New Approximation Algorithms for the Weighted Matching Problem.

Sven Hanke & Stefan Hougardy

Humboldt-Universität zu Berlin
Institut für Informatik
10099 Berlin, GERMANY
sven.hanke@gmx.de, hougardy@informatik.hu-berlin.de

TOPIC: Algorithms and data structures

Abstract. Finding matchings of maximum weight is a classical problem in combinatorial optimization. The fastest algorithm known today for solving this problem has running time $O(mn+n^2 \log n)$. Several applications require algorithms for the weighted matching problem with better running time. Up to now no algorithm with $o(mn)$ running time was known that achieves an approximation ratio better than $\frac{2}{3}$. We present two approximation algorithms for the weighted matching problem in graphs with an approximation ratio of $\frac{3}{4} - \epsilon$ respectively $\frac{4}{5} - \epsilon$ and running time $O(m \log n)$ respectively $O(m \log^2 n)$.

Keywords. approximation algorithms, analysis of algorithms, graph algorithms, maximum weight matching

1 Introduction

The weighted matching problem in graphs is to find a matching of maximum weight in an edge weighted graph. In 1965 Edmonds [5] presented the first polynomial time algorithm for this problem. The running time of his algorithm was improved over the years and the fastest algorithm currently known for the weighted matching problem has running time $O(mn+n^2 \log n)$ and is due to Gabow [7]. Many applications require faster algorithms for the weighted matching problem. Therefore one is interested in approximation

algorithms for the weighted matching problem, i.e., algorithms that guarantee to find a matching whose weight is at least some constant fraction of the largest possible weight. A very simple such algorithm is the greedy algorithm. This algorithm has running time $O(m \log n)$ and always finds a matching of weight at least $1/2$ of the maximum possible weight. In 1999 Preis [16] presented a linear time algorithm also achieving a factor $1/2$ approximation. Another such algorithm which is much simpler was presented in [3]. Drake and Hougardy [4] improved this result in 2004 by presenting a linear time algorithm which achieves a $2/3 - \epsilon$ approximation factor for arbitrarily small ϵ. Pettie and Sanders [15] presented another such algorithm whith better dependence of the running time on ϵ. No other constant factor approximation algorithms for the weighted matching problem are known that are faster than Gabow's exact algorithm, i.e, that have running time $o(mn)$.

Our Contribution In this paper we present two new approximation algorithms for the weighted matching problem. One has running time $O(m \log n)$ and achieves an approximation ratio of $3/4 - \epsilon$. The other algorithm has running time $O(m \log^2 n)$ and achieves an approximation ratio of $4/5 - \epsilon$. The algorithm of Drake and Hougardy [4] achieves an approximation ratio of $2/3 - \epsilon$ by considering augmentations which involve at most two non matching edges. It is easily seen that this approach can be generalized by considering augmentations which involve up to three non matching edges. This will result in an approximation algorithm whith approximation ratio $3/4 - \epsilon$. However, the running time of this algorithm is $O(mn)$.

We present two new ideas that allow to achieve an approximation ratio of $3/4-\epsilon$ with running time $O(m \log n)$. First we show that augmenting cycles of length six can be approximated reasonably well by overlapping augmenting paths of length seven. This requires a new definition of the gain of an augmentation. Secondly we show that by approximating the gain of augmenting paths of length seven and by using appropriate data structures, one can consider many augmentations simultaneously in one step. By combining these two ideas and repeating the algorithm a constant number of time we can show that it results in an $3/4-\epsilon$ approximation ratio with running time $O(m \log n)$. By extending these ideas we are able to show that augmentations involving up to four non matching edges can be handled similarly. This results in an approximation algorithm with approximation rato $4/5-\epsilon$ and running time $O(m \log^2 n)$.

