Fast Parallel Approximation Algorithms for Maximum Weighted Matching Problem

Ryuhei Uehara
uehara@komazawa-u.ac.jp

Natural Science Faculty, Komazawa University

Abstract

We present two parallel approximation algorithms for finding a matching of maximum weight in a given edge-weighted graph. To deal with unbounded weights efficiently on a PRAM, the algorithms only use the total order of the weights. That is, we assume that the algorithms can only compare with weights of edges. The first algorithm deterministically runs in $O(\log n)$ time. Its approximation ratio is $\frac{2}{3\Delta+2}$, where Δ is the maximum degree of the graph. The second algorithm is randomized $\frac{1}{2\Delta+4}$-approximation algorithm. It runs in $O(\log \Delta)$ time, not depending on the size of the graph. Hence the algorithm can be performed on a large scale distributed system.

Key words: approximation algorithm, maximum weighted matching, parallel algorithm, randomized algorithm.

1 Introduction

A matching in a graph is the set of independent edges. The problem to find a matching is a fundamental topic in graph theory. In this paper, we deal with the graphs having weighted edges. For given graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}^+$, a *maximum weighted matching* is the subset M of E such that M is a matching, and $\sum_{e \in M} w(e)$ is greater than or equal to $\sum_{e \in M'} w(e)$ for any other matching M'. The MWM problem is to find a maximum weighted matching for given graph. For the MWM problem, Edmonds' algorithm [Edm65] has stood as one of the paradigms in the search for polynomial time algorithms for integer programming problems (see also [Gal86]). Some sophisticated implementations of his algorithm have improved its time complexity: for example, Gabow [Gab90] has produced an algorithm using $O(n(m + n \log n))$ time and $O(m)$ space, where $n = |V|$ and $m = |E|$.

From the theoretical point of view, the complexity of the MWM problem is still open; Galil asks if the MWM problem is P-complete [Gal86], and only CC-hardness has been pointed in [GHR95] (see [MS92] for the detail of the class CC). It is worth remarking that Karp, Upfal, and Wigderson have shown that the MWM problem is in the class RNC if each weight is bounded by some polynomial of n [KUW86].

On the other hand, from the practical point of view, Edmonds' algorithm is too expensive. Faster algorithms are required, and heuristic algorithms and approximation algorithms have been widely investigated. For example, a survey of heuristic algorithms can be found in [Avi83], and some approximation algorithms can be found in [Ven87]. Recently, for general graphs, Preis [Pre99] has proposed a linear time $\frac{1}{2}$-approximation algorithm for the MWM problem. The algorithm is based on a greedy algorithm, that picks up the locally heaviest edges step by step, analyzed by Avis [Avi83]. On the other hand, for the maximum cardinality matching problem (or the maximum weighted matching problem on uniformly weighted graphs), several NC approximation algorithms have been known (see [KR98] for comprehensive reference). The maximum cardinality matching can be characterized by the notions of alternating paths and augmenting paths (see, e.g., [PS82] for the details). The NC approximation algorithms are based on the characterization. However, when the edges are weighted, we can not use the characterization. (For example, for a path $(e_1, e_2, e_3, e_4, e_5, e_6)$, when $w(e_1) = w(e_3) = w(e_4) = w(e_6) = 1$ and $w(e_2) = w(e_5) = 3$, $\{e_2, e_5\}$ is heavier than the alternating paths $\{e_1, e_3, e_5\}$ and $\{e_2, e_4, e_6\}$.) Hence the NC approximation

algorithms for the maximum cardinality matching problem cannot be extend to the maximum weighted matching problem.

In this paper, we propose two parallel approximation algorithms for the MWM problem. We first clarify how to deal with the unbounded weights of edges. Standard parallel computation models as PRAM cannot deal with unbounded weights efficiently when the weights are exponentially large, or real numbers taking many bits to represent. Our algorithms require only comparison operation between two weights of edges. That is, the algorithms are sufficient to know the total order of the weights of edges, and they do not need to store each weight itself. For given graph G with maximum degree Δ, the first algorithm is an NC $\frac{2}{3\Delta+2}$-approximation algorithm. It runs in $O(\log n)$ time using n processors on an EREW PRAM. The second algorithm is an RNC $\frac{1}{2\Delta+4}$-approximation algorithm. It runs in $O(\log \Delta)$ time using n processors on an EREW PRAM. Remark that the time complexity of the second algorithm does not depend on the size of the graph; the algorithm can be performed efficiently in parallel even on a large scale distributed system.

