Efficient Distributed Weighted Matchings on Trees*

Jaap-Henk Hoepman[1], Shay Kutten[2], and Zvi Lotker[3]

[1] Institute for Computing and Information Sciences
Radboud University Nijmegen, Nijmegen, the Netherlands
jhh@cs.ru.nl
[2] Faculty of Industrial Engineering and Management
Technion, Haifa, Israel
kutten@ie.technion.ac.il
[3] CWI, Amsterdam, the Netherlands
Z.Lotker@cwi.nl

Abstract. In this paper, we study distributed algorithms to compute a weighted matching that have constant (or at least sub-logarithmic) running time and that achieve approximation ratio $2 + \epsilon$ or better. In fact we present two such synchronous algorithms, that work on arbitrary weighted trees.
The first algorithm is a randomised distributed algorithm that computes a weighted matching of an arbitrary weighted tree, that approximates the maximum weighted matching by a factor $2 + \epsilon$. The running time is $O(1)$. The second algorithm is deterministic, and approximates the maximum weighted matching by a factor $2 + \epsilon$, but has running time $O(\log^* |V|)$. Our algorithms can also be used to compute maximum unweighted matchings on regular and almost regular graphs within a constant approximation.

1 Introduction

A *matching* $M(G)$ of a graph $G = (V, E)$ is any subgraph of G where no two edges are incident to the same vertex. A matching is *maximal* if no other edge from G can be added to the matching without violating this requirement. Let $w(e)$ be the weight of an edge $e \in E$ of G, where $w(e) > 0$. Define the weight $w(G)$ of a graph G to be the sum of the weights of all its edges. Then a *maximum weighted matching* $M^*(G)$ of G is a matching whose weight is the maximum among all matchings of G. We say that an algorithm achieves *approximation ratio* α if for all graphs G, the matching it returns has weight at least $\frac{1}{\alpha} w(M^*(G))$, i.e., $\frac{1}{\alpha}$ of the weight of the maximum weighted matching of that graph.

For sequential algorithms, the problem is well studied. For unweighted graphs, Micali and Vazirani [MV80] present an $O(\sqrt{|V|}|E|)$ time algorithm that computes a maximum matching. For weighted graphs Gabow [Gab90] gives an

* Id: random-matchings.tex,v 1.18 2006/04/20 08:28:26 jhh Exp

$O(|V||E|+|V|^2 \log |V|)$ time algorithm, computing the maximum weighted matching. Both return an exact solution, and not approximations.

Surprisingly, few distributed algorithms to compute (an approximation of) the maximum weighted matching of the communication graph are known. For unweighted graphs, there are deterministic distributed algorithms computing a *maximal* matching in trees [KS00], and bipartite and general graphs [CHS02]. A randomised algorithm for the general case also exists: Israeli and Itai [II86] compute a maximal matching (i.e., no approximation) in running time $O(\log |V|)$.

For weighted graphs, Uehara *et al.* [UC00] present a constant time distributed algorithm with approximation ratio $O(\Delta)$ (where Δ is the maximum degree of the graph). Recently, Wattenhofer *et al.* [WW04] presented a randomised distributed algorithm to compute a weighted matching $M(G)$ with approximation ratio 5 and running time $O(\log^2 |V|)$ for general graphs, and approximation ratio 4 and $O(1)$ running time for trees.

Hoepman [Hoe04] presents an $O(|E|)$ time[4] deterministic distributed algorithm that computes a weighted matching for general graphs with approximation ratio 2. This algorithm is based on sequential algorithms by Preis [Pre99] and Avis [Avi83], and does not require collecting all information in one node (which increases the message complexity).

In this paper, we study distributed algorithms to compute a weighted matching that have constant (or at least sub-logarithmic) running time and that achieve approximation ratio $2 + \epsilon$ or better. In fact, we present two such algorithms for arbitrary weighted trees, thus improving the previous algorithm of Wattenhofer *et al.* [WW04].

The first algorithm — presented in Sect. 3 — is randomised, and achieves approximation ratio $2 + \epsilon$ in running time $O(1)$. An interesting feature of this algorithm is that the quality of the approximation depends on the number of rounds the algorithm is allowed to run. The second algorithm — presented in Sect. 4 — is deterministic, and achieves approximation ratio $2 + \epsilon$ in running time $O(\log^* |V|)$. We start by introducing our computation model and some notation in Sect. 2, show how our algorithms can also be applied to achieve constant approximations to the maximum (unweighted) matchings for regular and almost regular graphs in Sect. 5, and finish with some pointers to further research in Sect. 6.

2 Model and notation

Consider a distributed system with n nodes, whose communication graph is $G = (V, E)$. In this paper, G is a tree (denoted T) or a regular graph. Nodes can exchange point-to-point messages with their neighbours over the edges E in the graph. Each edge e has a weight $w(e)$, known to both endpoints of that edge. The system is synchronous and operates in rounds of message exchanges. We measure time complexity of our algorithms as the number of rounds needed to perform

[4] Careful analysis shows that the time complexity is actually at most $O(\text{diam } G)$, the diameter of the graph.

the computation. We note that our algorithms also work in the asynchronous setting, after some minor modifications.