2 Preliminaries

We denote by n the number of vertices and by m the number of edges in a graph. A *matching* M in a graph $G = (V, E)$ is a subset of the edges E of G such that no two edges in M have a vertex in common. A vertex $v \in V$ is *covered* by a matching M, if there exists an edge in M that contains v. A vertex is called *uncovered* if it is not covered by M. For a vertex $v \in V$ and a matching M we denote by $M(v)$ all edges in M that are incident to v. As M is a matching, the set $M(v)$ is either empty or contains exactly one edge. For a matching M and a vertex $x \in V$ we define an *arm* of x as a path of length one or two starting in x with an edge not in M. If the path has length two, then the second edge must belong to M. The number of arms starting in a vertex is bounded by the degree of this vertex. Therefore we have:

Lemma 1 *A graph contains at most* $2m$ *arms.* □

In the following our graphs will always be *weighted*, i.e., together with the graph $G = (V, E)$ we have a function $w : E \to \mathbb{R}_+$ which assigns a positive weight to each of the edges of G. The weight $w(F)$ of a subset $F \subseteq E$ of the edges of G is defined as $w(F) := \sum_{e \in F} w(e)$. The *weighted matching problem* is to find a matching M in G that has maximum weight. Such a matching (which needs not to be unique) will be denoted by M^*. In this paper we present an *approximation algorithm* for the weighted matching problem, i.e., an algorithm that for any graph $G = (V, E)$ and any function $w : E \to \mathbb{R}_+$ returns a matching M such that $w(M) \geq c \cdot w(M^*)$, where c is a constant. The constant c is called the *approximation ratio*.

Let $G = (V, E)$ be a graph and $M \subseteq E$ be a matching. A path or cycle is called *alternating* if it contains alternately edges from M and $E \setminus M$. An *M-augmentation* A is an alternating cycle or path such that the symmetric difference of M and A, which we denote by $M \triangle A$, is again a matching. We consider an M-augmentation A to be a set of edges. If the matching M is clear from the context, we simply say augmentation instead of M-augmentation. The *gain* of an augmentation A is defined as $gain(A) := w(A \setminus M) - w(A \cap M)$, i.e., the gain of an augmentation is the difference of the weights of the two matchings M and $M \triangle A$. Note that the gain of an augmentation may be negative.

3 A simple Approximation algorithm

We start by presenting a very simple approximation algorithm for the weighted matching problem which achieves in polynomial time an approximation ra-

```
SimpleMatching: G = (V, E), w : E → R+, l ∈ ℕ, matching M
Output: matching M'
 1  ALG = ∅
 2  compute the set A_l
 3  while A_l ≠ ∅ do
 4      choose A ∈ A_l with maximum gain
 5      ALG := ALG ∪ {A}
 6      remove all elements from A_l that are not vertex disjoint with A
 7  endwhile
 8  M' = M augmented by all augmentations in ALG
```

Figure 1: A simple algorithm for increasing the weight of a matching M.

tio arbitrarily close to one. Even though the running time of this algorithm is much too high, it will be the starting point for our new approximation algorithm for the weighted matching problem and is useful to elucidate some of its ideas.

Let $G = (V, E)$ be a graph and $M \subseteq E$ be a matching. We denote by $\mathcal{A}_l$ the set of all M-augmentations which contain at most l edges not in M. For fixed l the set $\mathcal{A}_l$ can easily be computed in polynomial time. Figure 1 shows a simple polynomial time algorithm for improving the weight of a matching.

To analyse the algorithm `SimpleMatching` we need the following definition of a multiset Ω_l whose elements are from $\mathcal{A}_l$. Consider the symmetric difference of M and M^*. This consists of connected components which are M-augmentations. If such an M-augmentation contains at most l edges from M^* then put it into Ω_l with multiplicity l. If it contains more than l edges from M^* then label the edges of M^* within this augmentation consecutively. For any l consecutive edges (using wrap around) of M^* in this augmentation we include the M-augmentation containing these l edges of M^* in Ω_l. If a wrap around in a path appears, this will result in two M-augmentations that together contain l edges of M^* (see Figure 2).