The rest of the paper is organized as follows. In section 2, we give basic definitions of the problem and state a useful lemma for trees. We then present the first algorithm in section 3. In the section, we modify the first algorithm and obtain the second algorithm. The approximation ratios are analyzed in section 4. In section 5, we consider the limits of the algorithms and conclude the paper.

2 Preliminaries

We will deal only with graphs without loops or multiple edges. Throughout the paper, unless stated otherwise, $G = (V, E)$ always denotes the input (undirected) graph, and w always denotes a map from E to $\mathbb{R}^+$, set of positive real numbers. For each edge e in E, we call $w(e)$ the *weight* of the edge. We also denote the sum of the weights of edges in a subset $F \subseteq E$ by $w(F)$. We denote by n and m the number of vertices and edges, respectively. Without loss of generality, we assume that $w(e_1) \neq w(e_2)$ for each $e_1, e_2 \in E$.

The *neighborhood* of a vertex v in G, denoted by $N_G(v)$, is the set of vertices in G adjacent to v. The *degree* of a vertex v in G is $|N_G(v)|$, and denoted by $\deg_G(v)$. The *maximum degree* of a graph G is $\max_{v \in V} \deg_G(v)$, and denoted by Δ_G. Without loss of generality, we assume that $\Delta_G > 2$. The notations $\deg_G(v)$, $N_G(v)$, and Δ_G is sometimes denoted by just $\deg(v)$, $N(v)$, and Δ if G is understood. For $F \subseteq E$, $G[F]$ denotes the graph (V, F).

A *matching* of G is a subset $M \subseteq E$, such that no two edges of M are adjacent. A matching M is *maximal* if that is not properly contained in another matching. A matching M is a *maximum weighted matching* if $w(M) \geq w(M')$ for any other matching M' of G. Remark that a maximum weighted matching is a maximal matching since $w(e) > 0$ for all $e \in E$. A vertex incident to an edge of M is called *matched* (by M) and a vertex not incident to an edge of M is called *free*.

The MWM problem is to find a maximum weighted matching for given graph G. A deterministic algorithm is *δ-approximation algorithm* for the MWM problem if the algorithm produces a matching M such that $w(M)$ is at least $\delta w(M^*)$, where M^* is a maximum weighted matching. A randomized algorithm is *δ-approximation algorithm* for the MWM problem if the expected value of the weight of a matching produced by the algorithm is at least $\delta w(M^*)$.

An acyclic connected graph, one not containing any cycles, is called a *tree*. In a tree, the *leaves* are the vertices of degree 1, and the *internal vertices* are the vertices which are not leaves. We first state a proposition and a useful lemma for a tree:

Proposition 1 Let T be a tree with n vertices. We let n_i be the number of vertices of degree i with $1 \leq i \leq \Delta_T$. Then (a) $\sum_{i=1}^{\Delta_T} n_i = n$; and (b) $\sum_{i=1}^{\Delta_T} i n_i = 2(n-1)$.

Proof. (a) is trivial. When each vertex counts up its degree, each edge is counted exactly twice. Since any tree with n vertices has $n-1$ edges [Har72, Chapter 4], (b) follows. ∎

Lemma 2 Let $L(n, \Delta)$ be the maximum number of leaves in a tree of maximum degree Δ with n vertices. Then $L(n, \Delta) \leq \frac{(\Delta-2)n-2}{\Delta-1}$.

Proof. Let T be an n vertex tree with $L(n,\Delta)$ leaves. Let V_I be the set of internal vertex of T. Then, the graph induced by V_I is also a tree. It contains $|V_I|$ vertices and $|V_I|-1$ edges. Each leaf of T is incident to one internal vertex. Moreover, each vertex in V_I can be incident to at most Δ vertices. Thus we have $L(n,\Delta) \leq \Delta |V_I| - 2(|V_I| - 1)$. We also have $L(n,\Delta) + |V_I| = n$. Hence $L(n,\Delta) \leq \frac{(\Delta-2)n-2}{\Delta-1}$. ∎

Recall that the EREW PRAM is the parallel model where the processors operate synchronously and share a common memory, but no two of them are allowed simultaneous access to a memory cell (whether the access is for reading or for writing in that cell). We here clarify the operation on the weights of edges. The algorithms use one *comparison* operation to determine the heavier edge between two edges in unit cost. This assumption is natural and it can be implemented on an ordinary PRAM storing the total order of the weights.

In this paper, each algorithm uses n processors; every vertex in G has a processor associated with it. As the input representation of G, we assume that each vertex has a list of the edges incident to it. Thus, each edge $\{i,j\}$ has two copies - one in the edge list for vertex i and the other in the edge list for vertex j.

