# Fair Matchings and Related Problems*

**Chien-Chung Huang[1], Telikepalli Kavitha[2], Kurt Mehlhorn[3], and Dimitrios Michail[4]**

1 **Chalmers University, Sweden**
huangch@chalmers.se
2 **Tata Institute of Fundamental Research, India**
kavitha@tcs.tifr.res.in
3 **Max-Planck Institut für Informatik, Germany**
mehlhorn@mpi-inf.mpg.de
4 **Harokopio University of Athens, Greece**
michail@hua.gr

## Abstract

Let $G = (A \cup B, E)$ be a bipartite graph, where every vertex ranks its neighbors in an order of preference (with ties allowed) and let $r$ be the worst rank used. A matching $M$ is *fair* in $G$ if it has maximum cardinality, subject to this, $M$ matches the minimum number of vertices to rank $r$ neighbors, subject to that, $M$ matches the minimum number of vertices to rank $(r-1)$ neighbors, and so on. We show an efficient combinatorial algorithm based on LP duality to compute a fair matching in $G$. We also show a scaling based algorithm for the fair *b-matching* problem.

Our two algorithms can be extended to solve other profile-based matching problems. In designing our combinatorial algorithm, we show how to solve a generalized version of the minimum weighted vertex cover problem in bipartite graphs, using a single-source shortest paths computation—this can be of independent interest.

**1998 ACM Subject Classification** F.2.2 Computations on discrete structures

**Keywords and phrases** Matching with Preferences, Fairness and Rank-maximality, Bipartite Vertex Cover, Linear Programming Duality, Complementary Slackness

**Digital Object Identifier** 10.4230/LIPIcs.xxx.yyy.p

## 1 Introduction

Let $G = (A \cup B, E)$ be a bipartite graph on $n$ vertices and $m$ edges, where each $u \in A \cup B$ has a list ranking its neighbors in an order of preference (ties are allowed). Such an instance is usually referred to a *stable marriage* instance with incomplete lists and ties. A matching is a collection of edges, no two of which share an endpoint.

The focus in stable marriage problems is to find matchings that are *stable* [6]. However, there are many applications where stability is not a proper objective: for instance, in matching students with counselors or applicants with training posts, we cannot compromise on the size of the matching and a *fair* matching is a natural candidate for an optimal matching in such problems.

▶ **Definition 1.** A matching $M$ is fair in $G = (A \cup B, E)$ if $M$ has maximum cardinality, subject to this, $M$ matches the minimum number of vertices to rank $r$ neighbors, and subject

* This work is based on two pre-prints [11, 17].


Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to that, $M$ matches the minimum number of vertices to rank $(r-1)$ neighbors, and so on, where $r$ is the worst rank used in the preference lists of vertices.

The fair matching problem can be solved in polynomial time as follows: for an edge $e$ with incident ranks $i$ and $j$, let $w(e) = n^{i-1} + n^{j-1}$. It is easy to see that a maximum cardinality matching of minimum weight (under weight function $w$) is a fair matching in $G$. Such a matching can be computed via the maximum weight matching algorithm by resetting $e$'s weight to $4n^r - n^{i-1} - n^{j-1}$, where $r$ is the largest rank used in any preference list.

However this approach can be expensive even if we use the fastest maximum-weight bipartite matching algorithms [1, 3, 4, 5]. The running time will be $O(rmn)$ or $\tilde{O}(r^2 m\sqrt{n})$. Note that these complexities follow from the customary assumption that an arithmetic operation takes $O(r)$ time on weights of the order $n^r$. We present two different techniques to efficiently compute fair matchings and a generalization called fair *b-matchings*.

**A combinatorial technique.** Our first technique is an iterative combinatorial algorithm for the fair matching problem. The running time of this algorithm is $\tilde{O}(r^* m\sqrt{n})$ or $\tilde{O}(r^* n^\omega)$ with high probability, where $r^*$ is the largest rank used in a fair matching and $\omega \approx 2.37$ is the exponent of matrix multiplication. This algorithm is based on linear programming duality and in each iteration $i$, we solve the following "dual problem" – dual to a variant of the maximum weight matching problem.

> *Generalized minimum weighted vertex cover problem.* Let $G_i = (A \cup B, E_i)$ be a bipartite graph with edge weights given by $w_i : E_i \to \{0, 1, \ldots, c\}$. Let $K_{i-1} \subseteq A \cup B$ satisfy the property that there is a matching in $G$ that matches all $v \in K_{i-1}$. Find a cover $\{y_u^i\}_{u \in A \cup B}$ so that $\sum_{u \in A \cup B} y_u^i$ is minimized subject to (1) for each $e = (a, b)$ in $E_i$, we have $y_a^i + y_b^i \geq w_i(e)$, and (2) $y_u^i \geq 0$ if $u \notin K_{i-1}$.

When $K_{i-1} = \emptyset$, the above problem reduces to the standard weighted vertex cover problem. We show that the generalized minimum weighted vertex cover problem, where $y_v^i$ for $v \in K_{i-1}$ can be negative, can be solved via a single-source shortest paths subroutine in directed graphs, by a non-trivial extension of a technique of Iri [13].

