Distributed Weighted Matching

Mirjam Wattenhofer, Roger Wattenhofer

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

Abstract. In this paper, we present fast and fully distributed 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. In particular, the approximation ratio is 4. For the general graph algorithm we prove a constant ratio bound of 5 and a polylogarithmic time complexity of $O(\log^2 n)$.

1 Introduction and Related Work

In a weighted graph $G = (V, E)$, a maximum weighted matching is a subset $E' \subseteq E$ of edges such that no two edges in E' share a common vertex, every edge in $E - E'$ shares a common vertex with some edge in E' 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 shown 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 general distributed graph network model and the objective to have algorithms that are as "local" as possible has long been an important area of research. Among the first results on this topic we would like to mention the ingenious $O(\log^* n)$ coloring algorithm of Cole and Vishkin [CV86]. The matching lower bound was proven by Linial [Lin92]. Thanks to the applications for ad-hoc and sensor networks local graph algorithms recently experienced a second wind and are a field of intensive study ([JRS01], [KW03]). The model will be presented in detail in Sect. 2; for a proficient introduction to local distributed computing, we refer to [Pel00].

[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.

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 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 non-weighted graphs, presenting a linear-time algorithm. In [II86] Israeli and Itai give a randomized[2] $O(\log n)$ time algorithm for the maximal matching problem.[3] The methods Israeli and Itai use are similar to those used by Luby [Lub86] for the related maximal independent set problem. In Sect. 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 Sect. 3 finds a 4-approximation in $O(1)$ time. The graph algorithm in Sect. 4 computes a 5-approximation in $O(\log^2 n)$-time. Beforehand—in Sect. 2—we formally introduce the model. Finally, in Sect. 5, we put our work into perspective.

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, where V denotes the set of vertices, $|V| = n$, and E the set of edges. 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. Following the standard notation we say that our algorithm has a *approximation ratio* 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

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

[3] Note, that [KS00] and [CHS02] focus on self-stabilization, whereas [II86] does not.

$w_{\max}(G)$ 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.

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

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

Fact 2. *In a graph $G = (V, E)$, we have*

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

Fact 3. *For any matching M on a graph $G = (V, E)$, it holds that $|M| \leq \frac{1}{2} \cdot |V|$.*

Fact 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 approximation ratio of this algorithm are shown to be constant. For the sake of clarity and simplicity the algorithm is divided into two procedures, which are to be executed one by one. 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 $T = (V, E)$ we foremost reduce the degree of the vertices without losing too much weight. Towards this goal, in the first procedure, each vertex u *requests* its heaviest incident edge e_u by sending a message through it. Thereafter, u *confirms* the heaviest edge e_v, $e_v \neq e_u$, on which it received a request. If u received a request through e_u, it additionally confirms e_u. All unconfirmed edges are deleted and the result is a set of vertex disjoint paths P. On this set of paths a matching is computed in the second procedure. As input for the second procedure we have an additional parameter k, which controls how often a vertex tries to become incident to the matching and consequently how large the matching will be. The given approximation ratios hold for $k = 1$. For details see the Tree-Matching Algorithm in Sect. 3.2.

3.2 Tree-Matching Algorithm

In order to compute a matching on a weighted tree $T = (V, E)$ with parameter k, each vertex u executes the following algorithms.[4]

