Distributed weighted matching

Distributed Weighted Matching

Mirjam Wattenhofer, Roger Wattenhofer
{mirjam.wattenhofer,wattenhofer}@inf.ethz.ch
Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland

Abstract

In this paper, we present fast distributed approximation algorithms for matching in weighted trees and general weighted graphs. The time complexity as well as the approximation ratio of the tree algorithm is constant. For the general graph algorithm we prove a constant ratio bound and a polylogarithmic time complexity of $O(\log^2 n)$.

Keywords: parallel and distributed algorithms, maximum weighted matching, approximation algorithms

1 Introduction and Related Work

In a weighted graph, a maximum weighted matching is a subset of edges such that no two edges have a common vertex and the weight of the matching is maximized. Matching is one of the most fundamental problems studied in graph theory and computer science. A plethora of algorithms and heuristics for different matching variants have been proposed, culminating in the breakthrough work of Edmonds [Edm65] who has showed that the maximum weighted matching problem can be computed in polynomial time for general graphs.

The increasing importance of large-scale networks (e.g. Internet, ad-hoc and sensor networks) has shifted the focus of distributed computing research away from tightly-coupled multiprocessors towards loosely-coupled message passing systems. Solving a basic network problem such as matching by first collecting the graph topology of the network and then computing an optimal solution using Edmonds' algorithm is not economical because this approach leads to an immense data flow which is expensive in time and resources. Moreover, by the time the solution is computed, the network topology may already have changed.

In this paper we adopt the model of so-called *local graph (or network) algorithms*. Instead of sending the input (the network topology as a weighted graph) to a central processor, we let all the vertices of the network participate in the computation themselves. By only allowing the vertices to communicate with their direct neighbors in the graph, we keep the locality of the original problem.[1]

Related Work The study of local graph algorithms dates back to the ingenious $O(\log^* n)$ coloring algorithm by Cole and Vishkin [CV86] and the matching lower bound by Linial [Lin92]. Thanks to the applications for ad-hoc and sensor networks local graph algorithms have recently been a field of intensive study ([JRS01], [KW03]). The model will be presented in detail in Section 2; for a proficient introduction to local distributed computing, we refer to [Pel00].

Up to now, only a small number of distributed algorithms for matching have been proposed. Indeed, we are not aware of any distributed algorithm that solves the maximum weighted matching problem optimally. Distributed computing researchers often cherish time more than quality, and consequently prefer approximation algorithms that only need polylogarithmic time over optimal linear time algorithms. To our knowledge, there exists just one maximum weighted matching approximation: Uehara and Chen [UC00] present a constant-time algorithm that achieves a $O(\Delta)$ approximation, Δ being the maximum degree in

[1]In contrast, the widely studied parallel random access machine (PRAM) model does not preserve locality. In essence, a local graph algorithm is also a PRAM algorithm, but not vice versa.

the graph. In this paper we significantly improve the approximation ratio while staying polylogarithmic in time.

In contrast, there is a whole selection of algorithms studying the special case of *non*-weighted graphs. For non-weighted graphs Karaata and Saleh [KS00] give a $O(n^4)$ algorithm that solves the maximum matching problem for trees. In bipartite graphs where the vertices know their partition Chattopadhyay et al. [CHS02] give an algorithm for the same problem with time complexity $O(n^2)$. In the same paper Chattopadhyay et al. study the maximal matching problem for general graphs, presenting a linear-time algorithm. This second result is in fact a deterioration[2] of an earlier paper by Israeli and Itai [II86]; Israeli and Itai already give a randomized[3] $O(\log n)$ time algorithm for the maximal matching problem. The methods Israeli and Itai use are similar to those used by Luby [Lub86] for the related maximal independent set problem. In Section 4 we will use methods inspired by [Lub86] and [II86] to achieve our results.

Outline After this excursion to non-weighted graphs let us now return to weighted graphs. In our paper we present two randomized algorithms for approximating a maximum weighted matching, first one for trees, then one for general weighted graphs. The tree algorithm in Section 3 finds a $O(1)$-approximation in $O(1)$-time. The graph algorithm in Section 4 computes a $O(1)$-approximation in $O(\log^2 n)$-time. Beforehand—in Section 2—we formally introduce the model.

2 Notation and Preliminaries

In this section we introduce the notation as well as some basic theorems we will use throughout the paper.

Let $G = (V, E)$ be an undirected simple graph; as usual V the set of vertices, E the set of edges and $|V| = n$. With each edge $e \in E$ we associate a positive weight $w(e) \in \mathbf{R}^+$. For a subset S of E, $w(S)$ denotes the total weight of the edges in S, that is $w(S) = \sum_{e \in S} w(e)$. A set $M \subseteq E$ is a *matching* if no two edges in M have a common vertex. A matching is *maximal* if it is not properly contained in any other matching, it is a *maximum (cardinality)* matching if its *size* is maximized over all matchings in G. A matching is a *maximum weighted* or *optimal* matching of G if its *weight* is maximized over all matchings in G. Throughout the paper M^*_G will denote a maximum weighted matching in a graph G and M_G a matching computed by our algorithm. Following the standard notation we say that our algorithm has a *ratio bound* of ρ if $w(M_G)$ is within a factor of ρ of $w(M^*_G)$. Let $d_G(u)$ denote the degree of vertex u in G. We will make use of the maximum weight in the entire graph G, respectively in the neighborhood of a vertex u. For this purpose we define $w_{\max}(G)$ and $w_{\max}(u)_E$:

$$w_{\max}(G) := \max_{e \in E} w(e),$$
$$w_{\max}(u)_E := \max_{e \in E; e = \{u,x\}} w(e).$$

Whenever a vertex has to choose an edge with weight $w_{\max}(u)_E$ and there is more than one candidate, ties are broken lexicographically by choosing the edge with highest rank in a given ordering.