We write G for general graphs, T for trees, P for paths and S for segments (that are pieces of a path). The number of edges in G is $|G|$, and $w(G)$ is the weight of G, i.e., the sum of the weights of all edges of G. $M(G)$ is a weighted matching of graph G, and $M^*(G)$ is the maximum weighted matching of graph G.

Let X be a random variable. We write $\mathbf{E}\left[X \mid Q(X)\right]$ for the conditional expectation over X given that $Q(X)$ holds. By definition

$$\mathbf{E}\left[X \mid Q(X)\right] = \sum_{x:Q(x)} x\mathbf{Pr}\left[X = x \mid Q(X)\right] \tag{1}$$

and for disjoint Q_i that together span the whole range of X we have

$$\mathbf{E}\left[X\right] = \sum_{i} \mathbf{E}\left[X \mid Q_i(X)\right] \mathbf{Pr}\left[Q_i(X)\right] \ . \tag{2}$$

Our protocols are described in plain English, and not in any formal protocol notation, because they are quite straightforward.

3 Randomised case: constant running time

The protocol runs in four phases, and is parameterised by a real-valued constant p between 0 and 1 and an integer constant k greater than 1.

First, given input tree T, a set of paths $P(T)$ is generated by letting nodes select their heaviest incident edge as a potential member of a path. In the second phase, each path is cut into short segments by randomly removing edges from the path, each with probability p. Subsequently, each segment is tested to see whether its length is shorter than k. In the fourth phase, for these short segments an optimal matching is computed in time $O(k)$, while for the remaining longer segments a constant-time randomised algorithm computes a 2 approximation of the optimal matching for this segment. Combining all matchings, and compensating for the loss of dropped edges when cutting paths into segments, this gives an $O(k)$ algorithm to compute a matching, for which we prove an approximation ratio of $2 + \epsilon$ for arbitrary $\epsilon > 0$. Because k is a constant, the running time is $O(1)$.

3.1 Computing the paths

We use the same procedure to construct a set of paths $P(T)$ from a given tree T as presented by Wattenhofer *et al.* [WW04]. That is, a node u requests the addition of its heaviest incident edge (u, v) to this set of paths from its neighbour v. Such a requested edge is only added to the set of paths if either u also receives a request from v (for the same edge), or if v sent a grant to u for its request. Nodes request exactly one edge each. Nodes only grant at most one request, being the heaviest

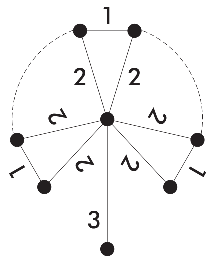

Fig. 1. Counterexample for non-tree graphs.

request coming in over an edge it didn't request itself (assuming unique weights, or breaking ties). All remaining edges (either not granted or not requested) are dropped and will not be members of any path. Because at most 2 edges per node remain, a node is a part of at most one path, and the procedure yields a set of disjoint paths. Because nodes select the heaviest incident edge, these paths cannot be cycles (we assume unique edge weights).

The following lemma shows that we do not loose too much weight constructing paths this way, provided that the graph we start with is a tree.

Lemma 3.1 ([WW04]). *For trees T and $P(T)$ computed as described above,*

$$w(P(T)) \geq w(M^*(T)) \ .$$

In contrast, the following counterexample shows that for non-trees the difference in weight between the paths constructed for that graph and its maximum weight matching can be unbounded. Consider the graph in Fig. 1. One node connects to the central node over an edge with weight 3. This node requests this edge as a path member, and the central node grants that request. Furthermore, $2n$ nodes are connected to the central node through an edge with weight 2, while these nodes are connected pairwise through edges with weight 1. These $2n$ nodes each request the edge with weight 2 as a potential path member, but all these requests are rejected by the central node. Hence, the path consists of just one edge, and has total weight $w(P(G)) = 3$. However, the maximum weighted matching consists of the edge with weight 3, as well as all n edges with weight 1. Hence $w(M^*(G)) = 3 + n$.

3.2 Cutting the paths into segments

In the next phase, each path P is cut into segments $S(P)$ as follows. Every vertex sends, for each of its edges on the path, a cut request over this edge with probability $\sqrt{p}$. An edge is cut if both endpoints sent a cut request to each other. The other edges remain. Each connected component forms a segment S in $S(P)$. We see that the probability for each edge to be cut is exactly p. The expected number of removed edges is $p|P|$, and the expected weight of all edges removed together is $p \cdot w(P)$.