3 Algorithms

The deterministic parallel approximation algorithm contains three phases:

Algorithm 1

1. For given G, construct a heavy spanning forest F (defined later) of G;
2. construct a set of paths P in $G[F]$;
3. produce a matching M_1 in $G[P]$.

The matching M_1 is the output of the algorithm. We describe phase by phase, and show the complexity of algorithm. We also show some useful lemmas for F, P, and M_1. Finally, we modify Algorithm 1 and obtain the randomized parallel approximation algorithm.

3.1 The First Phase

The first phase of the algorithm contains two steps:

1.1. In parallel, each vertex marks the heaviest edge incident to the vertex;

1.2. F is the set of marked edges.

We first show that $G[F]$ is acyclic.

Proposition 3 $G[F]$ is acyclic.

Proof. Assume $G[F]$ is not acyclic and $e_1, e_2, \cdots, e_l$ are edges producing a cycle in $G[F]$. We let $v_1, v_2, \cdots, v_l$ be vertices on the cycle. If two consecutive vertices in these vertices mark the same edge, there should be an edge on the cycle not marked by any vertices. Hence each vertex marks different edge. Thus we can assume that v_i marks e_i, and $w(e_1) < w(e_2)$. However, this implies that $w(e_1) < w(e_2) < \cdots < w(e_l) < w(e_1)$, that is a contradiction. ∎

Thus, $G[F]$ is a set of trees. Moreover, it is trivial that $\deg_{G[F]}(v) > 0$ for all v in V. Hence we call F *heavy spanning forest* of G. Hereafter, we denote by c the number of trees in $G[F]$.

We now introduce some notions for the heavy spanning forest F. Let T be a tree in $G[F]$, and n_T be the number of vertices in T. Then, in the first step, each of n_T vertices in T marks one edge, and T has $n_T - 1$ edges. This implies that T has exactly one edge marked by two vertices. We call the edge and two vertices *a root edge* and *two roots* of T, respectively. That is, F contains c trees, and each tree has one root edge and two roots. We can show the following lemma by a simple induction.

Lemma 4 Let T be a tree in F, and e_r be the root edge of T. Then for any leaf-root path (e_l, e_1, e_2, $\cdots$, e_r) in T, $w(e_l) < w(e_1) < w(e_2) < \cdots < w(e_r)$.

That is, the root edge is the heaviest edge in the tree.

Let M^* be a maximum weighted matching in G. We now show the theorem for the relation between $w(M^*)$ and $w(F)$.

Theorem 5 $w(F) \geq w(M^*)$.

Proof. Let $e = \{u, v\}$ be an edge in M^*, but not in F. Then, since $e \notin F$, e is not marked by both u and v. Let e_u and e_v be edges marked by u and v, respectively. Since M^* is a matching, both e_u and e_v are not in M^*. That is, $\{e_u, e_v\} \subseteq F - M^*$. Now we divide the weight $w(e)$ in two weights $\frac{1}{2}w(e)$, and *pile* them onto $\frac{1}{2}w(e_u)$ and $\frac{1}{2}w(e_v)$, respectively. Since e is not marked, $w(e) < w(e_u), w(e_v)$. Moreover, e_u and e_v are not piled by the other edges in M^* at the points u and v since e is an element in the matching M^*. That is, no edge e in $F - M^*$ will be piled more than $w(e)$ by the edges in $M^* - F$. Thus each edge in M^* is either in F or it can be divided and piled onto two edges in $F - M^*$. This implies that $w(F) \geq w(M^*)$. ∎

We moreover analyze the proof of Theorem 5 in detail. We now fix any maximum weighted matching M^*. Let $C = F \cap M^*$, $\hat{F} = F - C$, and $\hat{M} = M^* - C$. We let R be the set of the root edges of F. Then we have the following corollary.

Corollary 6 $w(F) \geq 2w(M^*) - w(C) - w(R)$.

Proof. In the proof of Theorem 5, each weight of an edge in $\hat{M}$ is divided and piled onto two edges in $\hat{F}$, because corresponding edges in $\hat{F}$ can be piled at both endpoints. However, only root edges can be piled at both endpoints. Now we pile each weight of an edge in $\hat{M}$ onto two edges in $\hat{F}$ without division. In the case, root edges may be piled twice. The observation implies that $w(\hat{F}) \geq 2w(\hat{M}) - w(R)$. Thus $w(F) = w(\hat{F}) + w(C) \geq 2w(\hat{M}) - w(R) + w(C) = 2w(M^* - C) - w(R) + w(C) = 2w(M^*) - w(C) - w(R)$. ∎

3.2 The Second Phase

The second phase of the algorithm easy to describe:

2.1. In parallel, each vertex v with $\deg_{G[F]}(v) > 2$ deletes all edges incident to v except two heaviest edges. Let P be the set of remaining edges. (Comment: It is easy to see that $G[P]$ is a set of paths, and $\deg_{G[P]}(v) > 0$ still holds if v was not a leaf in F.)