Though our tree-algorithm works for non-rooted trees, it simplifies some proofs to assume that there is a root. In this case, the terms *parent, child* and *sibling* have their *familiar* meaning. We define $n_{in}(T)$ for a tree $T = (V, E)$ as the number of interior (non-leaf) vertices:

$$n_{in}(T) := |\{u \mid u \in V, d_T(u) > 1\}| .$$

We use a purely synchronous model for communication. That is, in every communication round, each vertex is allowed to send a message to each of its direct neighbors. In our algorithms all messages need a constant number of bits only. The *time complexity* is the number of rounds the algorithms needs to solve the problem.

[2]Note that [CHS02] focus on self-stabilization; however, as they remark themselves in the extended version of the paper, self-stabilization essentially comes for free in the context of matching.

[3]It is worth noting that Hanckowiak et al. [HKP01] manage to give a $O(\log^4)$ time deterministic algorithm for the same model.

The section is concluded by giving four facts which will be used in subsequent sections. For a proof, we refer the reader to standard mathematical text books.

Fact 2.1. *For $n \geq x \geq 1$, we have*

$$\left(1 - \frac{x}{n}\right)^n \leq e^{-x}.$$

Fact 2.2. *In a graph $G = (V, E)$, we have*

$$|E| = \frac{1}{2} \cdot \sum_{u \in V} d_G(u).$$

Fact 2.3. *For any matching M on a graph $G = (V, E)$, it holds that $|M| \leq \frac{1}{2} \cdot |V|$.*

Fact 2.4. *If M^* is a maximum (cardinality) matching and M is a maximal matching then $|M^*| \leq 2 \cdot |M|$.*

3 Matching in Trees

3.1 The Algorithm

In this section we present a distributed algorithm for approximating a maximum weighted matching in (non-rooted) trees. The time complexity as well as the ratio bound of this algorithm are shown to be constant. For the sake of clarity and simplicity the algorithm is divided into three sub-algorithms, which are to be executed subsequently. Before we present the algorithm in detail we give a general overview.

Outline of the algorithm: To find a matching M_T in a weighted tree $T = (V, E)$ each vertex u chooses its heaviest incident edge. The chosen edges induce a forest F on T. If $d_F(u) > 2$, u deletes all incident edges in F except the two heaviest. By this construction we get a set of paths P on which we finally compute the matching. For details see Algorithm 1 [4] (*Tree-Matching*).

Algorithm 1 Tree-Matching ($T \to M_T$), executed by vertex u

```
Input: tree $T$
Output: matching $M_T$
1: (∗ See Footnote 4 ∗)
2: $F$ := Algorithm 2 (Forest), with Input $T$
3: $P$ := Algorithm 3 (Paths), with Input $F$
4: $M_T$ := Algorithm 4 (Matching), with Input $P$
```

Algorithm 2 Forest ($T \to F$), executed by vertex u

```
Input: tree $T = (V, E)$
Output: forest $F = (V_F, E_F)$
1: $V_F := V$, $E_F := \emptyset$
2: choose an edge $e = \{u, v\}$, $e \in E$, with $w(e) = w_{\max}(u)_E$
3: send message “you are my neighbor in $F$” to $v$,
   $E_F := E_F \cup \{e\}$
4: receive message from all neighbors $w$
5: if (received message “you are my neighbor” from $w$) then
6:     $E_F := E_F \cup \{u, w\}$
7: fi;
8: (∗ $F$ is a forest, since it is a subgraph of a tree. $d_F(u) \geq 1$ ∗)
```

[4] For the sake of readability, in all our algorithms we always use the global variables—like T—but of course, the vertices actually do not know (and need) the global variable—the entire graph T— but only their direct neighborhood in this global variable.

Algorithm 3 Paths $(F \to P)$, executed by vertex u

```
Input: forest $F = (V_F, E_F)$
Output: set of paths $P = (V_P, E_P)$
1: $V_P := V_F$, $E_P := E_F$
2: if ($d_F(u) > 2$) then
3:     choose two edges $e_1 = \{u, v_1\},\ e_2 = \{u, v_2\}$, $\{e_1, e_2\} \in E_F$,
       with $w(e_1) = w_{\max}(u)_{E_F}$, $w(e_2) = w_{\max}(u)_{(E_F \setminus \{e_1\})}$
4: fi
5: send message "edge $e_i$ is deleted in $P$" to all neighbors $v_i$, $i \in \{3, \dots, d_F(u)\}$,
   $E_P := E_P \setminus \bigcup_{i \in \{3,\dots,d_F(u)\}} \{e_i\}$
6: receive message from all neighbors
7: (* $d_P(u) \leq 2$ *)
```