On input $T = (V, E), k$ each vertex u executes the following algorithm.

```
(∗ See Footnote 4 ∗)
procedure Tree-Matching ($T, k$):$M$
 1: $P, M_T := \emptyset$
 2: $P :=$ Paths ($T$);
 3: $M_T :=$ Matching ($P, k$);
 4: return $M_T$
```

```
procedure Paths ($T$):$P$
 1: $R := \emptyset$
 2: choose heaviest incident edge $e = \{u, v\}$, $e \in E$, i.e. $w(e) = w_{\max}(u)_E$
 3: send message “requested” to $v$
 4: receive message from all neighbors and add $x$ to $R$ if received message from $x$
 5: if ($v \in R$) then
 6:     send message “confirmed” to $v$
 7: fi
 8: $w = \arg\max_{x \in R \setminus v} w(x)$
 9: send message “confirmed” to $w$
10: receive message from all neighbors
11: return all confirmed edges
12: (∗ Each node has degree at most 2 and hence the returned edges are a set of paths
    ∗)
```

```
procedure Matching ($P, k$):$M$
 1: $M_T := \emptyset$, $i := 0$;
 2: while $i < k$ do
 3:     choose uniformly at random one incident edge $e = \{u, v\}$ in $P$
 4:     send message “you are my matching partner” to $v$
 5:     receive message from all neighbors
 6:     if (received message from $v$) then
 7:         $M_T := M_T \cup \{e\}$
 8:         delete $e$ from $P$
 9:         return $M_T$
10:     fi
11:     $i := i + 1$
12: end while;
13: (∗ An edge is part of the matching $M_T$ if both its incident vertices have chosen it.
    Therefore, $M_T$ is a valid matching. ∗)
```

[4] For the sake of readability, in all our algorithms we omit the subscripts, which should indicate, that a vertex only knows its direct neighborhood, not the entire graph/matching.

3.3 Analysis

Let us call an confirmed edge which was confirmed by both its endpoints *doubly confirmed*. Otherwise it is *singly confirmed*. Then for an edge $e = \{u, v\}$ the following three statements are equivalent:

1. e is doubly confirmed
2. e was u as well as v's heaviest incident edge
3. e was confirmed by a node that requested it.

Also note, that if e was requested by u and not confirmed by v, then v must have confirmed some other heavier edge.

In a first step we will prove that the weight of P is at least as large as the weight of an optimal matching M_T^* on T .

Lemma 5. $w(M_T^*) \leq w(P)$.

Proof. In order to prove the lemma we will show that to each edge e in $M_T^* \backslash P$ we can assign one-to-one an edge e' in $P \backslash M_T^*$ with $w(e') \geq w(e)$. Towards this goal, we construct a 1:1 mapping $f : M_T^* \backslash P \to P \backslash M_T^*$, such that $w(f(e)) \geq w(e)$, which immediately implies $w(M_T^*) \leq w(P)$.

To set up the mapping we root T at an arbitrary node and orient its edges away from the root (note, that we need a root just for the analysis of the algorithm, not for the algorithm itself). Thus, every edge has a *tail* (the parent) and a *head* (the child). There can be at most two types of edges in $M_T^* \backslash P$. Either edge $e = \{u, v\}$ was requested solely by its head v (*type 1*), or it was not requested by its head v (*type 2*). (Note, that if e is requested by both its head and its tail, it is in P.) We will first show how f maps type 1 edges and afterwards how type 2 edges are processed.

$e = \{u, v\}$ *is of type 1*: Since u did not confirm e, it must have singly confirmed a heavier edge $e' \in P$. Map e to e'. This mapping has the desired properties, since e' is adjacent to e and hence not in M_T^* and $w(e') \geq w(e)$. Furthermore, the mapping is one-to-one since e' is either a parent edge of e, singly confirmed by its head, or a sibling of e, singly confirmed by its tail. If we would map another type 1 edge e'' to e' then either e'' would be adjacent to e - contradicting its being in M_T^* - or e' would be singly confirmed by both endpoints - an oxymoron.

$e = \{u, v\}$ *is of type 2*: Vertex v is adjacent to an edge in M_T^* (namely, e) that it has not requested. Since v did not request its parent edge e, it must have requested a heavier child edge. Descend the tree, starting with $x := v$, in the following manner. While x is adjacent to an edge in M_T^* that it did not request, and the edge x did request is a child edge, set $x := y$, where y is the head of the edge requested by x. Let the path thus traversed in the tree be $v = v_0, v_1, \ldots, v_k$. Clearly, $k \geq 1$. It is easy to see that the path contains no edges of M_T^*, and that every v_i, except possibly v_k, is adjacent to an edge in M_T^*. Furthermore, the edges in the path are oriented from v_i to v_{i+1} and each was requested by its tail. Additionally, the weights along the path are monotone nondecreasing and at least as heavy as e.