Theorem 1 $\sum_{A \in \Omega_l} gain(A) \geq l \cdot w(M^*) - (l+1) \cdot w(M)$

Proof. Each edge of $M^* \setminus M$ is contained in precisely l augmentations from Ω_l. Each edge of $M \setminus M^*$ is contained in at most $l+1$ augmentations from Ω_l. Therefore we get $\sum_{A \in \Omega_l} gain(A) \geq l \cdot w(M^* \setminus M) - (l+1) \cdot w(M \setminus M^*) = l \cdot w(M^*) - (l+1) \cdot w(M) + w(M \cap M^*) \geq l \cdot w(M^*) - (l+1) \cdot w(M)$. □

Theorem 2 *If the algorithm* `SimpleMatching` *gets a matching M as input then it returns a matching M' such that*

$$w(M') \;\geq\; w(M) + \frac{1}{2(l+1)} \cdot \left(\frac{l}{l+1} \cdot w(M^*) - w(M) \right) \;.$$

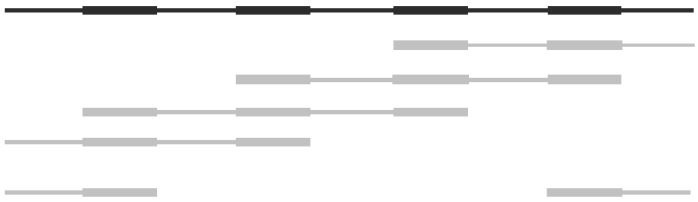

Figure 2: Illustration of the definition of the set Ω_l for $l = 2$. The figure shows at the top an augmentation in the symmetric difference of M and M^* which contains five edges of M^*. The edges of M are in bold. Below are the six augmentations that are included in Ω_2.

Proof. The algorithm `SimpleMatching` stops when the set $\mathcal{A}_l$ is empty. Therefore it must be the case that for every augmentation S in Ω_l there exists an augmentation A in ALG with $gain(S) \leq gain(A)$ and A and S are not vertex disjoint. Assign S to one of the vertices in the augmentation A which they have in common. By definition each vertex of the graph G is contained in at most $l+1$ augmentations of Ω_l and therefore at most $l+1$ augmentations can be assigned to it. Each augmentation of ALG has at most $2(l+1)$ vertices. Therefore it follows that

$$gain(ALG) \;\geq\; \frac{1}{2(l+1)^2} \sum_{S \in \Omega_l} gain(S) \;. \tag{1}$$

Now the result follows from Theorem 1. □

Theorem 3 *If there exists an algorithm A that by input of a matching M returns a matching M' with $w(M') \;\geq\; w(M) + \alpha\,(\beta \cdot w(M^*) - w(M))$, then for every $\epsilon > 0$ there exists an algorithm A' that finds a matching M'' of weight at least $w(M'') \geq \beta \cdot (1-\epsilon) \cdot w(M^*)$. The running time of A' is only by a constant factor (which depends on ϵ) larger than that of A.*

Proof. Repeat the algorithm A a constant number of times. Let M_0 be the empty matching and M_{i+1} be the matching that is obtained from M_i by applying the algorithm A. By assumption we have $w(M_{i+1}) \geq w_{i+1} \cdot w(M^*)$ where w_{i+1} is defined by the following recurrence

$$w_{i+1} \;=\; w_i + \alpha \cdot (\beta - w_i)\,, \text{ and } \quad w_0 = 0 \;.$$

By solving this linear recurrence equation we get

$$w(M_i) \;\geq\; \beta \cdot \left(1 - (1-\alpha)^i\right) \cdot w(M^*) \;.$$

This immediately shows that if i is larger than some constant $c \geq \log_\epsilon(1-\alpha)$ we have

$$w(M_c) \;\geq\; \beta \cdot (1-\epsilon) \cdot w(M^*) \;.$$

As the algorithm A is repeated c times and c is a constant (depending on ϵ) the result follows. □