Algorithm 4 Matching $(P \to M_T)$, executed by vertex u

```
Input: set of paths $P$
Output: matching $M_T$
1: $M_T := \emptyset$
2: choose uniformly at random one edge $e = \{u, v\}$, $e \in E_P$
3: send message "you are my matching partner" to $v$
4: receive messages from all neighbors
5: if (received message from $v$) then
6:     $M_T := M_T \cup \{e\}$
7: fi
8: (* An edge is part of the matching $M_T$ if both its incident vertices have chosen it. Therefore, $M_T$ is
   a valid matching. *)
```

3.2 Analysis

In the following, we will analyze the Tree-Matching algorithm given above. First with respect to the weight of the matching it finds and then regarding its running time and message complexity. If not stated otherwise, $T = (V, E)$ is the tree on which we like to find a matching, with $|V| = n$. $F = \bigcup_{i:T_i \in F} T_i$ is the forest induced on T by the Forest Algorithm and $P = \bigcup_{i:T_i \in F} P(T_i)$ is the set of paths we get by running the Paths Algorithm on F.

To prove the constant ratio bound we first present several important key lemmas which together result in the desired property.

Lemma 3.1. ***(First Key Lemma,* T → F*)*** *$w(M_T^*) \leq 4 \cdot w(M_F^*)$.*

The proof of the first key lemma is simplified by first proving the next two helper lemmas.

Lemma 3.2. *$w(F) \geq w(M_T^*)$*

Proof. In the Forest Algorithm every vertex $u \in T$ chooses exactly one edge. Hence, the degree of u in F is at least one. Together with Fact 2.4 we have

$$2\,|E_F| = \sum_{u \in V} d_F(u) \geq \sum_{u \in V} 1.$$

On the other hand,

$$2\,|M_T^*| = \sum_{u \in V} d_{M_T^*}(u) \leq \sum_{u \in V} 1.$$

Therefore, $|E_F| \geq |M_T^*|$. Since in the algorithm each vertex chooses its *heaviest* incident edge we may further conclude that $w(F) \geq w(M_T^*)$. □

Lemma 3.3. $w(M_F^*) \geq \frac{1}{4}w(F)$

Proof. Using the Third Key Lemma 3.11 we know that if we execute Algorithm 4(*Matching*) with input F then $E[w(M_F)] \geq \frac{1}{4}w(F)$. If the *expected* weight of a matching is at least $\frac{1}{4}w(F)$ then there must be a matching with weight at least $\frac{1}{4}w(F)$ and therefore the optimal matching must have weight at least $w(M_F^*) \geq \frac{1}{4}w(F)$. □

Proof. **(Proof of First Key Lemma 3.1)** By Lemma 3.3 and Lemma 3.2 we have

$$w(M_F^*) \geq \frac{1}{4}w(F) \geq \frac{1}{4}w(M_T^*).$$

□

Lemma 3.4. ***(Second Key Lemma,* F → P*)*** $w(P) \geq w(M_F^*)$.

In order to prove the second key lemma we need the following helper lemmas. In those lemmas we argue on each tree T_i in the forest F separately.

Lemma 3.5. *In a tree $T_i = (V_i, E_i)$ with more than two vertices, $\left|M_{T_i}^*\right| \leq n_{in}(T_i)$.*

Proof. In a tree with more than two vertices a leaf cannot be adjacent to another leaf. Therefore, if $e = \{u, v\} \in M_{T_i}^*$ and $d_{T_i}(u) = 1$ it holds that $d_{T_i}(v) > 1$ and

$$\sum_{u \in V_i, d_{T_i}(u)=1} d_{M_{T_i}^*}(u) \leq \sum_{u \in V_i, d_{T_i}(u)>1} d_{M_{T_i}^*}(u).$$

We get

$$\begin{aligned}
\left|M_{T_i}^*\right| &= \frac{1}{2}\sum_{u \in V_i} d_{M_{T_i}^*}(u) && \text{(Fact 2.2)}\\
&= \frac{1}{2}\sum_{u \in V_i, d_{T_i}(u)=1} d_{M_{T_i}^*}(u) + \frac{1}{2}\sum_{u \in V_i, d_{T_i}(u)>1} d_{M_{T_i}^*}(u)\\
&\leq \sum_{u \in V_i, d_{T_i}(u)>1} d_{M_{T_i}^*}(u)\\
&\leq \sum_{u \in V_i, d_{T_i}(u)>1} 1 = |\{u \mid u \in V_i, d_{T_i}(u) > 1\}| && \text{(in a matching } d(u) \leq 1)\\
&= n_{in}(T_i).
\end{aligned}$$

□

Lemma 3.6. *Let p_j be a path in $P(T_i)$. Let $u \in p_j$ with $d_{p_j}(u) = 1$ and $v \in p_j$ with $d_{p_j}(v) = 1$. Then either $d_{T_i}(u) = 1$ or $d_{T_i}(v) = 1$.*

Proof. To simplify this proof let us in short describe the inherent structure of a tree $T_i \in F$. Every tree T_i has an unique edge e_r—the root-edge—which was chosen in the Forest Algorithm by both its incident vertices. This is the edge with maximal weight in T_i. All other edges in T_i were chosen by exactly one of their incident vertices. Therefore it holds that for every path from a leaf to the root-edge e_r the weight along this path is monotonically increasing.