Theorem 7 $w(P) \geq \frac{1}{\Delta_{G[F]} - 1} w(F)$.

Proof. We first assume that $G[F]$ contains only one tree. We let n_i be the number of vertices of degree i in $G[F]$ with $1 \leq i \leq \Delta$. Then the number of edges deleted in step 2.1 is equal to $\sum_{i=3}^{\Delta}(i-2)n_i = \sum_{i=3}^{\Delta} in_i - 2\sum_{i=3}^{\Delta} n_i = 2(n-1) - n_1 - 2n_2 - 2(n - n_1 - n_2) = n_1 - 2$. Using Lemma 2, we have $\frac{|P|}{|F|} = \frac{n-1-n_1+2}{n-1} \geq \frac{n+1-L(n,\Delta)}{n-1}$. Thus, using Lemma 2, we have $\frac{n+1-L(n,\Delta)}{n-1} \geq \frac{(\Delta-1)(n+1)-(\Delta-2)n+2}{(n-1)(\Delta-1)} = \frac{n+\Delta+1}{(n-1)(\Delta-1)} = \frac{1}{\Delta-1} + \frac{\Delta+2}{(n-1)(\Delta-1)} \geq \frac{1}{\Delta-1}$.

We then consider the weights of deleted edges. For each deleted edge e, there exists at least one edge e' in P with $w(e') > w(e)$. On the other hand, for each remaining edge e' in P, it is corresponded by such deleted edges e at most $\Delta - 1$ times. This together with $\frac{|P|}{|F|} > \frac{1}{\Delta-1}$ implies that $\frac{w(P)}{w(F)} > \frac{1}{\Delta-1}$.

When $G[F]$ contains two or more trees, the discussion above can be applied on tree by tree. ∎

3.3 The Third Phase

We define the distance of edges to describe the third phase. Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path of length l. Then the *distance* of e_i from e_1 on α, denoted by $D(e_i, e_1)$, is defined by $D(e_1, e_1) = 0$, and $D(e_i, e_1) = D(e_{i-1}, e_1) + 1$ for $1 < i \leq l$. The third phase contains the following steps:

3.1. In parallel, find the heaviest edge in each path in P.

3.2. M_1 is the set of edges having even distance from the heaviest edge on the same path.

For any given path, we call a set of edges an *alternating path* on the path if it is a maximal matching on the path, and so is its complement. Using this notation, we can rewrite step 3.2 as follows: In each path in P, each edge in the alternating path containing heaviest edge is in M_1.

We here show a proposition and a useful lemma for paths with special properties.

Proposition 8 Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path with $w(e_1) > w(e_2) > \cdots > w(e_l)$. Then the maximum weighted matching M_α is either $\{e_1, e_3, \cdots, e_l\}$ for odd l, or $\{e_1, e_3, \cdots, e_{l-1}\}$ for even l. Moreover, $w(M_\alpha) \geq \frac{1}{2}w(\alpha)$.

Proof. When l is odd, considering $w(e_1) > w(e_2)$, $w(e_3) > w(e_4)$, $\cdots$, $w(e_{l-1}) > w(e_l)$, we immediately have the proposition. When l is even, the last edge e_l just increases the weight of M_α. ∎

Lemma 9 Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path such that $w(e_1) < w(e_2) < \cdots < w(e_{i-1}) < w(e_i) > w(e_{i+1}) > \cdots > w(e_l)$ for some i with $1 < i < l$. Let A_1 be the alternating path containing e_i, and A_2 be the other alternating path. Then

(1) Either A_1 or A_2 is the maximum weighted matching of α, and that has at least half weight of α.
(2) When A_2 is the maximum weighted matching of α, $w(e_{i-1}) + w(e_{i+1}) \geq w(e_i)$.
(3) $w(A_1) \geq \frac{1}{3}w(\alpha)$.

