.

THEOREM 3.1. *A $(1-\epsilon)$-MCM of any undirected graph can be computed distributively in $O(\epsilon^{-3}\log n)$ communication rounds, with high probability. The maximum message length is $O(|V|+|E|)$.*

The algorithm proceeds in phases as follows. In phase ℓ, we identify a maximal set of augmenting paths of length $2\ell-1$ and apply them to obtain the matching used in phase $\ell+1$. It is shown that after phase ℓ, our matching M is a $(1-\frac{1}{\ell+1})$-approximation. The only part of the algorithm which is slightly less standard is how to efficiently identify

a maximal set of non-conflicting augmenting paths. To do that, we construct a certain "conflict graph," and run a maximal independent set (MIS) algorithm on that graph. (A similar PRAM algorithm, using $n^{O(1/\epsilon)}$ processors, appears in [8].)

We now specify the algorithm in detail. First, we formalize the concept of conflict graphs.

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 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 iff their corresponding augmenting paths intersect at a node of G.*

Algorithm 1 contains pseudo-code of an abstract algorithm that uses conflict graphs. The key to its correctness is 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 abstract algorithm is implemented over the physical graph G as follows. The conflict graph construction (Step 4 of Algorithm 1) is implemented by Algorithm 2. The idea is to explore G to distance ℓ, and use a deterministic rule for assigning augmenting paths to nodes. The messages of Algorithm 2 contain descriptions of portions of the graph, so message size is bounded by $O(|G|) = O(|V| + |E|)$. Using Algorithm 2, we have the following immediate result.

LEMMA 3.2. *Step 4 of Algorithm 1 can be implemented in $O(\ell)$ time using $O(|V| + |E|)$-bit messages.*

After Step 4, each node P in $C_M(\ell)$ (P is an augmenting path) is associated with a physical node leader$(P) = v \in V$, and furthermore, each node $v \in V$, knows where are its "neighbors" in the conflict graph. More formally, if $v =$ leader(P), $u =$ 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 from P to P', say, it can be done over G. Therefore, Step 5 can be implemented efficiently by any distributed MIS algorithm, where messages between $C_M(\ell)$ nodes are routed between their leaders, along their augmenting paths. More formally, we have the following result.

LEMMA 3.3. *Step 5 of Algorithm 1 can be implemented in $O(\ell^2 \log n)$ time (with high probability).*

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 $C_M(\ell)$
5: Let $\mathcal{I}$ be an MIS 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*

Proof: We use either the algorithm of [20] or [1] to do the MIS computation. The distributed version of these algorithm runs in time $O(\log N)$ (w.h.p.), where N is the number of nodes in the graph. In our case, we apply MIS to $C_M(\ell)$, and the number of nodes in $C_M(\ell)$ is $n^{O(\ell)}$. Furthermore, each step of the MIS computation in $C_M(\ell)$ is implemented by $O(\ell)$ routing steps in G. ∎

To prove Theorem 3.1, we use the following two facts from Hopcroft and Karp [13].

LEMMA 3.4. *If the shortest augmenting path w.r.t. M has length ℓ and P is a maximal set of augmenting paths of length ℓ, the shortest augmenting path w.r.t. $M \oplus P$ has length greater than ℓ.*

LEMMA 3.5. *If the shortest augmenting path w.r.t. M has length $2k - 1 \geq 1$ then $|M| \geq (1 - \frac{1}{k})|M^*|$.*

Proof of Theorem 3.1: Let us first bound the high-probability time complexity of Algorithm 1 when Step 4 is implemented by Algorithm 2 and when Step 5 is implemented by the algorithm of [1]. There are $O(\epsilon^{-1})$ iterations. Steps 4 and 5 take $O(\epsilon^{-2} \log n)$ time w.h.p. by Lemmas 3.2 and 3.3. (There are only polynomially many MIS invocations, so we can apply the union bound to conclude that with high probability, all invocations complete in time.) Step 7 is implemented distributively by letting leader nodes send messages along the paths chosen to the MIS in Step 5. This takes additional $O(\ell)$ time, so the time bound follows.

The correctness of the algorithm is proved as follows. Inductive application of Lemma 3.4 shows that after iteration $2k-1$ of Algorithm 1, the length of the shortest augmenting path in G is more than $2k - 1$, and therefore, by Lemma 3.5, when the algorithm terminates, the size of M is at least $(1 - \frac{1}{k}) = (1 - \epsilon)$ times the optimum. The theorem follows. ∎

3.2 Bipartite Graphs

In this section we show how to reduce the message size to $O(\text{poly}(\frac{1}{\epsilon}) \log n)$ in the case of bipartite graphs. The idea is to avoid constructing the conflict graph explicitly: In bipartite graphs, MIS computation can be done while constructing the graph "on the fly."