**A scaling technique.** Our second technique uses scaling in order to solve the fair matching problem, by the aforementioned reduction to computing a maximum weight matching using exponentially large edge weights. It starts by solving the problem when each edge weight is 0 and then iteratively solves the problem for better and better approximations of the edge weights. This technique is applicable in the more generalized problem of computing fair *b-matchings*, where each vertex has a capacity associated with it. We solve the fair b-matching problem, in time $\tilde{O}(rmn)$ and space $O(m)$, by solving the capacitated transshipment problem, while carefully maintaining "reduced costs" whose values are within polynomial bounds. Brute-force application of the fastest known minimum-cost flow algorithms would suffer from the additional cost of arithmetic and an $O(rm)$ space requirement. For instance, using [9] would result in $\tilde{O}(r^2 mn)$ running time and $O(rm)$ space.

## 1.1 Background

Fair matchings are a special case of the *profiled-based* matching problems. So far fair matchings have received little attention in the literature. Except the two pre-prints [11, 17] on which this work is based, the only work dealing with fair matchings is the Ph.D. thesis of

Sng [23], where he gives an algorithm to find a fair b-matching[1] in $O(rQ \min\{m \log n, n^2\})$ time, where $Q = \sum_{v \in V} q(v)$, the sum of the capacity $q(v)$ of all vertices $v \in V$.

The first profiled-based matching problem was introduced by Irving [14] and is called "rank-maximal matching" problem.[2] This problem has been well-studied [15, 16, 18, 20].

▶ **Definition 2.** A matching $M$ in $G = (A \cup B, E)$ is rank-maximal if $M$ matches the maximum number of vertices to rank 1 neighbors, subject to this constraint, $M$ matches the maximum number of vertices to rank 2 neighbors, subject to the above two constraints, $M$ matches the maximum number of vertices to rank 3 neighbors, and so on.

However the rank-maximal matching problem has been studied so far in a more restricted model called the the *one-sided* preference lists model. In this model, only vertices of $A$ have preferences over neighbors while vertices in $B$ have no preferences. Note that a problem in the one-sided preference lists model can also be modeled as a problem with two-sided preference lists by making every $b \in B$ assign rank $r$ to every edge incident on it, where $r$ is the worst rank in the preference lists of vertices in $A$.

The current fastest algorithm to compute a rank-maximal matching in the one-sided preference lists model takes time $O(\min\{r^* m\sqrt{n}, mn, r^* n^\omega\})$ [15], where $r^*$ is the largest rank used in a rank-maximal matching. In the one-sided preference lists setting, each edge has a unique rank associated with it, thus the edge set $E$ is partitioned into $E_1 \dot\cup E_2 \dot\cup \cdots \dot\cup E_r$ – this partition enables the problem of computing a rank-maximal matching to be reduced to computing $r^*$ maximum cardinality matchings in certain subgraphs of $G$.

We show here that our fair matching algorithm can be easily modified to compute a rank-maximal matching in the two-sided preference lists model. Thus this problem can be solved in time $\tilde{O}(r^* m\sqrt{n})$ or $\tilde{O}(r^* n^\omega)$ with high probability, which almost matches its running time for the one-sided case. Another problem that our algorithm can solve is the "maximum cardinality" rank-maximal matching problem. A matching $M$ is a *maximum cardinality rank-maximal* matching if $M$ has maximum cardinality, and within the set of maximum cardinality matchings, $M$ is rank-maximal.

*Organization of the paper.* Section 2.1 contains our algorithm for the generalized bipartite vertex cover problem, Section 2.2 has our algorithm for fair matchings. Section 3 has our scaling algorithm. The omitted details can be found in the full version of this paper.

## 2 Our Combinatorial Technique for fair matchings

Recall that our input here is $G = (A \cup B, E)$ and $r$ is the worst or largest rank used in any preference list. The notion of *signature* will be useful to us in designing our algorithm. We first define edge weight functions $w_i$, for $1 \le i \le r-1$. The value $w_i(e)$, where $e = (a, b)$, is defined as follows:

$$w_i(e) = \begin{cases} 2 & \text{if both } a \text{ and } b \text{ rank each other as rank} \le r - i \text{ neighbors} \\ 1 & \text{if exactly one of } \{a, b\} \text{ ranks the other as a rank} \le r - i \text{ neighbor} \\ 0 & \text{otherwise} \end{cases}$$

▶ **Definition 3.** For any matching $M$ in $G$, let $\mathsf{signature}(M)$ be $(|M|, w_1(M), \ldots, w_{r-1}(M))$, where $w_i(M) = \sum_{e \in M} w_i(e)$, for $1 \le i \le r-1$.

[1] Sng used the term "generous maximum matching."
[2] Irving called it "greedy matching."

Thus $\mathsf{signature}(M)$ is an $r$-tuple, where the first coordinate is the size of $M$, the second coordinate is the number of vertices that get matched to neighbors ranked $r-1$ or better, and so on. Let $\mathsf{OPT}$ denote a fair matching. Then $\mathsf{signature}(\mathsf{OPT}) \succeq \mathsf{signature}(M)$ for any matching $M$ in $G$, where $\succeq$ is the lexicographic order on signatures.