In the Paths Algorithm every vertex deletes all its incident edges except the two heaviest. Since the weight from a leaf to the root-edge e_r in T_i is increasing a vertex u never deletes its edge which is nearest to e_r.

Let without loss of generality (w.l.o.g.) vertex u be the vertex further from e_r than v. We show that u is always a leaf in T_i and the lemma is proved. For contradiction, suppose that $d_{T_i}(u) > 1$. Hence, the child-edge $e = \{u, w\}$ u was choosing was deleted by its child w. This is a contradiction to the fact that a vertex does not delete its edge nearest to the root-edge. □

Corollary 3.7. *The number of leaves in T_i which have degree one in $P(T_i)$ is at least as large as the number of leaves in T_i which have degree greater than one in $P(T_i)$. Formally,*

$$\left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) = 1\}\right| \geq \left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) > 1\}\right| .$$

Proof. According to Lemma 3.6 in every path $p_j \in P(T_i)$ there is a vertex u with $d_{p_j}(u) = 1$ and $d_{T_i}(u) = 1$. That is, each path ends in a vertex which is a leaf in T_i and the corollary is proved. □

Lemma 3.8. *In a tree $T_i = (V_i, E_i) \in F$ with more than two vertices, let $P(T_i) = (V_i, E_{P_i})$ be the set of paths induced on T_i by the Paths Algorithm as stated above. It holds that $|E_{P_i}| \geq n_{in}(T_i)$.*

Proof.

$$\begin{aligned}
|E_{P_i}| &= \frac{1}{2}\sum_{v\in V_i} d_{P(T_i)}(v) && \text{(Fact 2.2)}\\
&= \frac{1}{2}\cdot 2\left|\{v \mid v \in V_i, d_{P(T_i)}(v) > 1\}\right| + \frac{1}{2}\left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1\}\right|\\
&= \left|\{v \mid v \in V_i, d_{P(T_i)}(v) > 1\}\right| +\\
&\quad \frac{1}{2}\left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) = 1\}\right| + \frac{1}{2}\left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) > 1\}\right|\\
&\geq \left|\{v \mid v \in V_i, d_{P(T_i)}(v) > 1\}\right| + \left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) > 1\}\right| && \text{(Corollary 3.7)}\\
&= \left|\{v \mid v \in V_i, d_{P(T_i)}(v) > 1, d_{T_i}(v) > 1\}\right| + \left|\{v \mid v \in V_i, d_{P(T_i)}(v) = 1, d_{T_i}(v) > 1\}\right|\\
&= |\{v \mid v \in V_i, d_{T_i}(v) > 1\}| = n_{in}(T_i)
\end{aligned}$$

□

Lemma 3.9. *The number of edges in $P(T_i)$ is at least as large as the size of an optimal matching in T_i, $|E_{P_i}| \geq \left|M^*_{T_i}\right|$.*

Proof. If $n = 2$ then $\left|M^*_{T_i}\right| = 1 = |E_{P_i}|$. For the case $n > 2$ Lemma 3.5 and Lemma 3.8 hold and $|E_{P_i}| \geq n_{in}(T_i) \geq \left|M^*_{T_i}\right|$. □

Lemma 3.10. *$w(M^*_{T_i}) \leq w(P(T_i))$.*

Proof. Suppose $e = \{u, v\} \in M^*_{T_i}$, $e \notin E_{P_i}$. By Lemma 3.9 we can assign to every such edge a distinct edge $e' \in E_{P_i}$. Since the Paths Algorithm always chooses the locally heaviest edge we can guarantee that $w(e') \geq w(e)$, otherwise e would have been chosen instead of e' . □

Proof. **(Proof of Second Key Lemma 3.4)** Since $P = \dot{\bigcup}_{i:T_i\in F} P(T_i)$ and $M^*_F = \dot{\bigcup}_{i:T_i\in F} M^*_{T_i}$ we have

$$\begin{aligned}
w(P) &= \sum_{i:T_i\in F} w(P(T_i))\\
&\geq \sum_{i:T_i\in F} w(M^*_{T_i}), && \text{(Lemma 3.10)}\\
&= w(M^*_F),
\end{aligned}$$

□

Lemma 3.11. ***(Third Key Lemma,*** **P → M*****)*** *If every vertex in P executes the Matching Algorithm then $E[w(M_P)] \geq \frac{1}{4}w(P)$.*

Proof. A vertex chooses an incident edge in P with probability at least $\frac{1}{2}$. Since the vertices choose independently of each other, an edge in P is chosen with probability at least $\frac{1}{4}$ and the lemma follows. □

Theorem 3.12. ***(Main Theorem,*** $\mathbf{T} \to \mathbf{M_T}$***)*** *In a tree T we have $E[w(M_T)] \geq \frac{1}{16}w(M_T^*)$, that is our algorithm finds a matching which is on average a constant approximation of a maximum weighted matching in T.*

Proof.

$$\begin{aligned} E[w(M_T)] &\geq \frac{1}{4}w(P), && \text{(Third Key Lemma 3.1)} \\ &\geq \frac{1}{4}w(M_F^*), && \text{(Second Key Lemma 3.4)} \\ &\geq \frac{1}{16}w(M_T^*). && \text{(First Key Lemma 3.11)} \end{aligned}$$

□

We conclude this section with the analysis of the time and message complexity.