Proof. (1) We first show that M_α satisfies either (a) $e_i \in M_\alpha$ or (b) $\{e_{i-1}, e_{i+1}\} \subseteq M_\alpha$. To derive a contradiction, assume that $e_i \notin M_\alpha$, $e_{i-1} \notin M_\alpha$, and $e_{i+1} \in M_\alpha$. Then $(M_\alpha - \{e_{i+1}\}) \cup \{e_i\}$ is a matching heavier than M_α since $w(e_i) > w(e_{i+1})$, that is a contradiction. The other symmetric case ($e_i \notin M_\alpha$, $e_{i-1} \in M_\alpha$, and $e_{i+1} \notin M_\alpha$) can be shown by the same argument. In the case (a), we have $e_i \in M_\alpha$, $e_{i-1} \notin M_\alpha$, and $e_{i+1} \notin M_\alpha$. Then we can consider α as two paths $(e_1, e_2, \cdots, e_{i-2})$ and $(e_{i+2}, \cdots, e_l)$ separately. Using Proposition 8, we have $M_\alpha = \{e_i\} \cup \{e_{i-2}, e_{i-4}, \cdots\} \cup \{e_{i+2}, e_{i+4}, \cdots\}$, that is an alternating path. In the case (b), we have $e_i \notin M_\alpha$, $e_{i-1} \in M_\alpha$, and $e_{i+1} \in M_\alpha$. Thus we have $M_\alpha = \{e_{i-1}, e_{i+1}\} \cup \{e_{i-3}, e_{i-5}, \cdots\} \cup \{e_{i+3}, e_{i+5}, \cdots\}$, that is also an alternating path. Heavier alternating path clearly has at least half weight of α. This completes the proof of (1).
(2) Let A_2' be $(A_2 - \{e_{i-1}, e_{i+1}\}) \cup \{e_i\}$. Then A_2' is a matching. Since A_2 is the maximum weighted matching, $w(A_2') \leq w(A_2)$. This implies that $w(e_{i-1}) + w(e_{i+1}) \geq w(e_i)$.
(3) When A_1 is the maximum weighted matching, it follows from Proposition 8. Thus we assume that A_1 is not the maximum weighted matching. Then, as in (1), A_2 is the maximum weighted matching.

We here consider two paths $\alpha_1 = (e_1, e_2, \cdots, e_{i-1}, e_i)$ and $\alpha_2 = (e_i, e_{i+1}, \cdots, e_l)$. Let A_1^1 (and A_1^2) be the alternating path of α_1 (and α_2, resp.) containing e_i. That is, A_1^1 is the former half of A_1, A_1^2 is the latter half of A_1, and $A_1^1 \cap A_1^2 = \{e_i\}$. Then, by Proposition 8, $w(A_1^1) \geq \frac{1}{2}w(\alpha_1)$ and $w(A_1^2) \geq \frac{1}{2}w(\alpha_2)$.

Thus, $w(A_1) = w(A_1^1 \cup A_1^2) = w(A_1^1) + w(A_1^2) - w(A_1^1 \cap A_1^2) \geq \frac{1}{2}(w(\alpha_1) + w(\alpha_2)) - w(e_i) = \frac{1}{2}(w(\alpha) + w(e_i)) - w(e_i) = \frac{1}{2}(w(\alpha) - w(e_i)) \geq \frac{1}{2}(w(\alpha) - w(A_1))$. This implies that $w(A_1) \geq \frac{1}{3}w(\alpha)$. ∎

We here remark that, in Lemma 9(1), we cannot determine in general which alternating path is heavier. (For example, a path (e_1, e_2, e_3) has different answers when $w(e_1) = 1, w(e_2) = 3, w(e_3) = 1$ and $w(e_1) = 2, w(e_2) = 3, w(e_3) = 2$). We also remark that Lemma 9(1) does not hold for general weighted path (for example, each alternating path of (e_1, e_2, e_3, e_4) is not the maximum weighted matching for $w(e_1) = 5, w(e_2) = 1, w(e_3) = 1, w(e_4) = 5$).

We now show the relation between $w(M_1)$ and $w(P)$.

Lemma 10 $w(M_1) \geq \frac{1}{3}w(P)$.

Proof. We first observe the following claim: in step 2.1, if vertex v delete an edge $\{u, v\}$, the edge was marked by u in step 1.1. This is easy because each vertex marked the heaviest edge in step 1.1, and remains the heaviest edge(s) in step 2.1. Using the claim and simple induction, we can show that each path in P is either

(1) a part of some leaf-root path in some tree in F; or

(2) two parts of leaf-root paths connected by the root edge in a tree in F.