In order to capture the first coordinate of $\mathsf{signature}(M)$ also via an edge weight function, let us introduce the function $w_0$ defined as: $w_0(e) = 1$ for all $e \in E$. Thus $|M| = w_0(M) = \sum_{e \in M} w_0(e)$. For any matching $M$ and $0 \leq j \leq r-1$, let $\mathsf{signature}_j(M)$ denote the $(j+1)$-tuple obtained by truncating $\mathsf{signature}(M)$ to its first $j+1$ coordinates.

▶ **Definition 4.** A matching $M$ is $(j+1)$-optimal if $\mathsf{signature}_j(M) = \mathsf{signature}_j(\mathsf{OPT})$.

Our algorithm runs for $r^*$ iterations, where $r^* \leq r$ is the largest index $i$ such that $w_{i-1}(\mathsf{OPT}) > 0$. For any $j \geq 0$, in the $(j+1)$-st iteration, our algorithm solves the minimum weighted vertex cover problem in a subgraph $G_j$. This involves computing a maximum $w_j$-weight matching $M_j$ in the graph $G_j$ under the constraint that all vertices of a *critical* subset $K_{j-1} \subseteq A \cup B$ have to be matched. In the first iteration which corresponds to $j = 0$, we have $G_0 = G$ and $K_{-1} = \emptyset$.

The problem of computing $M_j$ will be referred to as the primal program of the $(j+1)$-st iteration and the minimum weighted vertex cover problem becomes its dual. We will show $M_j$ to be $(j+1)$-optimal. The problem of computing $M_j$ can be expressed as a linear program (rather than an integer program) as the constraint matrix is totally unimodular and hence the corresponding polytope is integral. This linear program and its dual are given below. (Let $\delta(v)$ be the set of edges incident on vertex $v$.)

$$\max \sum_{e \in E} w_j(e) x_e^j$$

$$\sum_{e \in \delta(v)} x_e^j \leq 1 \qquad \forall v \in A \cup B$$

$$\sum_{e \in \delta(v)} x_e^j = 1 \qquad \forall v \in K_{j-1}$$

$$x_e^j \geq 0 \qquad \forall e \text{ in } G_j.$$

$$\min \sum_{v \in V} y_v^j$$

$$y_a^j + y_b^j \geq w_j(e) \qquad \forall e = (a,b) \text{ in } G_j$$

$$y_v^j \geq 0 \qquad \forall v \in (A \cup B) \setminus K_{j-1}.$$

▶ **Lemma 5.** *$M_j$ and $y^j$ are the optimal solutions to the primal and dual programs respectively, iff the following hold:*

1. *if $u$ is unmatched in $M_j$ (thus $u$ has to be outside $K_{j-1}$), then $y_u^j = 0$;*
2. *if $e = (u,v) \in M_j$, then $y_u^j + y_v^j = w_j(e)$;*

Proposition 5 follows from the complementary slackness conditions in the linear programming duality theorem. This suggests the following strategy once the primal and dual optimal solutions $M_j$ and $y^j$ are found in the $(j+1)$-st iteration.

- to prune "irrelevant" edges: if $e = (u,v)$ and $y_u^j + y_v^j > w_j(e)$, then no optimal solution of the $j$-th iteration primal program can contain $e$. So we prune such edges from $G_j$ and let $G_{j+1}$ denote the resulting graph. The graph $G_{j+1}$ will be used in the next iteration.
- to grow the critical set $K_{j-1}$: if $y_u^j > 0$ and $u \notin K_{j-1}$, then $u$ *has to* be matched in every optimal solution of the primal program of the $(j+1)$-st iteration. Hence $u$ should be added to the critical set. Adding such vertices $u$ to $K_{j-1}$ yields the critical set $K_j$ for the next iteration.

Below we first show how to solve the dual problem and then give the main algorithm.

## 2.1 Solving the dual problem

For any $0 \leq j \leq r-1$, let $G_j = (A \cup B, E_j)$ be the subgraph that we work with in the $(j+1)$-st iteration and let $K_{j-1} \subseteq A \cup B$ be the critical set of vertices in this iteration. Recall that for each $e \in E_j$, we have $w_j(e) \in \{0,1,2\}$. We now show how to solve the dual problem efficiently for a more general edge weight function, i.e., $w_j(e) \in \{0,1,\ldots,c\}$ for each $e \in E_j$.

Let $M_j$ be the optimal solution of the primal program (we discuss how to compute it at the end of this section). We know that $M_j$ matches all vertices in $K_{j-1}$. We now describe our algorithm to solve the dual program using $M_j$. Our idea is built upon that of Iri [13], who solved the special case of $K_{j-1} = \emptyset$. Recall that if a vertex $v$ is unmatched in $M_j$, then $v \notin K_{j-1}$.

- Add a new vertex $z$ to $A$ and let $A' = A \cup \{z\}$. Add an edge of weight 0 from $z$ to each vertex in $B \setminus K_{j-1}$. For convenience, we call the edges from $z$ to these vertices "virtual" edges. The matching $M_j$ still remains an optimal feasible solution after this transformation. [Note that there are only $O(n)$ virtual edges.]
- Next direct all edges $e \in E_j \setminus M_j$ from $A'$ to $B$ and set the edge weight $d(e) = -w_j(e)$; also direct all edges in $M_j$ from $B$ to $A'$ and let the edge weight $d(e) = w_j(e)$.
- Create a *source vertex* $s$ and add a directed edge of weight 0 from $s$ to each *unmatched* vertex in $A'$. See Figure 1.