Theorem 3.13. *Algorithm 1 (*Tree-Matching*) needs a constant number of time steps and a constant number of messages per edge with constant bit size.*

Proof. The theorem follows immediately from the description of the algorithm. □

Corollary 3.14. *For any tree T we have that if the vertices synchronously execute Algorithm 1 (*Tree-Matching*) they find a matching in T with on average constant ratio bound and linear message complexity in constant time.*

4 Matching in General Graphs

4.1 The Algorithm

In this section we present a distributed algorithm which finds a matching with constant ratio bound in polylogarithmic time on general weighted graphs. The algorithm consist of $\log n$ *phases* Φ_i. and each phase consists of $O(\log n)$ *rounds* $\mathcal{R}_j$. In each round, Algorithm 7 (*Uniform-Matching*) is called. This algorithm is divided conceptually into four sub-algortihms: *Select, Eliminate, Matching* and *Cleanup*. In the following we introduce some helpful terminology and then informally describe a phase of the algorithm.

Definition 4.1. *A vertex $u \in V$ calls an edge $e = \{u, v\}$ a* candidate *if $w(e) \geq \frac{1}{2} \cdot w_{\max}(u)_E$. Edge e is a* valid candidate *if it is a candidate for u and v.*

Outline of the algorithm: To find a matching M_G on a weighted graph $G = (V, E)$ each vertex u computes in each phase Φ_i all its incident valid candidates. The set of all valid candidates induce a graph $H^{(1)}$ on $G^{(i)}$. On this graph a maximal matching is computed by the repeated execution of the Uniform-Matching Algorithm (Algorithm 7). We will in short describe how this algorithm finds a maximal matching on $H^{(1)}$: In each round $\mathcal{R}_j$ a sparse subgraph of $H^{(j)}$ is computed by the Select Algorithm (Algorithm 8) and the Eliminate Algorithm (Algorithm 9). In the Select Algorithm, each vertex chooses randomly one incident edge and thus induces a graph $H_S^{(j)}$ on $H^{(j)}$.

Definition 4.2. *A vertex $u \in V$ calls the edge it has chosen in the Select Algorithm (Algorithm 8)* chosen. *The other incident edges in $H_S^{(j)}$ it calls* imposed.

In the Eliminate Algorithm the vertices bound their degree in $H_S^{(j)}$ by randomly choosing one imposed edge and deleting all the others, except the chosen one. The subgraph induced by this step is a collection of cycles and paths on which we find a matching in the Matching Algorithm (Algorithm 10). After having removed all edges of the matching with all their adjacent edges from $H^{(j)}$ in the Cleanup Algorithm (Algorithm 11) the Uniform-Matching Algorithm is repeated.

Algorithm 5 Graph-Matching $(G \rightarrow M_G)$, executed by vertex u

```
Input: Graph $G = (V, E)$
Output: Matching $M_G$
 1: (* See Footnote 4 *)
 2: $G^{(1)} := G$, $M_G := \emptyset$;
 3: for ($i$ from 1 to $\log n$ by 1) do
 4:     (* start of phase $\boldsymbol{\Phi_i}$ *)
 5:     $H^{(1)}$ := Algorithm 6 (Valid), with Input $G^{(i)}$;
 6:     $G^{(i+1)} := G^{(i)} \backslash H^{(1)}$;
 7:     $j := 1$;
 8:     while ($d_{H^{(j)}}(u) > 0$) do
 9:         (* start of round $\boldsymbol{\mathcal{R}_j}$ *)
10:         $(H^{(j+1)}, M_H^{(j)})$ := Algorithm 7 (Uniform-Matching), with Input $H^{(j)}$;
11:         $M_G := M_G \cup M_H^{(j)}$;
12:         $j := j + 1$;
13:     end while;
14: od;
```

Algorithm 6 Valid $(G^{(i)} \rightarrow H)$, executed by vertex u

```
Input: graph $G^{(i)} = (V_G^{(i)}, E_G^{(i)})$
Output: graph $H = (V_H, E_H)$
 1: $V_H := V_G^{(i)}$; $E_H = \emptyset$;
 2: $S := \{v \mid e = \{u, v\} \in G^{(i)}, w(e) \geq \frac{1}{2} \cdot w_{\max}(u)_{E_G^{(i)}}\}$;
 3: for all $v \in S$ do
 4:     send message "you are a candidate" to $v$;
 5: od;
 6: receive message from all neighbors $v$ in $G^{(i)}$;
 7: (* $\boldsymbol{u}$ waits until it received all messages *)
 8: if (received message "you are a candidate" from $v \in S$) then
 9:     $E_H := E_H \cup \{u, v\}$
10: fi;
11: (* $\boldsymbol{H}$ is the subgraph of $\boldsymbol{G^{(i)}}$ which contains all valid candidates. The weight of the incident edges
    of $\boldsymbol{u}$ in $\boldsymbol{H}$ is at least $\boldsymbol{\frac{1}{2} \cdot w_{\max}(u)_{E_G^{(i)}}}$. The degree of $\boldsymbol{u}$ in $\boldsymbol{H}$ may vary between zero and $\boldsymbol{d_{G^{(i)}}(u)}$.
    *)
```