Theorem 4 *For any $\delta > 0$ algorithm* `SimpleMatching` *yields an $1-\delta$ approximation algorithm for the weighted matching problem and has polynomial running time.*

Proof. By Theorem 2 we know that algorithm `SimpleMatching` returns a matching M' with

$$w(M') \;\geq\; w(M) + \frac{1}{2(l+1)} \cdot \left(\frac{l}{l+1} \cdot w(M^*) - w(M) \right) \;.$$

Set $l := 2/\delta - 1$. Now apply Theorem 3 with $\alpha = \frac{1}{2(l+1)}$, $\beta = \frac{l}{l+1}$ and $\epsilon = \frac{\delta}{2-\delta}$. This yields an algorithm with approximation ratio $\beta(1-\epsilon) = (1-\delta)$. The running time of this algorithm is at most $O(mn^{2/\delta-2} \log n)$. □

The running time of the algorithm resulting from the above theorem for $l \geq 3$ is larger than the running time of Gabow's exact matching algorithm. In the next section we will use several ideas to get a better running time while preserving the general approach that is used by the algorithm `SimpleMatching`.

4 A $3/4 - \epsilon$ approximation algorithm

In this section we show how to achieve an approximation ratio of $3/4 - \epsilon$ in time $O(m \log n)$. We will use a similar approach as in Section 3 but need several new ideas. First, instead of considering all augmentations in set $\mathcal{A}_3$, we will only consider $O(m \log n)$ of them. Secondly we will show that cycles of length 6 in $\mathcal{A}_3$ can be approximated by paths of length 7 in $\mathcal{A}_3$ if the gain function is redefined appropriately. Finally we will present data structures that allow to keep and update all necessary information for the algorithm.

In line 4 of the algorithm `SimpleMatching` an augmentation A from $\mathcal{A}_l$ is chosen with maximum gain. If instead we only compute an augmentation A' with a gain that achieves at least some γ-fraction of the maximum possible gain, then the proof of Theorem 2 shows, that Theorem 3 can still be applied with $\alpha = \frac{\gamma}{2(l+1)}$. Therefore one idea of our new algorithm is not to store all elements of $\mathcal{A}_3$ but to partition them appropriately and to keep only some information about the partition classes.

Let S be a finite set, $f : S \to \mathbb{R}$ be an arbitray weight function, and $\alpha > 0$, $\beta > 1$ be constants. Denote by f_{max} the maximum of $f(s)$ for all $s \in S$. Then we define the *rank* $r(s)$ of an element $s \in S$ as follows

- If $f(s) \leq \frac{\alpha \cdot f_{max}}{n}$ then $r(S) = 0$
- Otherwise $r(s) = i > 0$ where i is the smallest integer for which it is true that $f(s) \leq \beta^i \cdot \frac{\alpha \cdot f_{max}}{n}$.

We will call this ranking an (f, α, β)-ranking. The definition implies that the rank of an element s is an integer of size $O(\log_\beta \frac{n}{\alpha})$.

Let $G = (V, E)$ be a graph and M a matching. For each vertex $v \in V$ we will store an ordered list $arms_v$ which contains all arms of the vertex v ordered by their gains. By Lemma 1 these lists can be computed in $O(m \log n)$. Furthermore each vertex gets assigned $O(\log_\beta \frac{n}{\alpha})$ lists $arms_{v,i}$ for $i = 0, 1, \ldots$ which contain all rank i arms of v ordered by their gains of a $(gain, \alpha, \beta)$-ranking. The constants α and β will be specified later. Note that $gain_{max} \leq w_{max}$ where w_{max} is the maximum weight of an edge in E.