Lemma 3.2. *Let $S(P)$ be a random variable corresponding to the segments computed for a path P by the random process described above. Then*

$$\boldsymbol{E}\left[\sum_{S\in S(P)} w(S)\right] = (1-p)w(P) \ .$$

Proof. Only edges in some $S \in S(P)$ contribute to the weight. Let P consist of edges e_i, and define random variables X_{e_i} such that $X_{e_i} = 1$ iff e_i is a member of some segment (i.e., not cut), and 0 otherwise. Then $\mathbf{Pr}\left[X_{e_i} = 1\right] = (1-p)$ and

$$\begin{aligned}\mathbf{E}\left[\sum_{S\in S(P)} w(S)\right] =& \mathbf{E}\left[\sum_{e_i\in P} w(e_i)X_{e_i}\right] \\ =& \sum_{e_i\in P} \mathbf{E}\left[w(e_i)X_{e_i}\right] \\ =& \sum_{e_i\in P} w(e_i)\mathbf{Pr}\left[X_{e_i}=1\right] = (1-p)\sum_{e_i\in P} w(e_i) \ .\end{aligned}$$

This proves the lemma. □

3.3 Estimating the size of the segment

After cutting the path into segments, each vertex determines the size of the segment it is a member of. Or, to be more precise, it determines whether the segment is smaller or larger than k edges. It does so in the following manner. Vertices at the edge of a segment start the computation, by sending a *distance* message with value 1 along the only incident edge that belongs to a segment (and recording 0 as the distance from the other end). Nodes that receive a distance message record the distance coming in over that edge, and add one before forwarding it over the other segment edge. Forwarding stops if a segment endpoint is reached, or when the distance in the message equals k. If nodes do not record a distance for both edges, the total length of the segment is larger than k. Otherwise, the sum of both distances equals the length of the segment. In either case, all nodes know whether the length of the segment is $\leq k$ or $> k$. The running time is at most k.

3.4 Computing matchings on the segments

Segments compute their matching depending on their sizes, as determined in the previous phase. If a node finds it is not a member of a segment (or rather, it is a member of a segment of size 0) it does nothing. Otherwise, it cooperates with all other nodes in the same segment as described below.

Consider a segment $S \in S(P)$. If $|S| \leq k$, we compute a good matching $M(S)$ for S in time k by computing two matchings M and M' by adding edges in S

alternately to M or M', and selecting the matching with maximum weight. Such a computation could be initiated by both endpoints of the segment. We note the running time is $O(k)$.

In the following lemma we bound the weight of the resulting matching from below[5].

Lemma 3.3. *For a matching $M(S)$ of S (with $|S| \leq k$) computed as described above,*

$$\boldsymbol{E}[w(M(S))] \geq \frac{1}{2}w(S) \ .$$

Proof. By construction we find two matchings M and M' with $S = M + M'$, and therefore $w(S) = w(M) + w(M')$. As we select the heaviest matching, the lemma follows. □

If $|S| > k$, we compute a matching $M(S)$ for S in one round by letting vertices vote for the incident edge that should be added to the matching with equal probability (segment endpoints vote for their only edge with $p = 1$). An edge is only added if both its endpoints vote for it.

Lemma 3.4. *For a matching $M(S)$ of S (with $|S| > k$) computed as described above,*

$$\boldsymbol{E}[w(M(S))] \geq \frac{1}{4}w(S) \ .$$

Proof. As each edge has at most 2 incident edges on the segment, the probability that an edge is added is at least $\frac{1}{4}$. Define random variables X_{e_i} such that $X_{e_i} = 1$ if the i-th edge e_i of S is in the matching. Then, similar to the proof of lemma 3.2,

$$\mathbf{E}\left[w(M(S))\right] = \sum_{e_i \in S} w(e_i)\mathbf{Pr}\left[X_{e_i} = 1\right] \ .$$

As $\mathbf{Pr}\left[X_{e_i} = 1\right] \geq \frac{1}{4}$, the lemma follows. □

3.5 Merging the results

The final matching $M(T)$ for tree T is obtained by merging all matchings computed for all segments in $S(P)$ for each path P in $P(T)$. We conclude our analysis by estimating the weight of the resulting matching.

First we look at a single, but arbitrary, path P. In what follows, let $S(P)$ be a random variable corresponding to the segments computed for P by the random process described in section 3.2. Let S be a random variable ranging over all members of $S(P)$.

For such a path P, define $C(P)$ to be the cycle obtained by merging the two endpoints of P into a single node. Given a segmentation of P, define s_b and s_e to be the first and last segment of P, respectively, where s_b starts at the endpoint

[5] Even though the process is deterministic, we state the bound in terms of an expectation, because that is more useful further on.

and s_e ends at the endpoint (and where either segment may be empty if the first and/or last edge of P happened to be cut). A segmentation of P induces a segmentation on the cycle $C(P)$, by taking all segments, and merging s_b and s_e into a single segment $\bar{s}_c$. Let $S(C(P))$ be a random variable corresponding to the segments computed for $C(P)$. Let $\bar{S}$ be a random variable ranging over all members of $S(C(P))$.