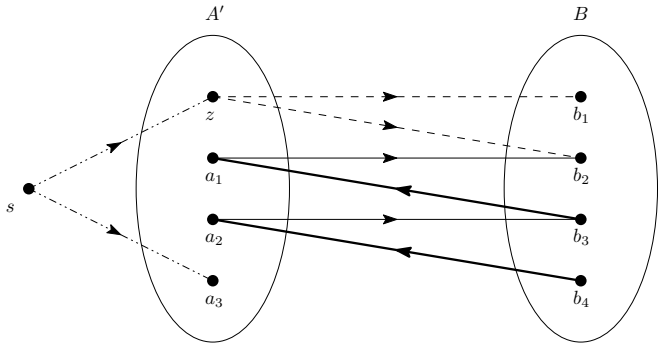


**Figure 1** The bold edges are edges of $M_j$ and are directed from $B$ to $A'$ while the edges of $E_j \setminus M_j$ are directed from $A'$ to $B$.

Let $\mathcal{R}$ denote the set of all vertices in $A' \cup B$ that are reachable from $s$. In Figure 1, $\mathcal{R} = \{z, a_3, b_1, b_2\}$.

▶ **Lemma 6.** *By the above transformation,*

1. $B \setminus K_{j-1} \subseteq \mathcal{R}$.
2. *There is no edge between* $A' \cap \mathcal{R}$ *and* $B \setminus \mathcal{R}$.
3. $M_j$ *projects on to a perfect matching between* $A' \setminus \mathcal{R}$ *and* $B \setminus \mathcal{R}$.

**Proof.** Part (1) holds because there is a directed edge from $s$ to $z$ and directed edges from $z$ to every vertex in $B \setminus K_{j-1}$. To show part (2), it is trivial to see that there can be no edge from $A' \cap \mathcal{R}$ to $B \setminus \mathcal{R}$ (by the definition of $B \setminus \mathcal{R}$). If there is an edge $(b,a)$ from $B \setminus \mathcal{R}$ to $A' \cap R$, then this has to be an edge in $M_j$ and hence it is $a$'s only incoming edge. So for $a$

to be reachable from $s$, it has to be the case that $b$ is reachable from $s$, contradicting that $b \in B \setminus \mathcal{R}$.

For part (3), observe that if $b \in B \setminus \mathcal{R}$ is unmatched in $M_j$, then $b \notin K_{j-1}$ and such a vertex can be reached via $z$, contradicting the assumption that $b \in B \setminus \mathcal{R}$. If $a \in A' \setminus \mathcal{R}$ is unmatched in $M_j$, then such a vertex can be reached from $s$, contradicting the assumption that $a \in A' \setminus \mathcal{R}$. So all vertices in $(A' \cup B) \setminus \mathcal{R}$ are matched in $M_j$. By (2), a vertex $b \in B \setminus \mathcal{R}$ cannot be matched to vertices in $A' \cap \mathcal{R}$. If a vertex $a \in A' \setminus \mathcal{R}$ is matched to a vertex $B$ in $\mathcal{R}$, then $a$ is also in $\mathcal{R}$, a contradiction. This proves part (3). ◀

Note that there may exist some edges in $E_j \setminus M_j$ that are directed from $A' \setminus R$ to $B \cap \mathcal{R}$. Furthermore, some vertices of $A \setminus K_{j-1}$ can be contained in $A \setminus \mathcal{R}$. Delete all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ from $G_j$; let $H_j$ denote the resulting graph. By Lemma 6.3, no edge of $M_j$ has been deleted, thus $M_j$ belongs to $H_j$ and $M_j$ is still an optimal matching in the graph $H_j$. Moreover, $H_j$ is split into two parts: one part is $(A' \cup B) \cap \mathcal{R}$, which is isolated from the second part $(A' \cup B) \setminus \mathcal{R}$. See Figure 2.

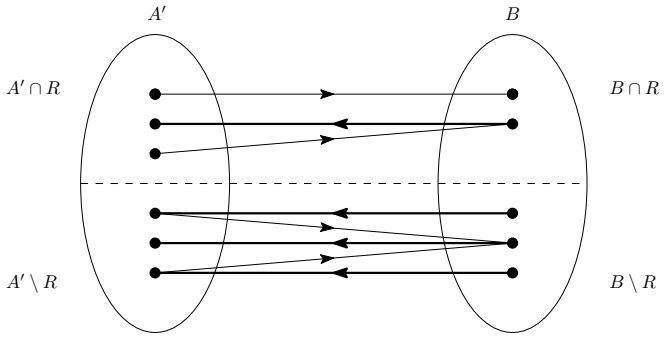


**Figure 2** The set $A' \cup B$ in the graph $H_j$ is split into two parts: $(A' \cup B) \cap \mathcal{R}$ and $(A' \cup B) \setminus \mathcal{R}$

Next add a directed edge from the source vertex $s$ to each vertex in $B \setminus \mathcal{R}$. Each of these edges $e$ has weight $d(e) = 0$. By Lemma 6.3, all vertices can be reached from $s$ now. Also note that there can be no negative-weight cycle, otherwise, we can augment $M_j$ along this cycle to get a matching of larger weight while still keeping the same set of vertices matched, which leads to a contradiction to the optimality of $M_j$.