The key is that it is sufficient for each node know the number of augmenting paths it is a member of. This is done

Algorithm 2 Implementation of Step 4 of Algorithm 1 for general graphs at a 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 $G_v(2\ell)$.
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).

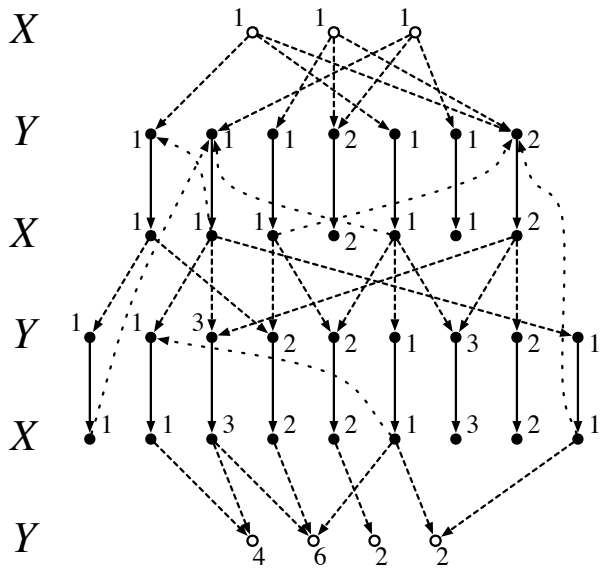

Figure 1: Illustration of Algorithm 3. Hollow and filled circles represent unmatched and matched vertices, resp. Dashed and solid arcs represent unmatched and matched edges, resp. The algorithm starts at the top layer and progresses one layer at a time. Numbers next to nodes are the sum of numbers received from the previous level.

as follows (see Figure 1 for an example, and Algorithm 3 for pseudo-code). Suppose the nodes of the bipartite graph are $X \cup Y$ where $E \subseteq X \times Y$. We count the number of augmenting paths of length ℓ by initiating breadth-first search (BFS) from all unmatched 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 1). 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 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 indicates an augmenting path that intersects a shorter one. This forwarding procedure is executed ℓ rounds. In the last round is slightly different: X nodes send their value to all their neighbors as usual, but in this case, only *unmatched* Y nodes react to messages by recording their values and origins.

Call a sequence of edges a *half-augmenting path* if it starts with a free X node and alternates between unmatched and matched edges, with no restrictions on the last edge. Let $d(v)$ be the length of the shortest half-augmenting path ending at v.

LEMMA 3.6. *Suppose that G and M are such that there are no augmenting paths shorter than ℓ. Consider a node v which receives its first messages of Algorithm 3 at time $d(v)$. Then there are $n_v \stackrel{\text{def}}{=} \sum_{i=1}^{\deg(v)} c_v[i]$ half-augmenting paths of length $d(v)$ that end at v, and furthermore, $n_v \leq \Delta^{\lceil d(v)/2 \rceil}$.*

Algorithm 3 At node v: Counting the number augmenting paths of length ℓ in bipartite graphs.

```
1: c_v[i] ← 0 for all 1 ≤ i ≤ deg(v)
                ▷ c_v[i] is the number of paths from edge i at v
2: if v ∈ X and v is not matched then
3:    send 1 to all neighbors and halt
4: end if
5: wait until a message is received (at time d(v))
6: c_v[i] ← the number received on edge i
7: n_v ← Σ_{i=1}^{deg(v)} c_v[i]
8: if v ∈ X then
9:    send n_v to all neighbors and halt
10: end if
11: if v ∈ Y and matched then
12:    send n_v to mate and halt
13: end if
```

Proof Sketch: The proof is by induction on the distance from X. The base case $d(v) = 0$ (i.e., v is a free X node) follows from lines 2–3. For the inductive step, let $d(v) = t+1$. If t is odd, then at time t, only Y nodes send, and only along matching edges, and the claim holds by induction and the definition of half-aumenting paths. If t is even, X nodes send messages, and again the lemma holds by induction, the definition of half-aumenting paths, and the assignments of Line 6. It is not difficult to see that messages arriving after time t correspond to non-shortest augmenting paths, and thus need not be counted.

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

It follows from Lemma 3.6 that, 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 use the following MIS algorithm [20]: Let each node y be the leader of n_y paths (as determined by Algorithm 3). The MIS algorithm works in iterations, where in each iteration each node in the conflict graph (i.e., each augmenting path of length ℓ) chooses a random number, and it is added to the MIS iff 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 $n\Delta^{(\ell+1)/2}$. After at most $O(\log N)$ iterations of this procedure, an MIS is found with high probability.