For segments computed on this cycle, we have the following proposition.

Proposition 3.5. *For all $i \leq |P|$,*

$$\boldsymbol{Pr}\left[|\bar{S}| = i\right] \leq (1-p)^i \ .$$

Proof. Clearly, to have a segment of length i, we need i uncut edges. This happens with probability $(1-p)^i$. If $i = |P|$, then this is the exact probability (the segment happens to be the whole cycle), otherwise we need at least 1 ($i = |P|-1$) or 2 cut edges, that each lower the probability with a factor p. □

We also need the following uniformity property on the distribution of the weights over the segments computed for the cycle.

Proposition 3.6. *For any $k \geq 0$,*

$$\boldsymbol{E}\left[w(\bar{S}) \mid |\bar{S}| = k\right] = k\frac{\boldsymbol{E}\left[w(\bar{S})\right]}{\boldsymbol{E}\left[|\bar{S}|\right]}$$

Proof. Let σ be a random variable, ranging over the single edges in a segment $\bar{S}$. Let $\sigma_1, \sigma_2, \ldots$ be the edges in $\bar{S}$. Then

$$\begin{aligned}
\mathbf{E}\left[w(\bar{S}) \mid |\bar{S}| = k\right] =& \{\text{Using the fact that } \bar{S} \text{ consists of } k \text{ edges } \sigma_i.\} \\
& \mathbf{E}\left[\sum_{i=1}^{k} w(\sigma_i) \mid |\bar{S}| = k\right] \\
=& \{\text{Independent of length of } \bar{S} \text{ now.}\} \\
& \mathbf{E}\left[\sum_{i=1}^{k} w(\sigma_i)\right] \\
=& \{\text{By symmetry of } C(p) \\
& \text{all edges appear the same number of times. }\} \\
& k\mathbf{E}\left[w(\sigma)\right]
\end{aligned}$$

By Eq. 2 we have

$$\begin{aligned}
\mathbf{E}\left[w(\bar{S})\right] =& \sum_i \mathbf{E}\left[w(\bar{S}) \mid |\bar{S}| = i\right] \mathbf{Pr}\left[|\bar{S}| = i\right] \\
=& \{\text{By the above.}\} \\
& \sum_i i\mathbf{E}\left[w(\sigma)\right] \mathbf{Pr}\left[|\bar{S}| = i\right] \\
=& \mathbf{E}\left[w(\sigma)\right] \mathbf{E}\left[|\bar{S}|\right]
\end{aligned}$$

Combining both equations proves the proposition. □

Next, we bound the expected weight of a matching of an arbitrary segment $\bar{S}$ from $S(C(P))$ in terms of the expected weight of $\bar{S}$ itself.

Lemma 3.7. *For a matching $M(\bar{S})$ of $\bar{S}$ computed as described above,*

$$\boldsymbol{E}\left[w(M(\bar{S}))\right] \geq \frac{1}{2}\left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right)\boldsymbol{E}\left[w(\bar{S})\right] \; .$$

Proof. First observe that $\mathbf{E}\left[w(M(\bar{S}))\right]$ depends on two random processes: the selection of a segment $\bar{s}$ from $S(C(P))$, and the random variable C denoting the coin sequence thrown by the randomised algorithm that computes $M(\bar{s})$. Let $M(c,\bar{s})$ denote the (deterministic) result of $M(\bar{s})$ when the coins thrown are fixed to sequence c. Then