4.2 Analysis

In this subsection we analyze the behavior of the Graph-Matching Algorithm given above. As in Section 3 we first study the quality of the computed matching, then the time and message complexity. For the analysis we assume w.l.o.g. that the vertices enter phase Φ_i synchronously. If they do not, it certainly only improves the running time of our algorithm. If not stated otherwise, $G = (V, E)$ denotes the graph on which the matching is to be computed, with $|V| = n$. $G^{(i)} \in G$ is the graph in phase Φ_i containing all edges not yet adjacent to or in the matching M_G. $H^{(1)}$ is the graph of all valid candidates of $G^{(i)}$. (For the sake of readability we omitted an extra subscript for $H^{(1)}$ to indicate the phase; this is always obvious from the context.) Let $w_{\max}(G)$ be abbreviated $w_{\max}$.

We first present several lemmas which simplify the proof of the constant ratio bound.

Lemma 4.3. ***(First Key Lemma)*** *For each vertex u and each phase Φ_i it holds that after $O(\log n)$ rounds the condition in line 8 of Algorithm 5 is false with high probability. That is, after $O(\log n)$ rounds a maximal matching in $H^{(1)}$ was found with high probability.*

Proof. In the Select Algorithm (Algorithm 8) a vertex u chooses uniformly at random one incident edge.

Algorithm 7 Uniform-Matching $(H^{(j)} \to H^{(j+1)})$, executed by vertex u

```
Input: graph $H^{(j)} = (V_H^{(j)}, E_H^{(j)})$
Output: graph $H^{(j+1)} = (V_H^{(j+1)}, E_H^{(j+1)})$, matching $M_H$
1: $H_S^{(j)}$ := Algorithm 8 (Select), with Input $H^{(j)}$;
2: $P^{(j)}$ := Algorithm 9 (Eliminate), with Input $H_S^{(j)}$;
3: $M_H^{(j)}$ :=Algorithm 10 (Matching), with Input $P^{(j)}$;
4: $M_H := M_H \cup M_H^{(j)}$;
5: $H^{(j+1)}$ := Algorithm 11 (Cleanup), with Input $(H^{(j)}, M_H^{(j)})$;
```

Algorithm 8 Select $(H^{(j)} \to H_S^{(j)})$, executed by vertex u

```
Input: graph $H^{(j)} = (V_H^{(j)}, E_H^{(j)})$
Output: graph $H_S^{(j)} = (V_{H_S}^{(j)}, E_{H_S}^{(j)})$
1: $V_{H_S}^{(j)} := V_H^{(j)}$; $E_{H_S}^{(j)} := \emptyset$;
2: choose uniform at random one edge $e = \{u, v\}$, $e \in E_H^{(j)}$, call $e$ chosen;
3: $E_{H_S}^{(j)} := E_{H_S}^{(j)} \cup \{e\}$;
4: send message “you are chosen” to $v$;
5: receive message from all neighbors $w$ in $H^{(j)}$;
6: if (received message from w) then
7:     $E_{H_S}^{(j)} := E_{H_S}^{(j)} \cup \{u, w\}$; call $\{u, w\}$ imposed;
8: fi;
9: (* If $u$ has positive degree in $H^{(j)}$, it has positive degree in $H_S^{(j)}$. *)
```

If u itself has at least one imposed edge after the Select Algorithm, it has positive degree after the Eliminate Algorithm (Algorithm 9) and is further incident to the matching computed in the Matching Algorithm (Algorithm 10) with probability at least $\frac{1}{2}$.[5] According to the terminology given in [II86] we call a vertex *good* if more than $1/3$ of its neighbors do not have a larger degree. An edge e is *good* if at least one incident vertex is good. Then at least half of the edges are good.[6] Suppose, u is a good vertex and let $v_1, \ldots, v_d$ be u's neighbors. A neighbor v_j has degree d_j. The probability that u has at least one imposed edge after the Select stage can be computed using the following standard trick:

$$\begin{aligned}
\text{Prob}(u \text{ has no imposed edge}) &= \prod_{i=1}^{d} \left(1 - \frac{1}{d_i}\right) \\
&\le \left(1 - \frac{1}{d}\right)^{d/3}, \quad \text{(at least } d/3 \text{ neighbors have smaller or equal degree)} \\
&\le e^{-1/3} \qquad \text{(Fact 2.1)}.
\end{aligned}$$

All together we may conclude that the probability that a good edge is removed in the Cleanup Algorithm (Algorithm 11) is constant. Since at least half of the edges of any graph are good, after logarithmic many rounds *all* edges are removed and a maximal matching is found with high probability.[7] □

Lemma 4.4. *All edges $e \in E$ with weight $w(e) \ge \frac{1}{2} \cdot w_{\max}$ are valid candidates in phase Φ_1, that is e is an edge in $H^{(1)}$ of phase Φ_1.*

Proof. For contradiction assume that $w(e) \ge \frac{1}{2} \cdot w_{\max}$ and $e = \{u, v\}$ is not an edge in $H^{(1)}$ of phase

[5]The math is not very intricate and hence omitted in this extended abstract due to the lack of space.