Apply the single-source shortest paths algorithm [7, 21, 22, 24] from the source vertex $s$ in this graph $H_j$ where edge weights are given by $d(\cdot)$. Such algorithms take $O(m\sqrt{n})$ time or $\tilde{O}(n^\omega)$ time when the largest edge weight is $O(1)$. Let $d_v$ be the distance label of vertex $v \in A' \cup B$.

We define an initial vertex cover as follows. If $a \in A'$, let $\tilde{y}_a := d_a$; if $b \in B$, let $\tilde{y}_b := -d_b$. (We will adjust this cover further later.)

▶ **Lemma 7.** *The constructed initial vertex cover $\{\tilde{y}_v\}_{v \in A' \cup B}$ for the graph $H_j$ satisfies the following properties:*

1. *For each vertex $v \in ((A \cup B) \cap \mathcal{R}) \setminus K_{j-1}$, $\tilde{y}_v \geq 0$.*
2. *If $v \in (A \cup B) \setminus K_{j-1}$ is unmatched in $M_j$, then $\tilde{y}_v = 0$.*
3. *For each edge $e = (a, b) \in H_j$, we have $\tilde{y}_a + \tilde{y}_b \geq w^j_e$.*
4. *For each edge $e = (a, b) \in M_j$, we have $\tilde{y}_a + \tilde{y}_b = w^j_e$.*

**Proof.** For part (1), suppose that $a \in (A \cap \mathcal{R}) \setminus K_{j-1}$ and $\tilde{y}_a < 0$. By Lemma 6.2 and the fact that all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ are absent, the shortest path from $s$ to $a$ cannot go

through $(A \cup B) \setminus \mathcal{R}$. So there exists an alternating path $P$ (of even length) starting from some *unmatched* vertex $a' \in (A' \cap \mathcal{R}) \setminus K_{j-1}$ and ending at $a$. The distance from $a'$ to $a$ along path $P$ must be negative, since $d_a = \tilde{y}_a < 0$. Therefore,

$$\sum_{e \in M_j \cap P} w_e < \sum_{e \in P \setminus M_j} w_e.$$

Note that it is possible that the first edge $e = (a', b) \in P$ is a virtual edge, i.e., $a' = z$ and the first edge $e$ connects $z$ to some vertex $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. In this case, $d_e = 0$ and $b$ is not an element of the critical set $K_{j-1}$. Therefore, irrespective of whether the first edge is virtual or not, we can replace the matching $M_j$ by $M_j \oplus P$ (ignoring the first edge in $P$ if it is virtual), thereby creating a feasible matching with larger weight than $M_j$, a contradiction.

So we are left to worry about the case when vertex $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. Recall that $\tilde{y}_b = -d_b$. We claim that $d_b \leq 0$. Suppose not. Then the shortest distance from $s$ to $b$ is strictly larger than 0. But this cannot be, since there is a path composed of edges $(s, z)$ and $(z, b)$, and such a path has total distance of exactly 0. This completes the proof of part (1).

To show part (2), by Lemma 6.3, an unmatched vertex must be in $\mathcal{R}$. First, assume that this unmatched vertex is $a \in (A \cap \mathcal{R}) \setminus K_{j-1}$. By our construction, there is only one path from $s$ to $a$, which is simply the directed edge from $s$ to $a$ and its distance is 0. So $y_a = d_a = 0$. Next assume that this unmatched vertex is $b \in (B \cap \mathcal{R}) \setminus K_{j-1}$. Suppose that $\tilde{y}_b > 0$. Then $d_b = -\tilde{y}_b < 0$. By Lemma 6.2 and the fact that all edges from $A' \setminus \mathcal{R}$ to $B \cap \mathcal{R}$ have been deleted, the shortest path from $s$ to $b$ cannot go through $(A \cup B) \setminus \mathcal{R}$. So the shortest path from $s$ to $b$ must consist of the edge from $s$ to some unmatched vertex $a \in (A' \cap \mathcal{R}) \setminus K_{j-1}$, followed by an augmenting path $P$ (of odd length) ending at $b$. As in the proof of (1), we can replace $M_j$ by $M_j \oplus P$ (irrespective of whether the first edge in $P$ is virtual or not) so as to get a matching of larger weight while preserving the feasibility of the matching, a contradiction. This proves part (2).

For parts (3) and (4), first consider an edge $e = (a, b)$ outside $M_j$ in $H_j$. Such an edge is directed from $a$ to $b$. So $\tilde{y}_a - w_e^j = d_a + d(e) \geq d_b = -\tilde{y}_b$. This proves part (3). Next consider an edge $e = (a, b) \in M_j$. Such an edge is directed from $b$ to $a$. Furthermore, $e$ is the only incoming edge of $a$, implying that $e$ is part of the shortest path tree rooted at $s$. As a result, $-\tilde{y}_b + w_e^j = d_b + d(e) = d_a = \tilde{y}_a$. This shows part (4). This completes the proof of Lemma 7. ◀

At this point, we possibly still do not have a valid cover for the dual program due to the following two reasons.