Let A be an augmentation in $\mathcal{A}_l$ that is a path. We call an edge c in A a *center* of A if $c \in M$ and the number of edges M in A on both sides of c differs by at most one. We will consider centers always as directed edges. For a directed edge $c = (x, y)$ we call an arm of x a *left arm* and we call an arm of y a *right arm*. An augmenting path in $\mathcal{A}_3$ has length at most seven. Therefore, it can be seen as a center $c = (x, y)$ together with a left arm starting in x and a right arm starting in y and an additional arm starting in the end vertex of the left arm of x. This additional arm will be called an *extension* of the left arm starting in x.

For each vertex x and each i we use a list $L_{x,i}$ to store all arms of x that can be extended by an arm of rank i. More precisely, let x be a vertex and $\{x, y\}, \{y, z\}$ be an arm of x. If $arms_{z,i}$ is not empty, then we put the arm $\{x, y\}, \{y, z\}$ into the list $L_{x,i}$. The lists $L_{x,i}$ can be created for all x and all i in $O(m \log_\beta \frac{n}{\alpha})$ by simply scanning all lists $arms_x$ and all lists $arms_{x,i}$. We also can assume that the arms within each list $L_{x,i}$ are ordered by their gains. Therefore we have

Theorem 5 *For all $x \in V(G)$ and all i the lists $arms_x, arms_{x,i}$ and $L_{x,i}$ can be computed in $O(m \log_\beta \frac{n}{\alpha})$. Furthermore we may assume that the elements within each list are ordered by their gain.* □

Let M be a matching and $e \in M$. Then using the data structure of $arms_x$ one can easily find an augmentation in $\mathcal{A}_2$ with e as a center that achieves the maximum gain. This was already shown in [1]. Here we extend this result in the following:

3/4-Matching: $G = (V, E), w : E \to R_+$, $l \in \mathbb{N}$, matching M
Output: matching M'

```
1   ALG = ∅
2   while M ≠ ∅ do
3       compute A_i(e⃗) for all e ∈ M, i ≥ 0 and both orientations of e
4       from these let A have maximum gain
5       ALG := ALG ∪ {A}
6       remove all vertices and edges from G and M that intersect A
7   endwhile
8   M' = M augmented by all augmentations in ALG
```

Figure 3: The basis part of a 3/4-approximation algorithm.

Theorem 6 *Assume that the lists $arms_x, arms_{x,i}$ and $L_{x,i}$ are given. Then for a given directed center $c = (x, y)$ and a given i if exists one can find in constant time an augmenting path with center c in $\mathcal{A}_2$ that has maximum gain and allows a left extension of rank i that has only one vertex in common with the path.*

Proof. We only have to consider arms in $L_{x,i}$ such that its extension can be combined with some arm in $arms_y$. As the elements in both these lists are sorted, we can find the elements with highest gain in both lists in constant time. A left arm with its extension may intersect a right arm. However, it is easily seen that it is enough to consider the four left and right arms with the highest gain. Then at least one of these pairs is intersection free and allows an extension of the left arm, i.e., it results in a path of length seven or a cycle of length six. As in total there are only constantly many combinations possible, all these can be checked in constant time, and the one with highest gain can be returned. □

We will denote by $A_i(\vec{c})$ an augmentation that results from Theorem 6. In the special case of $i = 0$ we will also consider augmentations consisting only of a left or right arm, i.e., do not have an extension. Also note that the right arm my be degenerated, i.e., it consists of only one edge. In the theorem the precise gain of the rank i extension is not specified. Therefore we will always assume in the following a rank i extension with lowest gain. The algorithm will choose always one extension that has at least this gain.

Figure 3 shows an algorithm based on computing the augmentations $A_i(\vec{c})$. There exist at most $O(m \log_\beta \frac{n}{\alpha})$ different augmentations $A_i(\vec{c})$ each of which can be computed in constant time by Theorem 6. This allows our algorithm to be much faster than listing all elements in $\mathcal{A}_3$. Clearly, the augmentations $A_i(\vec{e})$ do not have to be calculated in each iteration of the while-loop of the algorithm. Instead they are updated in line 6 of the algorithm.

To prove the correctness of the algorithm `3/4-matching` we have to modify

theorems 1 and 2 slightly.