Let e' be the last edge on this path. Then $e' \notin M_T^*$. If v_k requested e', then e' is doubly confirmed and therefore cannot have been mapped to by a type 1 edge. Map e to e'. Otherwise, v_k must have *singly* confirmed an edge e'' (possibly, $e'' = e'$) such that $w(e'') \geq w(e')$. Map e to e''. In this case we must show that e'' is not in M_T^* and has not been mapped to by a type 1 edge. Any edge in

M_T^* incident to v_k must be a child edge of v_k (because v_k's parent edge is on the path) and it must have been requested by its tail, namely, v_k (otherwise v_k would not be last on the path). Thus e'' cannot be such an edge, for then it would be doubly confirmed. Also, such an edge cannot by of type 1, so it cannot be mapped to e''. It follows that if some type 1 edge is mapped to e'', then either e'' is a child edge of v_k and the type 1 edge mapped to it is a child edge of e'', or $e'' = e'$ and the type 1 edge mapped to it is a child edge of v_{k-1}. In both cases e'' is doubly confirmed - a contradiction.

Finally, no two type 2 edges may be mapped to the same edge, because every type 2 edge is mapped to a descendent edge such that the path connecting them contains no edges in M_T^* (and in particular, no type 2 edges).

□

Lemma 6. *If every vertex in T executes the Matching Procedure with input parameter k and P and the output is denoted M_P, then $E[w(M_P)] \geq (1 - (3/4)^k)w(P)$.*

Proof. A vertex chooses an incident edge in P with probability at least $1/2$. Since the vertices choose independently of each other, an edge in P is chosen with probability at least $1/4$. Hence, if we denote by M_P the the chosen edges after k trials we have

$$E[X] \geq w(P) \sum_{i=1}^{k} (3/4)^i \cdot 1/4 = (1 - (3/4)^k)w(P).$$

□

Theorem 7. *In a tree T and for $k = 1$ we have that $E[w(M_T)] \geq \frac{1}{4}w(M_T^*)$. That is, the Tree-Matching Algorithm finds a matching which is on average a four approximation of a maximum weighted matching in T.*

Proof.

$$\begin{aligned} E[w(M_T)] &\geq \frac{1}{4}w(P) && \text{(Lemma 6)} \\ &\geq \frac{1}{4}w(M_T^*) && \text{(Lemma 5)} \end{aligned}$$

□

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

Theorem 8. *The Tree-Matching Algorithm of Sect. 3.2 (*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 9. *For any tree T we have that if the vertices synchronously execute the Tree-Matching Algorithm of Sect. 3.2 they find a matching in T with on average an approximation ratio of 4 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 on a weighed graph G with a constant approximation ratio of 5 in polylogarithmic time. Conceptually, the algorithm consists of several rounds in each of which it tries to thin out the input graph according to the edge-weights. On this thinned out graph a maximal matching is computed. After logarithmic many of such rounds, we can guarantee that the union of the maximal matchings on the thinned out graphs is a constant approximation of an optimal matching on the original graph. In the following we give a more detailed overview of the algorithm before we provide in Sect. 4.2 the algorithm itself.

Outline of the algorithm: The algorithm consist of $\log n$ *phases* Φ_i. Let us denote the input graph of phase Φ_i by $G^{(i)}$, where $G^{(1)} = G$. Then, in each phase, the algorithm first computes a subgraph $H_i^{(1)}$ of $G^{(i)}$, where for each node u the weight of its incident edges in $H_i^{(1)}$ is at least $1/2 \cdot w_{max}(u)_{E_G^{(i)}}$ (*"validation of edges phase"*). Secondly, it computes in $O(\log n)$ *rounds* $\mathcal{R}_j$ a maximal matching on $H_i^{(1)}$ (*"maximal matching phase"*). The two phases are described in the following.

"validation of edges phase": 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. Each vertex u computes in phase Φ_i all its incident valid candidates (Procedure Valid). The set of all valid candidates induce the graph $H_i^{(1)}$ on $G^{(i)}$.