- Some vertex $a \in A \setminus K_{j-1}$ has $\tilde{y}_a < 0$. (However it cannot happen that some vertex $b \in B \setminus K_{j-1}$ has $\tilde{y}_b < 0$, since Lemma 6.1 states that such a vertex is in $\mathcal{R}$ and Lemma 7.1 states that $\tilde{y}_b$ must be non-negative.)
- The edges deleted from $G_j$ (to form $H_j$) are not properly covered by the initial vertex cover $\{\tilde{y}_v\}_{v \in A \cup B}$.

We can remedy these two defects as follows. Define $\delta = \max\{\delta_1, \delta_2, 0\}$,

$$\text{where } \delta_1 = \max_{e=(a,b) \in E} \{w_e^j - \tilde{y}_a - \tilde{y}_b\} \quad \text{and} \quad \delta_2 = \max_{a \in A \setminus K_{j-1}} \{-\tilde{y}_a\}.$$

In $O(n+m)$ time, we can compute $\delta$. If $\delta = 0$, the initial cover is already a valid solution to the dual program. In the following, we assume that $\delta > 0$ exists (if the initial cover is already a valid solution for the dual program, then the proof that it is also optimal is just the same as in Theorem 8.) We build the final vertex cover as follows.

1. For each vertex $u \in (A \cup B) \cap \mathcal{R}$, let $y_u = \tilde{y}_u$;
2. For each vertex $a \in A \setminus \mathcal{R}$, let $y_a = \tilde{y}_a + \delta$;
3. For each vertex $b \in B \setminus \mathcal{R}$, let $y_b = \tilde{y}_b - \delta$;

▶ **Theorem 8.** *The final vertex cover $\{y_v\}_{v \in A \cup B}$ is an optimal solution for the dual program.*

Given $M_j$, it follows that the dual problem can be solved in time $O(m\sqrt{n})$ or $\tilde{O}(n^\omega)$. The problem of computing $M_j$ can be solved by the following folklore technique: form a new graph $\tilde{G}_j$ by taking two copies of $G_j$ and making the two copies of a vertex $u \notin K_{j-1}$ adjacent using an edge of weight 0. A maximum weight *perfect* matching in $\tilde{G}_j$ yields a maximum weight matching in $G_j$ that matches all vertices in $K_{j-1}$, i.e., an optimal solution to the primal program of the $j$-th iteration. Since $c = O(1)$, a maximum weight perfect matching in $\tilde{G}_j$ can be found in $O(m\sqrt{n}\log n)$ time by the fastest bipartite matching algorithms [1, 3, 5], or in $\tilde{O}(n^\omega)$ time with high probability by Sankowski's algorithm [22].

## 2.2 Our main algorithm

We now present our algorithm to compute a fair matching. Recall that $r$ is the worst rank in the problem instance and $r^*$ is the worst rank in a fair matching. We first present an algorithm that runs for $r$ iterations and we show later in this section how to terminate our algorithm in $r^*$ iterations.

1. *Initialization.* Let $G_0 = G$ and $K_{-1} = \emptyset$.
2. For $j = 0$ to $r - 1$ do
   a. Find the optimal solution $\{y^j_u\}_{u \in A \cup B}$ to the dual program of the $(j+1)$-st iteration.
   b. Delete from $G_j$ every edge $(a, b)$ such that $y^j_a + y^j_b > w_j(e)$. Call this subgraph $G_{j+1}$.
   c. Add all vertices with positive dual values to the critical set, i.e., $K_j = K_{j-1} \cup \{u\}_{y^j_u > 0}$.
3. Return the optimal solution to the primal program of the last iteration.

The solution returned by our algorithm is a maximum $(w_{r-1})$-weight matching in the graph $G_{r-1}$ that matches all vertices in $K_{r-2}$. By Proposition 5, this is, in fact, a matching in the subgraph $G_r$ that matches all vertices in $K_{r-1}$. Lemma 10 proves the correctness of our algorithm. Lemma 9 guarantees that our algorithm is never "stuck" in any iteration due to the infeasibility of the primal or dual problem.

▶ **Lemma 9.** *The primal and dual programs of the $(j+1)$-st iteration are feasible, for $0 \le j \le r - 1$.*

▶ **Lemma 10.** *For every $0 \le j \le r - 1$, the following hold:*

1. *any matching $M$ in $G_j$ that matches all $v \in K_{j-1}$ is $j$-optimal;*
2. *conversely, a $j$-optimal matching in $G$ is a matching in $G_j$ that matches all $v \in K_{j-1}$.*

**Proof.** We proceed by induction. The base case is $j = 0$. As $K_{-1} = \emptyset$, $G_0 = G$, and all matchings are, by definition, 0-optimal, the lemma holds vacuously.

For the induction step $j \ge 1$, suppose that the lemma holds up to $j-1$. As $K_{j-1} \supseteq K_{j-2}$ and $G_j$ is a subgraph of $G_{j-1}$, $M$ is a matching in $G_{j-1}$ that matches all vertices of $K_{j-2}$. Thus by induction hypothesis, $M$ is $(j-1)$-optimal. For each edge $e = (a, b) \in M$ to be present in $G_j$, $e$ must be a tight edge in the $j$-th iteration, i.e., $y^{j-1}_a + y^{j-1}_b = w_{j-1}(e)$. Furthermore, as $K_{j-1} \supseteq \{u\}_{y^{j-1}_u > 0}$, we have

$$w_{j-1}(M) = \sum_{e=(a,b)\in M} w_{j-1}(e) = \sum_{e=(a,b)\in M} y_a^{j-1} + y_b^{j-1} \geq \sum_{u\in A\cup B} y_u^{j-1},$$

where the final inequality holds because all vertices $v$ with positive $y_v^{j-1}$ are matched in $M$. By linear programming duality, $M$ must be optimal in the primal program of the $j$-th iteration. So the $j$-th primal program has optimal solution of value $w_{j-1}(M)$.