We define a set Ω_3' similarly as the set Ω_3 as a multiset whose elements are from $\mathcal{A}_3$. The only exception are cycles of length six in $M^* \triangle M$. For these we now include three artificial paths of length seven in Ω_3', which result by going around the cycle and taking an edge of M as start and end edge of the path. With this definition of Ω_3' the proof of Theorem 1 can be adopted and we get:

Theorem 7 $\sum_{A \in \Omega_3'} gain(A) \geq 3 \cdot w(M^*) - 4 \cdot w(M)$ □

Similarly the proof of Theorem 2 can be adopted by few changes to analyse the algorithm `3/4-matching`.

Theorem 8 *If the algorithm `3/4-matching` gets a matching M as input then it returns a matching M' such that*

$$w(M') \;\geq\; w(M) + \frac{\beta}{8} \cdot \left(\frac{3}{4} \cdot (1-\alpha) \cdot w(M^*) - w(M) \right) \;.$$

Proof. We essentially follow the proof of Theorem 2. The only point where we have to change the argument is the point where we claimed that for every augmentation S in Ω_3 there exists an augmentation A in ALG with $gain(S) \leq gain(A)$ and A and S are not vertex disjoint. This is no longer true as the chosen A contains only one possible extension of rank i, but not necessarily the best one. Therefore, for $i > 0$ there might be an error of up to a factor β that comes in here (note that by definition all elements with the same rank are at most factor β apart.) For $i > 0$ we have $gain(S) \leq \beta \cdot gain(A)$. For $i = 0$ the error might be arbitrarily large, as the weight of rank 0 elements may be arbitrarily close to 0. For that reason we need some other argument. Consider all rank 0 extensions in $M^* \triangle M$. There are at most n of them and by definition of the rank 0 each have weight at most $\alpha \frac{w_{max}}{n}$. As we can bound w_{max} by $w(M^*)$ the total weight of all rank 0 extensions is at most $\alpha \cdot w(M^*)$. We might think of our algorithm as to operate on the graph where all rank 0 extensions have been removed before. Then the rest of the proof of Theorem 2 remains unchanged and we have to replace $w(M^*)$ by $(1-\alpha) \cdot w(M^*)$. □

We still have to consider the running time of algorithm `3/4-matching`. As mentioned above we do not explicitely recompute the augmentations $A_i(\vec{e})$ in line 3 of the algorithm. Instead we will only update them, if they are affected by line 6 of the algorithm. To see that these updates can be done in total time $O(m \log_\beta \frac{n}{\alpha})$ note that each non-matching edge belongs to at most two arms. We keep pointers for each arm to all the lists it belongs to. This allows to do removal from lists in constant time. As long as the lists

$arms_{v,i}$ contain more than four elements we do not have to recalculate the augmentation $A_i(\vec{e})$ as we do not keep track of the extension that belongs to the augmentation but we only need to know that there exists at least one extension of rank i that can be used. If there are less than four elements in $arms_{v,i}$ each time we remove one element from the list we can recalculate $A_i(\vec{e})$ in constant time. To find the augmentation with maximum gain in line 4 we can use for example fibonacci heaps.

Theorem 9 *For any given $\epsilon > 0$ there exists an $O(m \log n)$ algorithm that finds in a graph a matching of weight at least $(\frac{3}{4} - \epsilon) \cdot w(M^*)$.*

Proof. As argued above the running time of algorithm `3/4-matching` is within the required bounds. Now by applying Theorem 3 with $\alpha > 0$ and $\beta > 1$ chosen sufficiently small (depending on ϵ only) we get the desired result. □