"maximal matching phase": A maximal matching on $H_i^{(1)}$ is computed in several rounds, where in each round $\mathcal{R}_j$ a sparse subgraph of a subgraph $H_i^{(j)}$ of $H_i^{(1)}$ is computed by the Select and Eliminate Procedure. In the Select Procedure, each vertex chooses randomly one incident edge and thus induces a graph $H_{Si}^{(j)}$ on $H_i^{(j)}$. A vertex $u \in V$ calls the edge it has chosen in the Select Procedure *chosen*. The other incident edges in $H_{Si}^{(j)}$ it calls *imposed*. In the Eliminate Procedure the vertices bound their degree in $H_{Si}^{(j)}$ 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 Procedure. After having removed all edges of the matching with all their adjacent edges from $H_i^{(j)}$ in the Cleanup Procedure the Uniform-Matching Procedure is repeated.

4.2 Graph-Matching

On input $G = (V, E)$ each vertex u executes the following algorithm.

```
(* See Footnote 4 *)
Procedure Graph-Matching $(G) : M$
 1: $G^{(1)} := G$, $M_G := \emptyset$;
 2: for ($i$ from 1 to $\log n$ by 1) do
 3:     (* start of phase $\Phi_i$ *)
 4:     $H_i^{(1)}$ := Valid($G^{(i)}$;
 5:     $G^{(i+1)} := G^{(i)} \backslash H_i^{(1)}$;
 6:     $j := 1$;
 7:     while ($d_{H^{(j)}}(u) > 0$) do
 8:         (* start of round $\mathcal{R}_j$ *)
 9:         $(H_i^{(j+1)}, M_H^{(j)})$ := Uniform-Matching($H_i^{(j)}$);
```

```
10:       $M_G := M_G \cup M_H^{(j)}$;
11:       $j := j + 1$;
12:    **end while**;
13:    remove all edges adjacent to $M_G$ from $G^{(i+1)}$
14: **od**;
15: **return** $M_G$
```

Procedure Valid $(G^{(i)}) : H$

```
 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: (* $u$ 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: **return** $H$
12: (* $H$ is the subgraph of $G^{(i)}$ which contains all *valid* candidates. The weight of
    the incident edges of $u$ in $H$ is at least $\frac{1}{2} \cdot w_{\max}(u)_{E_G^{(i)}}$. The degree of $u$ in $H$
    may vary between zero and $d_{G^{(i)}}(u)$. *)
```

The Uniform-Matching subroutine computes a maximal matching on an input graph $H^{(j)}$. Each vertex u executes the following algorithm.

Procedure Uniform-Matching $(H^{(j)}) : (H^{(j+1)}, M_H)$

```
 1: $H_S^{(j)}$ := Procedure Select($H^{(j)}$)
 2: $P^{(j)}$ := Eliminate($H_S^{(j)}$);
 3: $M_H^{(j)}$ := Matching($P^{(j)}$);
 4: $M_H := M_H \cup M_H^{(j)}$;
 5: $H^{(j+1)}$ := Cleanup(($H^{(j)}, M_H^{(j)}$));
 6: **return** $(H^{(j+1)}, M_H)$
```

Procedure Select $(H^{(j)}) : H_S^{(j)}$

```
 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: **return** $H_S^{(j)}$
10: (* If $u$ has positive degree in $H^{(j)}$, it has positive degree in $H_S^{(j)}$. *)
```

Procedure Eliminate $(H_S^{(j)}) : P^{(j)}$

```
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: **return** $P^{(j)}$
10: $(*$ If $u$ has at least one *imposed* edge, $d_{P^{(j)}}(u) \geq 1$. In general, $d_{P^{(j)}}(u) \leq 2$. $*)$
```

Procedure Matching $(P^{(j)}) : M_G^{(j)}$

```
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: **return** $M_G^{(j)}$
9: $(*$ 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. $*)$
```