$$\begin{aligned}
\mathbf{E}\left[w(M(\bar{S}))\right] =& \{\bar{S} \text{ and } C \text{ are independent}\} \\
& \sum_{\bar{s}}(\sum_{c} M(c,\bar{s})\mathbf{Pr}\left[C=c\right])\mathbf{Pr}\left[\bar{S}=\bar{s}\right] \\
\geq& \{\text{Split according to } |\bar{s}| \text{ and using Lemma 3.3 and 3.4}\} \\
=& \sum_{\bar{s}:|\bar{s}|\leq k} \frac{1}{2} w(\bar{s})\mathbf{Pr}\left[\bar{S}=\bar{s}\right] + \sum_{\bar{s}:|\bar{s}|>k} \frac{1}{4} w(\bar{s})\mathbf{Pr}\left[\bar{S}=\bar{s}\right] \\
=& \{\text{Rearranging sums and definition of } \mathbf{E}\left[w(\bar{S})\right].\ \} \\
& \frac{1}{2}\mathbf{E}\left[w(\bar{S})\right] - \sum_{\bar{s}:|\bar{s}|>k} \frac{1}{4} w(\bar{s})\mathbf{Pr}\left[\bar{S}=\bar{s}\right] \\
=& \{\text{Using Eq. 1 and } \mathbf{Pr}\left[\bar{S}=\bar{s} \mid |\bar{S}|>k\right]\mathbf{Pr}\left[|\bar{S}|>k\right] = \mathbf{Pr}\left[\bar{S}=\bar{s}\right]\} \\
& \frac{1}{2}\mathbf{E}\left[w(\bar{S})\right] - \frac{1}{4}\mathbf{E}\left[w(\bar{S}) \mid |\bar{S}|>k\right]\mathbf{Pr}\left[|\bar{S}|>k\right] \\
=& \{\text{Using Prop. 3.6}\} \\
& \frac{1}{2}\mathbf{E}\left[w(\bar{S})\right] - \frac{1}{4}\mathbf{E}\left[w(\bar{S})\right]\sum_{i>k} i\frac{\mathbf{Pr}\left[|\bar{S}|=i\right]}{\mathbf{E}\left[|\bar{S}|\right]} \\
\geq& \{\text{Using Prop. 3.5 and } \mathbf{E}\left[|\bar{S}|\right] \geq 1\} \\
& \left(\frac{1}{2} - \frac{1}{4}\sum_{i>k} i(1-p)^i\right)\mathbf{E}\left[w(\bar{S})\right] \\
=& \{\text{Rearranging sums and computing geometric series.}\} \\
=& \left(\frac{1}{2} - \frac{1}{4}\left(\frac{1-p}{p^2} - \frac{k(1-p)^{k+2}-(k+1)(1-p)^{k+1}+(1-p)}{(-p)^2}\right)\right) \\
& \mathbf{E}\left[w(\bar{S})\right] \\
=& \left(\frac{1}{2} - \frac{1}{4}(1+kp)\frac{(1-p)^{k+1}}{p^2}\right)\mathbf{E}\left[w(\bar{S})\right]
\end{aligned}$$

This proves the lemma. □

We also need the following two propositions.

Proposition 3.8.

$$\boldsymbol{E}\left[w(\bar{S}\right] \geq \boldsymbol{E}\left[w(S)\right] \ .$$

Proof. Any random segmentation for $S(P)$ induces a segmentation of $S(C(P))$, with s_e and s_b merged into $\bar{s}_c$ where $w(\bar{s}_c = w(s_b) + w(s_e)$. □

Proposition 3.9.

$$\boldsymbol{E}\left[\sum_{S \in S(P)} w(M(S))\right] \geq \boldsymbol{E}\left[\sum_{\bar{S} \in S(C(P))} w(M(\bar{S}))\right]$$

Proof. In what follows, let $S(P)$ be a random variable corresponding to the segments computed for a path P by the random process described in section 3.2. Let $S(C(P))$ be the corresponding set of segments for the cycle $C(P)$.

For all $S \in S(P)$ unequal to the end segments s_b and s_e, the corresponding segment $\bar{S}$ in $S(C(P))$ is the same, and hence $w(M(S)) = w(M(\bar{S}))$. It remains to show that $\mathbf{E}\left[w(M(s_b)) + w(M(s_e))\right] \geq \mathbf{E}\left[w(M(\bar{s}_c))\right]$. Split the matching $M(\bar{s}_c)$ into two parts, m_b (for $\bar{s}_b$) and m_e (for $\bar{s}_e$), that cover s_b and s_e respectively. Let $i \in \{e, b\}$. We show $\mathbf{E}\left[w(M(s_i))\right] \geq \mathbf{E}\left[w(m_i)\right]$. There are two cases.

$|s_i| \leq k$: In this case (see proof Lemma 3.3), the matching computed for s_i is optimal. As matching m_i on $\bar{s}_i$ is also a matching for s_i, the statement follows.

$|s_i| > k$: Then $|\bar{s}| > k$ as well, and the matching $M(\bar{s}_c)$ is computed using the probabilistic method for long segments (cf. Lemma 3.4). Consider the random process that selects edges for inclusion in $M(\bar{s}_c)$ and hence m_i. The 'end-edge' of $\bar{s}_i$ (the splitting edge at which the cycle is cut into the path P) has probability $1/2$ to be included in $M(s_i)$ but only probability $1/4$ to be included in $m_i = M(\bar{s}_i)$. Hence the expected weight of the matching m_i is lower than $M(s_i)$.

This completes the proof. □

We now bound the weight of the matching computed for P as a whole.

Theorem 3.10. *For any path P, and matching $M(P)$ computed as above,*

$$\boldsymbol{E}\left[w(M(P))\right] \geq \frac{1}{2}\left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right)(1-p)w(P) \ .$$

Proof. In what follows, let $S(P)$ be a random variable corresponding to the segments computed for a path P by the random process described in section 3.2.