5 A $4/5 - \epsilon$ approximation algorithm

The $O(m \log^2 n)$ algorithm for finding a matching of weight at least $4/5 - \epsilon$ follows essentially the approach of the last section. The only difference here is that instead of considering augmentations $A_i(\vec{e})$ for a center $e \in M$ we have to consider augmentations $A_{i,j}(\vec{e})$ which consist of a left and a right arm and an extension of rank i of the left arm and an extension of rank j of the right arm. We still can use the fact that only a constant number of extensions of the left and the right arm can intersect each other completely. This results in an additional factor $\log_\beta \frac{n}{\alpha}$ for computing $A_{i,j}(\vec{e})$ instead of $A_i(\vec{e})$. The rest of the argumentation follows the line of proof given in the previous section. We therefore get:

Theorem 10 *For any given $\epsilon > 0$ there exists an $O(m \log^2 n)$ algorithm that finds in a graph a matching of weight at least $(\frac{4}{5} - \epsilon) \cdot w(M^*)$.*

6 Conclusion

We have presented two approximation algorithms for the weighted matching problem. The first hast approximation ratio $3/4 - \epsilon$ and running time $O(m \log n)$. The well known and often used greedy algorithm has the same running time, but achieves only an approximation ratio of $1/2$. Our second algorithm has running time $O(m \log^2 n)$ and has an approximation ratio of $4/5 - \epsilon$. It is an interesting open question whether an approximation ratio of $1 - \epsilon$ can be achieved in running time $O(m \text{poly} \log n)$.

References

[1] D.E. Drake, S. Hougardy, Linear Time Local Improvements for Weighted Matchings in Graphs. In: WEA 2003, LNCS 2647, 107-119.

[2] D.E. Drake, S. Hougardy, Improved linear time approximation algorithms for weighted matchings, In: Approximation, Randomization, and Combinatorial Optimization, (Approx/Random) 2003, S.Arora, K.Jansen, J.D.P.Rolim, A.Sahai (Eds.), LNCS 2764, Springer 2003, 14–23.

[3] D.E. Drake, S. Hougardy, A simple approximation algorithm for the weighted matching problem, Information Processing Letters 85:4 (2003), 211–213.

[4] D.E. Drake, S. Hougardy, A linear time approximation algorithm for weighted matchings in graphs, ACM Transactions on Algorithms 1 (2005), 107–122.

[5] J. Edmonds, Maximum matching and a polyhedron with 0, 1-vertices, Journal of Research of the National Bureau of Standards 69B (1965), 125–130.

[6] T. Fischer, A.V. Goldberg, D.J. Haglin, S. Plotkin, Approximating matchings in parallel, Information Processing Letters 46 (1993), 115–118.

[7] H.N. Gabow, Data structures for weighted matching and nearest common ancestors with linking, SODA 1990, 434–443.

[8] M. Goldberg, T. Spencer, A new parallel algorithm for the maximal independent set problem, SIAM Journal on Computing 18:2 (1989), 419–427.

[9] J. Jájá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, Massachusetts 1992.

[10] H.J. Karloff, A Las Vegas RNC algorithm for maximum matching, Combinatorica 6:4 (1986), 387–391.

[11] M. Karpinski, W. Rytter, Fast Parallel Algorithms for Graph Matching Problems, Clarendon Press, Oxford 1998.

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

[13] S. Micali and V.V. Vazirani, An $O(\sqrt{V}E)$ Algorithm for Finding Maximum Matching in General Graphs, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science (1980) 17–27.

[14] K. Mulmuley, U.V. Vazirani, V.V. Vazirani, Matching is as easy as matrix inversion, Combinatorica 7:1 (1987), 105–113.

[15] S. Pettie, P. Sanders, A simpler linear time 2/3- ε approximation for maximum weight matching, Information processing letters 91:6 (2004), 271–276.

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

[17] P.W. Purdom, C.A. Brown, The Analysis of Algorithms, Holt, Rinehart and Winston, New York, 1985.

[18] R. Uehara, Z.-Z. Chen, Parallel approximation algorithms for maximum weighted matching in general graphs, Information Processing Letters 76:1-2 (2000), 13–17.

[19] V.V. Vazirani, A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm, Combinatorica 14:1 (1994), 71–109.