Procedure Cleanup $(H^{(j)}) : H^{(j+1)}$

```
1: $H^{(j+1)} := H^{(j)}$
2: remove all edges $e \in M_H^{(j)}$ from $H^{(j+1)}$, $E_{H^{(j+1)}} := E_H^{(j)} \backslash E_{M_H^{(j)}}$;
3: remove all edges adjacent to an edge in $M_H^{(j)}$ from $H^{(j+1)}$,
   $E_{H^{(j+1)}} := E_{H^{(j+1)}} \backslash \{e \mid e \text{ adjacent to } M_H^{(j)}\}$;
4: **return** $H^{(j+1)}$
```

4.3 Analysis

In this subsection we analyze the behavior of the Graph-Matching Algorithm given above. As in Sect. 3 we first study the quality of the computed matching, then the time and message complexity. 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. $H_i^{(1)}$ is the graph of all valid candidates of $G^{(i)}$. Let $w_{\max}(G)$ be abbreviated $w_{\max}$.

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

Lemma 10. *For each vertex u and each phase Φ_i it holds that after $O(\log n)$ rounds the condition in line 7 of Procedure Graph-Matching is false with high probability. That is, after $O(\log n)$ rounds all vertices in $H_i^{(1)}$ are incident to or in the matching and hence a maximal matching in $H_i^{(1)}$ was found with high probability.*

Proof. In the Select Procedure a vertex u chooses uniformly at random one incident edge. If u itself has at least one imposed edge after the Select Procedure, it has positive degree after the Eliminate Procedure and is further incident to the matching computed in the Matching Procedure with probability at least $\frac{1}{2}$. (This is apparent, since with probability $1/4$ an edge is in the matching itself and if it is not, with probability at least $1/4$ one of its incident edges is in the matching.) 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 than itself. An edge e is *good* if at least one incident vertex is good. Then at least half of the edges are good.[5] Suppose, u is a good vertex and let $v_1, \dots, v_d$ 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 Procedure can be computed using the following standard trick:

$$\begin{aligned}\Pr(u \text{ has no imposed edge}) &= \prod_{i=1}^{d}\left(1-\frac{1}{d_i}\right)\\ &\le \left(1-\frac{1}{d}\right)^{d/3}\\ &\le e^{-1/3} \quad \text{(Fact 1).}\end{aligned}$$

where the second equation follows since at least $d/3$ neighbors have smaller or equal degree. All together we may conclude that the probability that a good edge is removed in the Cleanup Procedure 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.[6] □

Observation 11. *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 $H_1^{(1)}$.*

Proof. For contradiction assume that $w(e) \ge \frac{1}{2} \cdot w_{\max}$ and $e = \{u, v\}$ is not an edge in $H_i^{(1)}$ of phase Φ_1. Then e was not a candidate for at least one incident vertex. W.l.o.g. let this vertex be u. Then $w_{\max}(u)_E > 2 \cdot w(e) \ge w_{\max}$, which is a contradiction. □

Corollary 12. *In phase Φ_1 and after $O(\log n)$ rounds all edges e with weight $w(e) \ge \frac{1}{2} \cdot w_{\max}$ are either adjacent to or in the matching M_G with high probability.*

Proof. The corollary follows immediately from Lemma 10 and Observation 11. □

Definition 13. *We say that an edge $e \in E$ is* heavy *if $w(e) \ge w_{\max}/n$, else it is* light.

Lemma 14. *After $\log n$ phases all heavy edges are either adjacent to or in the matching M_G with high probability.*

Proof. By Observation 11 all edges e with weight $w(e) \ge \frac{1}{2} \cdot w_{\max}$ are valid candidates in phase Φ_1 and therefore in $H_1^{(1)}$. After a maximal matching on $H_1^{(1)}$ was computed the vertices enter the next phase Φ_2. In $G^{(2)}$ the heaviest edge has

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

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

weight less than $1/2 \cdot w_{\max}$ and following the argument of Corollary 12 all edges e with weight $w(e) \geq 1/4 \cdot w_{\max}$ 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 $w_{\max}/2^{\log n} = w_{\max}/n$ with high probability and all heavier edges are either adjacent to or in the matching M_G. $\square$

Observation 15. *We can partition the edge-set of a graph G in the following way:*