In the case (1), combining Lemma 4 and Proposition 8, M_1 contains the maximum weighted matching of the path, and that has at least half weight of the path. Thus it is sufficient to show for the case (2). By Lemma 4, the path $\alpha = (e_1, e_2, \cdots, e_l)$ satisfies that $w(e_1) < w(e_2) < \cdots < w(e_{r-1}) < w(e_r) >$

$w(e_{r+1}) > \cdots > w(e_l)$, where e_r is the root edge. Thus, according to Lemma 9(3), for the alternating path A_r containing e_r, $w(A_r) \geq \frac{1}{3}w(\alpha)$. Thus $w(M_1) \geq \frac{1}{3}w(P)$. ∎

Combining Theorem 5, Theorem 7, and Lemma 10, we can show that Algorithm 1 is a $\frac{1}{3(\Delta-1)}$-approximation algorithm. But the better approximation ratio $\frac{2}{3\Delta+2}$ will be stated in Section 4.

3.4 Complexity of Algorithm 1

Theorem 11 Algorithm 1 is the NC algorithm that runs in $O(\log n)$ time using n processors on an EREW PRAM. It requires only comparison operation.

Proof. We first assume that each processor has two memory cells to store the edges in P. Then the first and second phases can be efficiently implemented modifying the following:

1.1'. In parallel, each vertex v finds the heaviest edge $e = \{v, u\}$ incident to v;

1.2'. In parallel, v stores the first cell of v with e.

2.1'. In parallel, each vertex v checks the contents of the first cell of u. If it is e, then the process is end. If it is not e, v tries to store the second cell of u with e. This trial will succeed if $w(e)$ is the heaviest among the other edges that are tried to store the same cell.

The step 1.2' can be done in a unit time. Moreover, it is not difficult to see that the steps 1.1' and 2.1' can be done in $O(\log \Delta)$ time using standard technique with comparison operation.

In the third phase, we can easy to see the following:
(1) if $e = \{u, v\}$ is a root edge, e is stored in the first cells of both u and v; and
(2) otherwise, e is stored in the first cell of one endpoint, and in the second sell of the other.

Moreover, each non-root edge knows which endpoint is close to the root edge; the endpoint storing the second cell with the edge. Thus step 3.1 can be done in $O(1)$ time, and step 3.2 can be done in $O(\log n)$ time using standard list ranking technique (see e.g., [KR90]). We remind that, throughout the computation, only comparison operation is required. ∎

3.5 Algorithm 2

In Algorithm 1, almost all computations are performed "locally". That is, all computations can be performed within the neighbors except steps 3.1 and 3.2. Algorithm 2 is the same as Algorithm 1 except the third phase. We modify the steps 3.1 and 3.2, and obtain the RNC approximation algorithm using randomization.

3.1'. In parallel, each vertex randomly choose one of two edges incident to the vertex in $G[P]$. (The vertices of degree one choose the unique edge incident to the vertex.)

3.2'. M_2 is the set of edges chosen by both endpoints.

Since each vertex choose one edge, the resulting M_2 is a matching. These modified steps can be performed in $O(1)$ time. This immediately implies the following theorem.

Theorem 12 Algorithm 2 is the RNC approximation algorithm that runs in $O(\log \Delta)$ time using n processors on an EREW PRAM. It requires only comparison operation.

4 Analysis of Approximation Ratios

We remind that M^* is any fixed maximum weighted matching, F is the heavy spanning forest, R is the set of the root edges of F, and P is a set of paths obtained in step 2.1. Moreover we let $C = F \cap M^*$, $\hat{F} = F - C$, and $\hat{M} = M^* - C$.

To derive good approximation ratios, we define two maximum matchings: M_P denotes any fixed maximum weighted matching of $G[P]$, and M_F denotes any fixed maximum weighted matching of $G[F]$.

Lemma 13 $w(M_P) \geq \frac{1}{2(\Delta-1)}w(F)$.

Proof. As seen in the proof of Lemma 10, each path in P is either

(1) a part of some leaf-root path in some tree in F; or

(2) two leaf-root paths connected by a root edge in a tree in F.

For each path, by Lemma 9(1), M_P contains heavier alternating path that has at least half weight of the path. This together with Theorem 7 implies that $w(M_P) \geq \frac{1}{2}w(P) \geq \frac{1}{2(\Delta-1)}w(F)$. ∎

Lemma 14 $w(M_F) \geq \frac{1}{\Delta}w(M^*)$.