We have

$$\begin{aligned}\mathbf{E}\left[w(M(P))\right] =& \mathbf{E}\left[w\left(\bigcup_{S\in S(P)} M(S)\right)\right] = \mathbf{E}\left[\sum_{S\in S(P)} w(M(S))\right] \\ \geq& \{\text{By Prop. 3.9}\} \\ & \mathbf{E}\left[\sum_{\bar{S}\in S(C(P))} w(M(\bar{S}))\right] = \sum_{\bar{S}\in S(C(P))} \mathbf{E}\left[w(M(\bar{S}))\right]\end{aligned}$$

By Lemma 3.7 this is greater than or equal to

$$\frac{1}{2}\left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right) \sum_{\bar{S}\in S(C(P))} \mathbf{E}\left[w(\bar{S})\right]$$

which is bounded from below through Prop 3.8

$$\frac{1}{2}\left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right) \sum_{S\in S(P)} \mathbf{E}\left[w(S)\right]$$

which is further bounded from below using Lemma 3.2 by

$$\frac{1}{2}\left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right)(1-p)w(P)\ .$$

This completes the proof. □

We finish by combining all matchings for all paths.

Corollary 3.11. *For any tree T, and matching $M(T)$ computed as above, we have*

$$2\boldsymbol{E}\left[w(M(T))\right] \geq \left(1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}\right)(1-p)M^*(T)\ .$$

Hence, the approximation ratio of the algorithm is $2+\epsilon$ for arbitrary $\epsilon > 0$.

Proof. Follows from Theorem 3.10 and Lemma 3.1 and the fact that $M(T) = \cup_{P\in P(T)} M(P)$.

To see that the approximation ratio is equivalent to $2+\epsilon$, set $\epsilon = \frac{p}{1-p}$ and observe that $1 - \frac{(1+kp)(1-p)^{k+1}}{2p^2}$ tends to 1 for k going to infinity. □

4 Deterministic case: $O(\log^* n)$ running time

Now consider a model where every node v has a unique identity $ID(v)$. Again, fix a constant k. In [KP98] a deterministic distributed algorithm is presented to partition a tree (or a forest) into clusters of diameter $O(k)$, each containing at least $k+1$ nodes. This constructions is done in $O(k \log^* n)$ time. More precisely:

Lemma 4.1. *[KP98] The collection P_{out} output by Algorithm DOM-Partition(k) is a partition (of the input tree T). Furthermore, if T is of size $n \geq k+1$, then every cluster C in P_{out} has the following properties:*

- *(a)* $|C| \geq k+1$.
- *(b)* $Radius(C) \leq 5k+2$.

Moreover, Algorithm DOM-Partition(k) requires time $O(k \log^* n)$.

This algorithm uses, as a subroutine, the algorithm of [GPS87].

Using this algorithm, it is easy to modify the randomised algorithm of the previous sections to become a deterministic algorithm (with a slightly higher time complexity of $O(\log^* n)$) as follows. Replace the second phase of the probabilistic algorithm (the randomised cutting of the paths into segments) by Algorithm DOM-Partition(k).

This change would have been enough, had the weights of the edges been equal. We need to take some care here, because DOM-Partition(k) may have deleted heavy edges. The nodes of each segment s_i cooperate to perform the following operation. Consider e^i_l, e^i_r, the edges whose deletion separated segment s_i from the rest of the path. Let also e^i_m be the minimum weight edge in this segment. If the weight of either e^i_l or e^i_r is larger than that of e^i_m then e^i_m is removed from the segment, and the largest of e^i_l or e^i_r is reinserted as a part of the segment. If two adjacent segments wish to swap the same separator (e^i_l equals e^{i+1}_r) for a lighter one, then the segment with the smallest minimum weight edge e^i_m wins and performs the swap. Clearly the above can be performed distributively in constant time (since k is a constant).

The above correction may create segments that contain less than k nodes, or segments that are at most three times as long as the original segments. (A segment that itself is not split may join both the segment at its left and its right, however then these two segments are cut in exchange for the separating edges.)

The rest of the algorithm needs no changes, except that we will have no segments that are longer than $15k+6$ for some constant k, so the special treatment of long segments is not necessary. The time complexity of the resulting procedure is $O(\log^* n)$ since in the current paper we assume that k is a constant.

Theorem 4.2. *For any tree T, and matching $M(T)$ computed as above, we have*

$$2w(M(T)) \geq \left(1 - \frac{1}{k+1}\right) M^*(T) \ .$$

Hence, the approximation ratio of the algorithm is $2/(1-\frac{1}{k+1}) = 2\frac{k+1}{k}$, which equals $2+\epsilon$ if we set $\epsilon = \frac{2}{k}$.