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

Observation 16. *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 17. *The sum of weights of all light edges in M_G^* is less than half of the weight of the heaviest edge:*

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

Proof. Edges in E_i, $i > \log n$, have weight less than $w_{\max}/n$. Together with Fact 3 we have

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

$\square$

Lemma 18. *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 Lemma 14 there must be an edge $e' \in M_G$, $e' \notin M_G^*$. This edge has weight at least $w(e') \geq 1/2 \cdot w(e)$, 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. $\square$

Lemma 19. *The weight of the matching M_G is at least half of the weight of the heaviest edge in G:*

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

Proof. Let $e = \{u, v\} \in E$ be an edge with maximal weight, $w(e) = w_{\max}$. According to Observation 11, 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 $1/2 \cdot w_{\max}$. By Lemma 10 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 20. *After $O(\log^2 n)$ time we have, $w(M_G) > 1/5 \cdot w(M_G^*)$ with high probability.*

Proof.

$$\begin{aligned}
w(M_G^*) &= w(M_G^* \cap E') + \sum_{i > \log n} w(M_G^* \cap E_i) && \text{(Observation 16)}\\
&< w(M_G^* \cap E') + \frac{1}{2} \cdot w_{\max} && \text{(Lemma 17)}\\
&\leq 4 \cdot w(M_G) + \frac{1}{2} \cdot w_{\max} && \text{(Lemma 18)}\\
&\leq 5 \cdot w(M_G) && \text{(Lemma 19)}.
\end{aligned}$$

□

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

Theorem 21. *The Graph-Matching Algorithm of Sect. 4.2 has time complexity $O(\log^2 n)$ and message complexity $O(n^2 \log^2 n)$.*

Proof. It follows immediately from the description of the algorithm that each of the Procedures Valid, Uniform-Matching, Select, Eliminate, Matching and Cleanup needs a constant number of time steps and a constant number of messages per edge with constant bit size. By Lemma 10 the while-loop of the Graph-Matching Algorithm is executed $O(\log n)$ times and therefore the claimed time and message complexity can be derived. □

Corollary 22. *For any graph G we have that if the vertices synchronously execute the Graph-Matching Algorithm of Sect. 4.2 they find with high probability a matching in G with approximation ratio 5 and polylogarithmic message complexity per edge in polylogarithmic time.*

5 Conclusions

In this paper we presented two distributed constant-approximation algorithms for weighted matching, one for trees which runs in constant time, and one for general graphs which runs in time $O(\log^2 n)$, where n denotes the number of nodes in the graph. Recently, Kuhn et al. [KMW04] showed that in general (non-weighted) graphs matching cannot be approximated constantly without spending at least $\Omega(\log \Delta / \log \log \Delta + \sqrt{\log n / \log \log n})$ communication rounds, where Δ denotes the maximum degree in the graph. This result bounds the running time of our algorithm from below. Furthermore, in light of our result for trees, an interesting area of future research is to investigate which classes of graphs allow constant distributed approximations in constant time, and for which classes of graphs constant-time algorithms experience the lower bound of [KMW04].

We believe that a deeper general knowledge of local algorithms leads to a better understanding of a variety of problems in distributed computing and/or networking. Many *en vogue* research areas, such as ad-hoc and sensor networks, or peer-to-peer computing, essentially boil down to local algorithms, since local algorithms produce solutions with a low communication overhead, or work well in a highly dynamic environment. We believe that classic graph problems will be beneficial when building such systems. For matching in particular, we envision applications in the distributed match-making process of massive multiplayer online games. How our algorithms can be turned into practical match-making solutions, where the problems of self-stabilization and fault-tolerance also need to be addressed, is one of the goals of our future research.

References

[CHS02] S. Chattopadhyay, L. Higham, and K. Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 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 (PODC)*, pages 33–42, 2001.

[KMW04] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Lower and upper bounds for distributed packing and covering. Technical report, ETH Zurich, Dept. of Computer Science, 2004.

[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. *Journal of Algorithms*, 17(2):280–289, 1994.

[KW03] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 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.