[6]For a proof see e.g. [KVY94].

[7]A standard probabilistic argument can be applied to derive constant fraction and high probability from constant fraction and constant probability.

Algorithm 9 Eliminate $(H_S^{(j)} \to P^{(j)})$, executed by vertex u

```
Input: graph $H_S^{(j)} = (V_{H_S}^{(j)}, E_{H_S}^{(j)})$
Output: set of cycles and paths $P^{(j)} = (V_P^{(j)}, E_{P^{(j)}})$
1: $V_P^{(j)} := V_{H_S}^{(j)}, E_P^{(j)} := \emptyset$
2: choose uniform at random one imposed edge $e = \{u, v\}$;
3: send message “this edge is in $P^{(j)}$” to $v$;
4: $E_P^{(j)} := E_P^{(j)} \cup \{e\}$;
5: receive message from all neighbors $w$ in $H_S^{(j)}$;
6: if (received message from w) then
7:     $E_P^{(j)} := E_P^{(j)} \cup \{u, w\}$;
8: fi;
9: (* If $u$ has at least one imposed edge, $d_{P^{(j)}}(u) \geq 1$. In general, $d_{P^{(j)}}(u) \leq 2$. *)
```

Algorithm 10 Matching $(P^{(j)} \to M_G^{(j)})$, executed by vertex u

```
Input: set of paths and cycles $P^{(j)} = (V_P^{(j)}, E_P^{(j)})$
Output: matching $M_G^{(j)}$
1: $M_G^{(j)} := \emptyset$;
2: choose uniformly at random one incident edge $e = \{u, v\}$ in $P^{(j)}$;
3: send message “you are my matching partner” to $v$;
4: receive message from all neighbors $w$ in $P^{(j)}$;
5: if (received message from $v$) then
6:     $M_G^{(j)} := M_G^{(j)} \cup \{e\}$;
7: fi;
8: (* An edge is part of the matching $M_G^{(j)}$ if both its endpoints have chosen it, therefore $M_G^{(j)}$ is a
   valid matching. *)
```

Φ_1. Then e was not a candidate for at least one incident vertex. W.l.o.g. let this vertex be u. Then $w_{\max}(u)_E > 2 \cdot w(e) \geq w_{\max}$, which is a contradiction. □

Corollary 4.5. *In phase Φ_1 and after $O(\log n)$ rounds all edges e with weight $w(e) \geq \frac{1}{2} \cdot w_{\max}$ are either adjacent to or in the matching M_G with high probability.*

Proof. The corollary follows immediately from the First Key Lemma 4.3 and Lemma 4.4. □

Definition 4.6. *We say that an edge $e \in E$ is* heavy *if $w(e) \geq \frac{w_{\max}}{n}$, else it is* light.

Lemma 4.7. ***(Second Key Lemma)*** *After $\log n$ phases all heavy edges are either adjacent to or in the matching M_G with high probability.*

Proof. By Lemma 4.4 all edges e with weight $w(e) \geq \frac{1}{2} \cdot w_{\max}$ are valid candidates in phase Φ_i and therefore in $H^{(1)}$. After a maximal matching on $H^{(1)}$ was computed the vertices enter the next phase Φ_2. In $G^{(2)}$ the heaviest edge has weight less than $\frac{1}{2} \cdot w_{\max}$ and following the argument of Corollary 4.5 all edges e with weight $w(e) \geq \frac{1}{4} \cdot w_{\max}$ are adjacent to or in the matching M_G with high probability after another $O(\log n)$ rounds. We repeat this argument $\log n$ times. In the graph $G^{(\log n+1)}$ the heaviest edge has weight less than $\frac{w_{\max}}{2^{\log n}} = \frac{w_{\max}}{n}$ and all heavier edges are either adjacent to or in the matching M_G. □

Observation 4.8. *We can partition the edge-set of a graph G in the following way:*

$$E = \dot{\bigcup_i} E_i, \quad E_i = \{e \mid e \in E, \frac{w_{\max}}{2^{i+1}} < w(e) \leq \frac{w_{\max}}{2^i}\}.$$

Algorithm 11 Cleanup $(H^{(j)} \to H^{(j+1)})$, executed by vertex u

```
Input: Graph $H^{(j)} = (V_H^{(j)}, E_H^{(j)})$, Matching $M_H^{(j)}$
Output: Graph $H^{(j+1)}$
1: remove all edges $e \in M_H^{(j)}$ from $H^{(j+1)}$, $E_{H^{(j+1)}} := E_H^{(j)} \setminus E_{M_H^{(j)}}$;
2: remove all edges adjacent to an edge in $M_H^{(j)}$ from $H^{(j+1)}$,
   $E_{H^{(j+1)}} := E_{H^{(j+1)}} \setminus \{e \mid e \text{ adjacent to } M_H^{(j)}\}$;
```