We emulate an iteration distributively as follows. First, instead of choosing a number for each path, we let each leader v choose a single number which is the "winner" of all paths which v leads (this number is chosen according to the appropriate distribution), and then choose the winning path by constructing it inductively, working backwards (bottom-up in Figure 1) as follows. The first node of the path is the leader node. If we are at a node $y \in Y$, the next edge is a non-matching edge i, chosen randomly with probability

$c_y[i]/n_y$ (using the c array computed by Algorithm 3). If we are at a node $x \in X$, we just follows the unique matching edge incident with x.

In the algorithm, all leaders y send a token message carrying their chosen number w_y. The path traversed by the token is selected stochastically as described above, and the path is recorded by the nodes. Whenever tokens meet, only the token carrying the largest number w_y continues to construct the path. By the way the paths were constructed, tokens may arrive at a node only at a single round. Finally, when the token arrives at the last node of its path, it traces that path back while augmenting along it, i.e., flipping matching edges to non-matching edges and vice-versa.

We can now prove the following.

LEMMA 3.7. *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.*

Proof: Consider the algorithm described above, where routing the messages is implemented as follows. By Lemma 3.6, 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*, and send the chunks one by one in a pipelining 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, the maximal chunk is 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 3.7 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 3.8. *In bipartite graphs, a $(1-\frac{1}{k})$-approximation to the maximum matching can be found in $O(k^3 \log \Delta + k^2 \log n)$ time using messages of size $O(\log \Delta)$ bits.*

The proof follows from combining Lemma 3.7 within Algorithm 1.

In Algorithm 3 we assumed for simplicity's sake that the shortest augmenting path had length ℓ. With relatively minor modifications the algorithm can find a maximal set of augmenting paths with length *at most* ℓ. This fact is used in the following section.

3.3 Matching in General Graphs Revisited

In this section we give a randomized reduction from general graphs to bipartite graphs. The idea is to repeatedly sample a bipartite subgraph of the original graph and find a maximal set of short augmenting paths in the bipartite subgraph. 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. Here $k > 2$ is any integer. 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 3.4 and 3.5 directly .

Pseudo-code for the algorithm is given below. It uses a subroutine $\text{Aug}(H, M, \ell)$, where M is a matching in a bipartite graph H, which returns a maximal set of disjoint augmenting paths w.r.t. M, each of length at most ℓ.

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 colors itself red or blue with equal prob.
4: Let $\hat{G} \leftarrow (\hat{V}, \hat{E})$, where
$\hat{V} = \{u \mid u \in V \text{ is free, or } (u,v) \in M \text{ is bichromatic}\}$, and
$\hat{E} = \Big\{(u,v) \mid (u,v) \in (E \cap \hat{V} \times \hat{V}) \text{ is bichromatic}\Big\}$.
5: $\mathcal{P} \leftarrow \text{Aug}(\hat{G}, M, 2k-1)$ ▷ *$\mathcal{P}$ is a maximal set of disjoint augmenting paths of length at most $2k-1$*
6: $M \leftarrow M \oplus \mathcal{P}$
7: **end for**
8: **return** M

We start the analysis with two simple observations.

OBSERVATION 3.1. *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. $\hat{G}$ and $\hat{M}$, then P is also an augmenting path w.r.t. G and M.*

OBSERVATION 3.2. *If P is an augmenting path of length ℓ w.r.t. G and M, then $\Pr\left[P \subseteq \hat{E}\right] = 2^{-\ell}$.*

Using Observations 3.1 and 3.2 we can now analyze the overall behavior of the algorithm.

LEMMA 3.9. *Let $\alpha = (1 - \frac{1}{k+1})|M^*| - |M|$. In Line 5 of Algorithm 4, $\Pr\left[|\mathcal{P}| \geq \frac{\alpha}{(k+1)2^{2k}}\right] = 1 - e^{-\Omega(\alpha)}$.*