Proof. First we compute $\sum_{S \in S(P)} w(S)$ for an arbitrary path P. As Algorithm DOM-Partition(k) returns segments of length at least $k+1$ (if the path has length at least $k+1$, otherwise no edges are cut), the number of cut edges from P is at most $|P|/(k+1)$. Swapping the originally cut edges by lower weight ones does not change the number of cut edges. Since in every original segment we

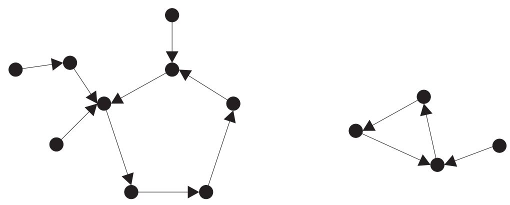

Fig. 2. Examples of simple graphs.

swapped the minimum weight edge for a removed edge, the expected weight of the cut edges (after the swap) is at most $w(P)/(k+1)$. Hence

$$\sum_{S \in S(P)} w(S) = (1 - \frac{1}{k+1}) w(P) \ .$$

Next we consider the weight of the matching returned for P. Similar to the proof of Th. 3.10, and using only Lemma 3.3 to bound $w(M(S))$ by $\frac{1}{2}w(S)$ (the proof of that lemma not only works in the expected case but also in the deterministic case, because all segments have length less than $15k+6$) we have

$$w(M(P)) \geq \sum_{S \in S(P)} \frac{1}{2} w(S) \ .$$

This equals

$$\frac{1}{2} \sum_{S \in S(P)} w(S) \ ,$$

and using the above result we see that

$$w(M(P)) \geq \frac{1}{2}(1 - \frac{1}{k+1}) w(P) \ .$$

Combining the matching for all paths constructed for the tree, and by Lemma 3.1, the theorem follows. □

5 Regular and almost regular graphs

We now show how the algorithms from the previous sections can be used to compute a constant approximation for the maximum matching of unweighted, arbitrary regular graph and almost regular graphs.

In order to describe the algorithm we need the following notations. We denote by "directed simple" a directed graph were each node has only one outgoing edge, in other words the out degree of the node is 1. Note that the number of directed edges in a directed-simple graph is n and therefore there is at least one directed cycle. See Fig. 2.

The algorithm works in three phases. In the first phase we generate a directed-simple spanning subgraph, and in the second phase we transform the directed-simple spanning subgraph into a collection of disjoint paths. Finally, in the third phase, we can use one of the algorithms of the previous sections to compute the maximum matching for the graph (by assigning weight 1 to all edges). We make sure that the number of remaining edges after the second phase is a constant fraction of the number of nodes n. Because the maximum matching of an unweighted graph is never larger than $\frac{n}{2}$, the resulting maximum matching is a constant approximation.

A natural way to construct a directed-simple spanning subgraph is for each node to select a random neighbour from the d neighbours uniformly. After this step, we have n directed edges. Denote the union of all these directed edges by $G'(V, E')$. Let V'_i be the set of all the nodes in G' that have a degree i (in-degree plus out-degree) and $\overline{V'}_i$ be the set of all the nodes in G' of degree bigger than or equal to i. The next lemma estimates the size of $V'_1, V'_2, \overline{V'}_3$.

Lemma 5.1. *Let G' be the directed graph constructed as above. Then*

$$E[|V'_1|] = (1 - \frac{1}{d})^d n \leq \frac{1}{e} n$$

$$E[|V'_2|] = ((1 - \frac{1}{d})^{d-1}) n \geq \frac{1}{e} n$$

$$E[|\overline{V'}_3|] = n - E[|V'_2|] - E[|V'_1|]$$

Proof. First we compute the probability for a node to have a degree 2. $P[d'(v) = 2] = \frac{d}{d}(1 - \frac{1}{d})^{d-1} = (1 - \frac{1}{d})^{d-1} \geq \frac{1}{e}$. The probability of a node to be of degree 1 is $P[d'(v) = 1] = (1 - \frac{1}{d})^d \leq \frac{1}{e}$. Now we use the linearity of the expectation and the lemma follows.

The next corollary shows that for a regular graph the expected size of V'_2 is linear.

Corollary 5.2. *For all $d \geq 2$*

$$n/e \geq E[|V'_1|] \geq n/4$$

$$n/2 \geq E[|V'_2|] \geq n/e$$

$$0.416n \geq E[|\overline{V'}_3|] \geq (1 - 1/e - 1/2)n = 0.132n$$

When we remove the direction from the edges we get that each connected component in G' is a union of paths, and it ends in a cycle.

Let v be a node in G' s.t $d(v') > 2$. From the definition of G' it follows that there is only one edge that is outgoing from the node (this is the edge that was chosen by the node) and this node has more than one ingoing edge. In the next time step we transform the graph G' into the graph G'' by randomly removing all the extra incoming edges except one. After this step all the nodes in $\overline{V'}_3$ have out-degree at most 1 and in-degree equal to 1. Note that the out-degree can actually become 0 if the outgoing edge happens to be incoming to another node v'' with degree $d(v'') > 2$ who removes this edge to reduce its in-degree to 1. Clearly after this step $E[|V_1''|] + E[|V_2''|] \geq E[|V_3'|] > 0.132n$.

Because all these nodes have degree 1 or 2, we have at least as many edges in the graph as well. Now, we assign weights 1 to all remaining edges, and run one of the algorithms from the previous section to compute a maximum weighted matching for the remaining graph. This yields a matching of size at least $(E[|V_1''|] + E[|V_2''|])/3 \geq \frac{0.132}{3}n$. The reason we divide by 3 is that we may have a cycle of length 3.