Observation 4.9. *Let E' be the union of all heavy edges, $E' = \bigcup_{i=0}^{\log n} E_i$. The weight of a matching M_G^* can be decomposed by*

$$w(M_G^*) = w(M_G^* \cap E') + \sum_{i>\log n} w(M_G^* \cap E_i).$$

Lemma 4.10. *The sum of weights of all light edges in M_G^* is less than half of the weight of the heaviest edge:*

$$\sum_{i>\log n} w(M_G^* \cap E_i) < \frac{1}{2} \cdot w_{\max}.$$

Proof. Edges in E_i, $i > \log n$, have weight less than $\frac{w_{\max}}{n}$. Together with Fact 2.3 we have

$$\sum_{i>\log n} w(M_G^* \cap E_i) < |M_G^*| \cdot \frac{w_{\max}}{n} \leq \frac{1}{2} \cdot |V| \cdot \frac{w_{\max}}{n} = \frac{1}{2} \cdot w_{\max}.$$

□

Lemma 4.11. ***(Third Key Lemma)*** *The sum of weights of all heavy edges in M_G^* is at most four times the weight of the matching computed by the Graph-Matching Algorithm, formally:*

$$w(M_G^* \cap E') \leq 4 \cdot w(M_G).$$

Proof. We will define a mapping $f : (M_G^* \cap E') \to M_G$ with the property that if $f : e \mapsto e'$ then $w(e) \leq 2 \cdot w(e')$. Furthermore, at most two elements of $(M_G^* \cap E')$ are mapped to the same element of M_G. Obviously, if f is well-defined, the lemma follows.

Let $e \in E$ be a heavy edge in M_G^*, that is $e \in (M_G^* \cap E')$. If $e \in M_G$ then $f : e \mapsto e$. Else, by the Second Key Lemma 4.7 there must be an edge $e' \in M_G$, $e' \notin M_G^*$. This edge has weight at least $w(e') \geq \frac{1}{2} \cdot w(e)$, otherwise it would not have been a valid candidate and hence not in the matching. We let $f : e \mapsto e'$. Since each edge in M_G is adjacent to at most two edges in M_G^*, at most two elements of $(M_G^* \cap E')$ are mapped to the same element of M_G and f is well-defined. □

Lemma 4.12. *The weight of the matching M_G is at least half of the weight of the heaviest edge in G:*

$$w(M_G) \geq \frac{1}{2} \cdot w_{\max}.$$

Proof. Let $e = \{u, v\} \in E$ be an edge with maximal weight, $w(e) = w_{\max}$. According to Lemma 4.4, edge e is a valid candidate in phase Φ_1. All other valid candidates of phase Φ_1 incident to u and v must have weight at least $\frac{1}{2} \cdot w_{\max}$. By the First Key Lemma 4.3 we have found a maximal matching on the valid candidates of phase Φ_1 after $O(\log n)$ rounds. Therefore, either e or an adjacent edge to e must be in the matching M_G after phase Φ_1. □

Theorem 4.13. ***(Main Theorem)*** *After $O(\log^2 n)$ time we have, $w(M_G) > \frac{1}{5} w(M_G^*)$ with high probability.*

Proof.

$$\begin{aligned}
w(M_G^*) &= w(M_G^* \cap E') + \sum_{i > \log n} w(M_G^* \cap E_i) && \text{(Observation 4.9)} \\
&< w(M_G^* \cap E') + \frac{1}{2} \cdot w_{\max} && \text{(Lemma 4.10)} \\
&\leq 4 \cdot w(M_G) + \frac{1}{2} \cdot w_{\max} && \text{(Third Key Lemma 4.11)} \\
&\leq 5 \cdot w(M_G) && \text{(Lemma 4.12).}
\end{aligned}$$

□

We conclude this section with the analysis of the time and message complexity.

Theorem 4.14. *Each of the Algorithms 6-11 needs a constant number of time steps and a constant number of messages per edge with constant bit size. By the First Key Lemma 4.3 the while-loop of Algorithm 5 (*Graph-Matching*) is executed $O(\log n)$ times and therefore the Graph-Matching Algorithm has time complexity $O(\log^2 n)$ and message complexity $O(n^2 \log^2 n)$.*

Proof. Follows immediately from the description of the algorithm. □

Corollary 4.15. *For any graph G we have that if the vertices synchronously execute Algorithm 5 (*Graph-Matching*) they find with high probability a matching in G with constant ratio bound and polylogarithmic message complexity per edge in polylogarithmic time.*

References

[CHS02] S. Chattopadhyay, L. Higham, and K. Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 290–297. ACM Press, 2002.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[Edm65] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[HKP01] M. Hanckowiak, M. Karonski, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.

[II86] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, 1986.

[JRS01] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pages 33–42, 2001.

[KS00] M. Karaata and K. Saleh. A distributed self-stabilizing algorithm for finding maximal matching. *Computer Systems Science and Engineering*, 3:175–180, 2000.

[KVY94] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *J. Algorithms*, 17(2):280–289, 1994.

[KW03] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Principles of distributed computing*, 2003.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21:193–201, 1992.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[Pel00] D. Peleg. *Distributed Computing, A Locality-Sensitive Approach*. SIAM, 2000.

[UC00] R. Uehara and Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76:13–17, 2000.