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 $2k-1$. By Lemma 3.5, $|\mathcal{P}^*| \geq \alpha$. Let $\hat{\mathcal{P}}^* \subseteq \mathcal{P}^*$ be those augmenting paths appearing in $\hat{G}$. Since each augmenting path in $\mathcal{P}^*$ is included in $\hat{\mathcal{P}}^*$ independently with probability at least 2^{-2k+1}, a standard Chernoff bound [2] implies that the probability that $|\hat{\mathcal{P}}^*| < |\mathcal{P}^*|/2^{2k}$ (i.e., half its expectation) is $\exp(-\Omega(\alpha))$. The call to $\text{Aug}(\hat{G}, M, 2k-1)$ finds a maximal set $\mathcal{P}$ of non-conflicting augmenting paths in $\hat{G}$. Each augmenting path in $\mathcal{P}$ can intersect at most $k+1$ paths in $\hat{\mathcal{P}}^*$, which implies $|\mathcal{P}| \geq |\hat{\mathcal{P}}^*|/(k+1)$ and, in turn, that $|\mathcal{P}| \geq |\mathcal{P}^*|/(k+1)2^{2k}$ with probability $1 - \exp(-\Omega(\alpha))$. ∎

LEMMA 3.10. *The matching returned by Algorithm 4 is a $(1 - \frac{1}{k})$-MCM with high probability.*

Proof: Let δ_i be the gap between $|M|$ and $(1 - \frac{1}{k+1})|M^*|$ after i iterations of Line 2. Thus, $\delta_0 = (1 - \frac{1}{k+1})|M^*|$. Under the assumption that $\delta_i > \frac{1}{k(k+1)}|M^*|$, i.e., that M is not

already a $(1-\frac{1}{k})$-approximation, it follows from Lemma 3.9 that $\delta_{i+1} \leq \left(1-\frac{1}{(k+1)2^{2k}}\right)\delta_i$ 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 complexity of Algorithm 4 is dominated by the call to $\mathrm{Aug}(\hat{G}, M, 2k-1)$, which is implemented by the algorithm of Section 3.2, and hence, by repeated application of Lemma 3.7, $O(k^3\log n)$ time with $O(\log n)$-bits messages suffice. As a corollary, we arrive at our main result of this section.

THEOREM 3.11. *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 $O(\log\frac{1}{\epsilon}\log n)$. The algorithm is based on a reduction to any δ-MWM algorithm with $\delta>0$ and running time $O(\log n)$. Plugging in the algorithm of [18], the result follows.

Preliminaries.

For an edge (u,v) in a matching M, let $M(u)\stackrel{\text{def}}{=}v$. For any edge $(r,s)\in E\setminus M$ let $\mathrm{wrap}(r,s)$ be the path consisting of the edges $(M(r),r)$, (r,s), and $(s,M(s))$. Each of the edges $(M(r),r)$ and $(s,M(s))$ may not exist. We emphasize that $\mathrm{wrap}(e)$ is always defined with respect to the matching M. Given a set $\mathcal{P}$ of augmenting paths w.r.t. a matching M, define

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

i.e., $g(\mathcal{P})$ is the resulting *gain* in weight if $\mathcal{P}$ were applied to M.

Algorithm.

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

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

More 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 2 for an example.

The idea of the algorithm (see Algorithm 5 below) 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. 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.

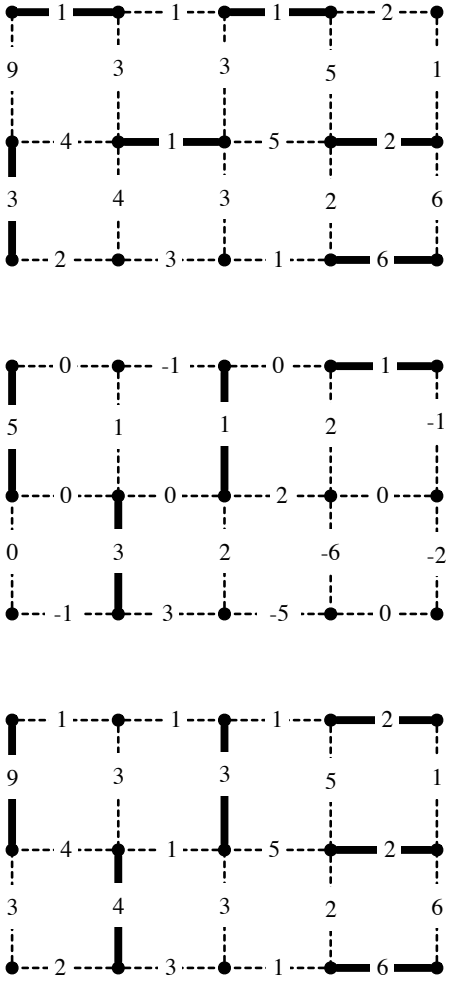

Figure 2: Top: a matching M (bold edges matched, dashed edges unmatched) with weight 14 under the original weight function w. Middle: a matching M' with weight 10 under the weight function w_M. Bottom: the matching $M''=M\oplus\bigcup_{e\in M'}\mathrm{wrap}(e)$, having weight $w(M'')=26\geq w(M)+w_M(M')$.