Recall that by definition, $\mathsf{OPT}$ is also $(j-1)$-optimal. By (2) of the induction hypothesis, $\mathsf{OPT}$ is a matching in $G_{j-1}$ and $\mathsf{OPT}$ matches all vertices in $K_{j-2}$. So $\mathsf{OPT}$ is a feasible solution of the primal program in the $j$-th iteration. Thus $w_{j-1}(\mathsf{OPT}) \leq w_{j-1}(M)$. However, it cannot happen that $w_{j-1}(\mathsf{OPT}) < w_{j-1}(M)$, otherwise, $\mathsf{signature}(M) \succ \mathsf{signature}(\mathsf{OPT})$, since both $\mathsf{OPT}$ and $M$ have the same first $j-1$ coordinates in their signatures. So we conclude that $w_{j-1}(\mathsf{OPT}) = w_{j-1}(M)$, and this implies that $M$ is $j$-optimal as well. This proves (1).

In order to show (2), let $M'$ be a $j$-optimal matching in $G$. Since $M'$ is $j$-optimal, it is also $(j-1)$-optimal and by (2) of the induction hypothesis, it is a matching in $G_{j-1}$ that matches all vertices in $K_{j-2}$. So $M'$ is a feasible solution to the primal program of the $j$-th iteration. As $\mathsf{signature}(M')$ has $w_{j-1}(\mathsf{OPT})$ in its $j$-th coordinate, $M'$ must be an optimal solution to this primal program; otherwise there is a $j$-optimal matching with a value larger than $w_{j-1}(\mathsf{OPT})$ in the $j$-th coordinate of its signature, contradicting the optimality of $\mathsf{OPT}$. By Proposition 5.2, all edges of $M'$ are present in $G_j$ and by Proposition 5.1, all vertices $u \notin K_{j-2}$ with $y_u^{j-1} > 0$, in other words, all vertices in $K_{j-1} \setminus K_{j-2}$ have to be matched by the optimal solution $M'$. This completes the proof of (2). ◀

Since our algorithm returns a matching in $G_r$ that matches all vertices in $K_{r-1}$, we know from Lemma 10.1 that this matching is $r$-optimal, thus the matching returned is fair. As mentioned earlier, our algorithm can be modified so that it terminates in $r^*$ iterations. For that, we need to know the value of $r^*$.

We continue to use the weight function $w_0 : E \to \{1\}$, however instead of $w_1, \ldots, w_r$, we should use the weight functions $\tilde{w}_1, \ldots, \tilde{w}_{r^*-1}$ where for $1 \leq i \leq r^* - 1$, $\tilde{w}_i$ is defined as: for any edge $e = (a, b)$, $\tilde{w}_i(e)$ is 2 if both $a$ and $b$ rank each other as rank $\leq r^* - i + 1$ neighbors, it is 1 if exactly one of $\{a, b\}$ ranks the other as a rank $\leq r^* - i + 1$ neighbor, otherwise it is 0. The value $r^*$ can be easily computed right at the start of our algorithm as follows.

- Let $M^*$ be a maximum cardinality matching in $G$. The value $r^*$ is the smallest index $j$ such that the subgraph $\bar{G}_j$ admits a matching of size $|M^*|$, where $\bar{G}_j$ is obtained by deleting all edges $e = (a, b)$ from $G$ where either $a$ or $b$ (or both) ranks the other as a rank $> j$ neighbor.
- We compute $r^*$ by first computing $M^*$ and then computing a maximum cardinality matching in $\bar{G}_1, \bar{G}_2, \ldots$ and so on till we see a subgraph $\bar{G}_j$ that admits a matching of size $|M^*|$. This index $j = r^*$ and it can be found in $O(r^* m\sqrt{n})$ time [12] or in $O(r^* n^\omega)$ time [10, 19].

We now bound the running time of our algorithm. We showed how to solve the dual program in $O(m\sqrt{n})$ time once we have the solution to the primal program and we have seen that the primal program can be solved in $O(m\sqrt{n}\log n)$ time. Alternatively, both the primal and dual problems can be solved in $\tilde{O}(n^\omega)$ time with high probability. Theorem 11 follows.

▶ **Theorem 11.** *A fair matching $M$ in $G = (A \cup B, E)$ can be computed in $\tilde{O}(r^* m\sqrt{n})$ time, or in $\tilde{O}(r^* n^\omega)$ time with high probability, where $r^*$ is the largest rank incident on an edge in $M$, $n = |A \cup B|$, $m = |E|$, and $\omega \approx 2.37$ is the exponent of matrix multiplication.*

In the full version, we show how our algorithm can be adapted to find a rank-maximal and a maximum cardinality rank-maximal matching.