Since a maximum matching of the original unweighted regular graph is never larger than $\frac{n}{2}$, it follows that we have a randomised algorithm that proximate the maximum matching with a in expected $\frac{1}{2}\frac{3}{0.132} = 11.36$ approximation ratio for a regular graph in a constant time.

Note that the same proof will work if the graph is not an r-regular graph but an almost r-regular graph. We say that a graph G is an α-d-regular graph if $\frac{\Delta}{\delta} < \alpha$, where Δ is the maximal degree and δ is the minimal degree. The next lemma replaces lemma 5.1 for α-d regular graphs.

Lemma 5.3. *Let G be an α-d-regular graph then*

$$E[|V_1'|] \leq ne^{-1/\alpha}$$

$$E[|V_2'|] \leq ne^{-1/\alpha}\frac{\Delta}{\Delta - 1}$$

$$E[|\overline{V'}_3|] = n - E[|V_2'|] - E[|V_1'|]$$

If $\Delta = 2$ then our graph is a forest with some cycles. Using the results from section 3.5 it follows that for a graph without cycles we have a $2+\epsilon$ approximation. Sine a cycle is very close to a path we can use the same idea for cycles, and get a $2+\epsilon$ approximation for graphs with $\Delta \leq 2$. So we may assume that $\Delta \geq 3$. The next corollary shows that for $1/\log(5/2) = 1.09136 > \alpha$-$d$-regular graph the expected size of V_3' is linear.

Corollary 5.4. *for all $\delta \geq 2$ and $1.09136 > \alpha$,*

$$E[|V_3'|] \geq n\left(1 - e^{-1/\alpha}\frac{2\Delta - 1}{\Delta - 1}\right) = \Omega(n) \ .$$

Since the size of V_3' is linear in n we can apply the algorithm from the previous section and get a constant approximation which depends on α.

Corollary 5.5. *Let $G(n,p)$ when $p > \frac{\log(n)}{n}$ be a random graph. Then our approximation algorithm for matching is a constant approximation.*

From the previous corollary it may seem our algorithm always computes a matching with a constant approximation to the maximum matching. The next example shows that this is not the case. Let K_n be a clique of size n. Let C_{n^2} be a cycle that contains n^2 nodes. We connect all the node in C_{n^2} to all the nodes in K_n. Note that the number of node in this graph is $|V| = n^2 + n$, and the number of edges is $n^2 + n^3$. The order of the maximum matching is $O(n^2)$. However, the expected size of the matching that our algorithm computes is $2n$. So in this case the approximation ratio of the algorithm is $O(\sqrt{|V|})$.

6 Conclusions

We have presented efficient distributed algorithms that compute good approximations for the maximum weighted matchings for arbitrary weighted trees. Equally good algorithms for general graphs that compute constant approximations in sub-logarithmic time are not known. We have shown why our approach of constructing paths fails in the general case. Different techniques therefore seem required to handle arbitrary graphs efficiently.

References

[Avi83] Avis, D. A survey of heuristics for the weighted matching problem. *Networks* **13** (1983), 475–493.

[CHS02] Chattopadhyay, S., Higham, L., and Seyffarth, K. Dynamic and self-stabilizing distributed matching. In *21st PODC* (Monterey, CA, USA, 2002), ACM Press, pp. 290–297.

[Gab90] Gabow, H. Data structures for weighted matching and nearest common ancestors with linking. In *1th SODA* (San Fransisco, Ca., USA, 1990), ACM Press, pp. 434–443.

[GPS87] Goldberg, A. V., Plotkin, S., and Shannon, G. Parallel symmetry breaking in sparse graphs. In *19th STOC* (New York City, NY, USA, 1987), ACM Press.

[Hoe04] Hoepman, J.-H. Simple distributed weighted matchings, 2004. eprint cs.DC/0410047.

[II86] Israeli, A., and Itai, A. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Proc. Letters* **22** (1986), 77–80.

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

[KP98] Kutten, S., and Peleg, D. Fast distributed construction of k-dominating sets and applications. *Journal of Algorithms* **28**, 1 (1998), 40–66.

[MV80] Micali, S., and Vazirani, V. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *21st FOCS* (Syracuse, NY, USA, 1980), IEEE Comp. Soc. Press, pp. 17–27.

[Pre99] Preis, R. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *16th STACS* (Trier, Germany, 1999), C. Meinel and S. Tison (Eds.), LNCS 1563, Springer, pp. 259–269.

[UC00] Uehara, R., and Chen, Z. Parallel approximation algorithms for maximum weighted matching in general graphs. *Inf. Proc. Letters* **76** (2000), 13–17.

[WW04] Wattenhofer, M., and Wattenhofer, R. Distributed weighted matching. In *18th DISC* (Amsterdam, the Netherlands, 2004), R. Guerraoui (Ed.), LNCS 3274, Springer, pp. 335–348.