Analysis.

We first prove that after each iteration, M is a matching of increased weight. Formally:

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

Proof: If M'' is not a matching then it must contain two

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

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

adjacent edges $f \in M$ and $e \in \text{wrap}(e')$ for some $e' \in M'$, but then, $f \in \text{wrap}(e')$, a contradiction to $f \in M''$. Turning to the second part, we have the 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 2 for an example. ∎

We use the following fact (recall that M^* denotes an optimal matching).

LEMMA 4.2. **[24]** *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, 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 $w_{M_{i-1}}(\mathcal{P}\backslash M_{i-1}) = g(\mathcal{P})$. 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 recall the following fact.

LEMMA 4.4. **[18]** *For any $\epsilon>0$, $(\frac{1}{4}-\epsilon)$-MWM can be computed in $O(\epsilon^{-1}\log\epsilon^{-1}\log n)$ time (w.h.p.).*

Now we conclude with the following theorem.

THEOREM 4.5. *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 [18] in Line 4 with $\delta = 1/5$. Since in constant time we can find 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. ∎

Remark: We note that $(1-\epsilon)$-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 [14] to the distributed setting using Algorithm 2. Details are omitted from this extended abstract.

5. CONCLUSION

In this paper 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. For unweighted graphs, it is interesting to see whether there exists a $(1-\epsilon)$-approximation for general graphs with logarithmic message size and time complexity $O(\text{poly}(\epsilon^{-1})\log n)$. And of course, the long-standing open questions: can maximal matching and independent set be computed deterministically in $O(\log n)$ time on general graphs?

6. REFERENCES

[1] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. of Algorithms*, 7:567–583, 1986.

[2] N. Alon and J. H. Spencer. The Probabilistic Method, John Wiley and Sons, New York, 1992.

[3] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, High speed switch scheduling for local area networks. *ACM Trans. Comput. Syst.* 11(4):319–352, Nov. 1993.

[4] A. Czygrinow and M. Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. Proceedings 9th Annual Int'l Conference on Computing and Combinatorics (COCOON), pages 242–251, 2003.

[5] 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.

[6] D. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. Information Processing Letters 85, pp. 211–213, 2003.

[7] D. Drake and S. Hougardy. Improved linear time approximation algorithms for weighted matchings. Proc. 7th Int. Workshop on Randomization and Approximation Techniques in Computer Science (APPROX), pp. 14–23, 2003.

[8] T. Fischer, A. V. Goldberg, D. J. Haglin, and S. Plotkin. Approximating matchings in parallel. *Info. Proc. Lett.*, 46(3):115, 1993.

[9] M. Hańćkowiak, M. Karoński, and A. Panconesi. A faster distributed algorithm for computing maximal matchings deterministically. Proc. 18th Ann. ACM Symp. on Principles of Distributed Computing (PODC), pp. 219–228, 1999.

[10] N. Harvey. Algebraic structures and algorithms for matching and matroid problems. Proceedings 47th Symposium on Foundations of Computer Science (FOCS), pp. 531–542, 2006.

[11] J.-H. Hoepman. Simple distributed weighted matchings CoRR cs.DC/0410047, 2004.

[12] J.-H. Hoepman, S. Kutten and Z. Lotker. Efficient distributed weighted matchings on trees. In *Proc. SIROCCO 2006*, pages 115–129.

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

[14] S. Hougardy and D. Vinkemeier. Approximating weighted matchings in parallel. *Info. Proc. Lett.*, 99(3):119–123, 2006.

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

[16] R. M. Karp, E. Upfal and A. Wigderson. Constructing a perfect matching is in RandomNC. In *Proc. 17th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 22–32, 1985.

[17] 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.

[18] Z. Lotker, B. Patt-Shamir and A. Rosén. Distributed approximate matching. In *Proc. 26 ACM Ann. Symp. on Principles of Distributed Computing*, pages 167–174, 2007.

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

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

[21] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. Proc. 45th Symp. on Foundations of Computer Science (FOCS), 248–255, 2004.

[22] D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, Philadelphia, PA, USA, 2000.

[23] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking* 7(2):188–201, April 1999.

[24] S. Pettie and P. Sanders. A simpler linear time $2/3 - \epsilon$ approximation to maximum weight matching. Information Processing Letters 91(6):271–276, 2004.

[25] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *Proc. 16th Ann. Symp. on Theoretical Aspects of Computer Science (STACS)*, LNCS 1563, 259–269, 1999.

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