Proof. We first remind that M_F is the maximum weighted matching in F. Thus, since C is a matching in F, $w(M_F) \geq w(C)$. It is easy to see that R is a matching in F. This implies that $w(M_F) \geq w(R)$. It is also easy to see that M_P is a matching in F, and thus $w(M_F) \geq w(M_P)$. Hence, combining Corollary 6, we have $w(F) \geq 2w(M^*) - w(C) - w(R) \geq 2w(M^*) - 2w(M_F)$. On the other hand, by Lemma 13, $w(M_F) \geq w(M_P) \geq \frac{1}{2(\Delta-1)}w(F)$. Combining the equations, we have $(2(\Delta-1)+2)w(M_F) \geq 2w(M^*)$, consequently, $w(M_F) \geq \frac{1}{\Delta}w(M^*)$. ∎

Lemma 15 $w(C) \leq w(M_F) \leq 2w(M_P)$.

Proof. It is clear that $w(C) \leq w(M_F)$. Thus we show $w(M_F) \leq 2w(M_P)$. We are going to show that each edge in M_F can be piled onto an edge in M_P, and each edge in M_P is piled by such edges at most twice. Let $e = \{u, v\}$ be any edge in M_F. Then three cases occur according to e.
(1) $e \in M_P$. We pile e onto itself.
(2) $e \in P - M_P$. We first assume that e is not a root edge. We assume that u is closer to the root edge than v on $G[F]$. In the case, e is incident to e' in P at the vertex u with $w(e') > w(e)$. Thus we pile $w(e)$ onto $w(e')$. We next assume that e is a root edge. That is, e is a root edge not in the maximum weighted matching of $G[P]$. Then, by Lemma 9, M_P contains two edges e' and e'' such that e' and e'' are the edges incident to e at vertex u and v, respectively, and $w(e') + w(e'') \geq w(e)$. Thus we divide $w(e)$ into $w(e')$ and $w(e) - w(e')(\leq w(e''))$, and pile them onto $w(e')$ and $w(e'')$, respectively.
(3) $e \notin M_P$. We assume that u is closer to the root edge than v on $G[F]$. In the case, e was deleted by u in step 2.1. The vertex u remains two edges e' and e'' in P with $w(e'), w(e'') > w(e)$. Moreover, either e' or e'' is in M_P. Thus we pile $w(e)$ onto the edge in M_P.

Since M_F is a matching, no two edges are piled at the same endpoint. Thus each edge in M_P is piled at most twice at both endpoints. This implies that $w(M_F) \leq 2w(M_P)$. ∎

Theorem 16 $w(M_P) \geq \frac{2}{2\Delta+1}w(M^*)$.

Proof. Combining Corollary 6 and Lemma 13, we get $w(M_P) \geq \frac{1}{2(\Delta-1)}w(F) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(R) - w(C))$. Using Lemma 15, we have $2w(M_P) \geq w(C)$. On the other hand, since $R \subseteq M_1$, $w(M_P) \geq w(M_1) \geq w(R)$. Thus, $w(M_P) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(R) - w(C)) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(M_P) - 2w(M_P))$. Thus $w(M_P) \geq \frac{2}{2\Delta+1}w(M^*)$. ∎

Theorem 17 The approximation ratio of Algorithm 1 is $\frac{2}{3\Delta+2}$.

Proof. We first show that $w(M_1) \geq \frac{1}{2}w(M_P)$. As seen in the proof of Lemma 10, each path in P is either

(1) a part of some leaf-root path in some tree in F; or

(2) two leaf-root paths connected by a root edge in a tree in F.

In the case (1), both M_1 and M_P contains the same alternating path that contains the heaviest edge. We consider the paths in the case (2). Let α be the path in P, and A_1 be the alternating path of α containing the root edge, and A_2 be the other alternating path. According to Lemma 9, A_1 or A_2 is the maximum weighted matching of α. When A_1 is the maximum weighted matching, both M_1 and M_P contains it. Now we assume that A_2 is the maximum weighted matching of α. Then, by Lemma 9(3), $w(A_1) \geq \frac{1}{3}w(\alpha)$, consequently, $w(A_2) \leq \frac{2}{3}w(\alpha)$. Thus $w(A_1) \geq \frac{1}{2}w(A_2)$. Therefore, in any cases, $w(M_1) \geq \frac{1}{2}w(M_P)$.

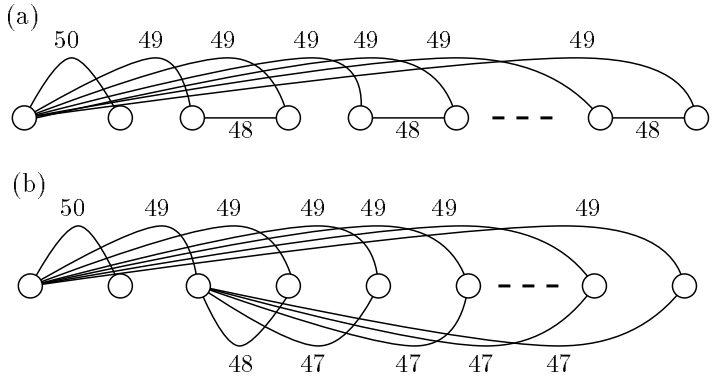

Figure 1: Bad examples for the algorithms

Combining Corollary 6, Theorem 7, and Lemma 10, we have $w(M_1) \geq \frac{1}{3}w(P) \geq \frac{1}{3(\Delta-1)}w(F) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - w(C) - w(R))$. It is clear that $w(M_1) \geq w(R)$ since $R \subseteq M_1$. Thus, using Lemma 15, we have $w(M_1) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - w(C) - w(R)) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - 2w(M_P) - w(M_1)) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - 5w(M_1))$, consequently, $w(M_1) \geq \frac{2}{3\Delta+2}w(M^*)$. ∎

Theorem 18 The approximation ratio of Algorithm 2 is $\frac{1}{2\Delta+4}$.

Proof. Each edge in P is chosen with probability at least $\frac{1}{4}$. Thus, the expected value of $w(M_2)$ is at least $\frac{1}{4}w(P)$. Using Corollary 6 and Theorem 7, we have $E(w(M_2)) \geq \frac{1}{4(\Delta-1)}w(F) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - w(C) - w(R))$.

We now compare $w(M_2)$ with $w(M_P)$. Each edge in M_P appears in M_2 with probability at least $\frac{1}{4}$. This implies that the expected value of $w(M_2)$ is at least $\frac{1}{4}w(M_P)$. Thus, using Lemma 15, we have $E(w(M_2)) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - w(C) - w(R)) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - 3w(M_P)) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - 12E(w(M_2)))$, consequently, $E(w(M_2)) \geq \frac{1}{2\Delta+4}w(M^*)$. ∎

5 Concluding Remarks

In this paper, we state two parallel approximation algorithms for the maximum weighted matching problem. In this section, we briefly show the limits of the algorithms. Each approximation ratio contains the factor $\frac{1}{\Delta}$. This factor is hard to improve by reason of the strategy in the first phase. Figure 1(a) is a bad example for the algorithms. At the first phase, all algorithms mark the edges of weights 50 and 49, and then they miss all edges of weights 48. In the time, the approximation ratio for the graph in Figure 1(a) is actually $\Theta(\frac{1}{\Delta})$. For such graph, it seems to be effective to repeat the algorithms until the matching becomes maximal. (The matchings produced by the algorithms are not maximal in general.) However, there exists the graphs such that the repeating algorithm takes $\Theta(n)$ time; see the graph in Figure 1(b) that recursively contains the upper half structure of the graph in Figure 1(a). (In the figure edges are written only in the first and second levels.)

References

[Avi83] D. Avis. A Survey of Heuristics for the Weighted Matching Problem. *Networks*, 13:475–493, 1983.

[Edm65] J. Edmonds. Paths, Trees and Flowers. *Canad. J. Math.*, 17:449–467, 1965.

[Gab90] H.N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proc. 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 434–443. ACM, 1990.

[Gal86] Z. Galil. Efficient Algorithms for Finding Maximum Matching in Graphs. *Computing Surveys*, 18(1):23–38, 1986.

[GHR95] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation.* Oxford University Press, 1995.

[Har72] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[KR90] R.M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, pages 870–941. Elsevier, 1990.

[KR98] M. Karpinski and W. Rytter. *Fast Parallel Algorithms for Graph Matching Problems.* Clarendon Press, 1998.

[KUW86] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a Perfect Matching is in Random NC. *Combinatorica*, 6(1):35–48, 1986.

[MS92] E.W. Mayr and A. Subramanian. The Complexity of Circuit Value and Network Stability. *Journal of Computer and System Science*, 44:302–323, 1992.

[Paw87] S. Pawagi. Parallel Algorithm for Maximum Weight Matching in Trees. In *Proc. International Conference on Parallel Processing*, pages 204–206. IEEE, 1987.

[Pre99] R. Preis. Linear Time $\frac{1}{2}$-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *STACS '99*, pages 259–269. Lecture Notes in Computer Science Vol. 1563, Springer-Verlag, 1999.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Dover, 1982.

[Ven87] S.M. Venkatesan. Approximation Algorithms for Weighted Matching. *Theoretical Computer Science*, 54:129–137, 1987.