▶ **Theorem 12.** *A rank-maximal/maximum cardinality rank-maximal in $G = (A \cup B, E)$ with two-sided preference lists, can be computed in $\tilde{O}(r^* m\sqrt{n})$ time, or in $\tilde{O}(r^* n^\omega)$ time with high probability, where $r^*$ is the largest rank used in such a matching.*

## 3 The fair b-matching problem: our scaling technique

The fair matching problem can be generalized by introducing capacities on the vertices. We are given $G = (A \cup B, E)$ as before, along with the capacity function $q : V \to \mathbb{Z}_{>0}$. What we seek is a subset $E'$ of $E$ where each vertex $v \in A \cup B$ is incident to at most $q(v)$ edges in $E'$. Such a subset $E'$ is a *b-matching*. Our goal here is to find a fair b-matching, i.e., a b-matching $M$ which has the largest possible size, subject to this constraint, $M$ matches the minimum number of vertices to their rank $r$ neighbors, and so on.

The fair b-matching problem can be reduced to the minimum-cost flow problem as follows. Add two additional vertices $s$ and $t$. For each vertex $a \in A$, add an edge $(s, a)$ with capacity $q(a)$ and cost zero; for each vertex $b \in B$, add an edge $(b, t)$ with capacity $q(b)$ and cost zero. Every edge $(a, b)$ where $a \in A, b \in B$ has capacity one and is directed from $A$ to $B$. If the incident ranks on edge $e$ are $i$ and $j$, then $e$ will be assigned a cost of $-(4n^r - n^{i-1} - n^{j-1})$. The resulting instance has a trivial upper bound of $n^2/4$ on the maximum $s$-$t$ flow. We also add an edge from $t$ to $s$ with zero cost and capacity larger than the $n^2/4$ upper bound. It is easy to verify that a minimum-cost circulation yields a fair b-matching.

We note however, that the above reduction involves costs that are exponential in the size of the original problem. We now present a general technique in order to handle these huge costs – we focus on solving the *capacitated transshipment* version of the minimum-cost flow problem [8]. Let $G = (V, E)$ be a directed network with a cost $c : E \to \mathbb{Z}$ and capacity $u : E \to \mathbb{Z}_{\geq 0}$ associated with each edge. With each $v \in V$ a real number $b(v)$ is associated, where $\sum_{v \in V} b(v) = 0$. If $b(v) > 0$, then $v$ is a supply node, and if $b(v) < 0$, then $v$ is a demand node. We assume $G$ to be *symmetric*, i.e., $e \in E$ implies that the reverse arc $e^R \in E$. The reversed edges are added in the initialization step. The cost and capacity functions satisfy $c(e) = -c(e^R)$ for each $e \in E$, $u(e) \geq 0$ for the original edges and $u(e^R) = 0$ for the additional edges. From now on, $E$ denotes the set of original and artificial edges.

A *pseudoflow* is a function $x : E \to \mathbb{Z}$ satisfying the *capacity* and *antisymmetry* constraints: for each $e \in E$, $x(e) \leq u(e)$ and $x(e) = -x(e^R)$. This implies $x(e) \geq 0$ for the original edges. For a pseudoflow $x$ and a node $v$, the *imbalance* $imb_x(v)$ is defined as $imb_x(v) = \sum_{(w,v) \in E} x(w, v) + b(v)$. A flow is a pseudoflow $x$ such that, $imb_x(v) = 0$ for all $v \in V$. The cost of a pseudoflow $x$ is $cost(x) = \sum_{e \in E} c(e)x(e)$. The minimum-cost flow problem asks for a flow of minimum cost.

For a given flow $x$, the residual capacity of $e \in E$ is $u_x(e) = u(e) - x(e)$. The residual graph $G(x) = (V, E(x))$ is the graph induced by edges with positive residual capacity. A *potential* function is a function $\pi : V \to \mathbb{Z}$. For a potential function $\pi$, the *reduced cost* of an edge $e = (v, w)$ is $c^\pi(v, w) = c(v, w) + \pi(v) - \pi(w)$. A flow $x$ is optimal if and only if there exists a potential function $\pi$ such that $c^\pi(e) \geq 0$ for all residual graph edges $e \in E(x)$. For a constant $\varepsilon \geq 0$ a flow is $\varepsilon$-optimal if $c^\pi(e) \geq -\varepsilon$ for all $e \in E(x)$ for some potential function $\pi$. Consider an $\varepsilon$-optimal flow $x$ and any original edge $e$. If $c^\pi(e) < -\varepsilon$, the residual capacity of $e$ must be zero and hence $e$ is saturated, i.e., $x(e) = u(e)$. If $c^\pi(e) > \varepsilon$, we have $c^\pi(e^R) = -c^\pi(e) < -\varepsilon$ and hence the residual capacity of $e^R$ must be zero. Thus $e^R$ is saturated, i.e., $x(e^R) = u(e^R) = 0$. So $e$ is unused.

We are now ready to describe our scaling algorithm, which is presented in a concise form in Figure 3. The details can be found in the full version. We conclude this section with Theorem 13, which follows from the edge cost values used in our reduction.

1. *Reduction.*
   - **a.** Add two additional vertices $s$ and $t$. For each vertex $a \in A$, add an edge $(s, a)$ with capacity $q(a)$ and cost zero; for each vertex $b \in B$, add an edge $(b, t)$ with capacity $q(b)$ and cost zero. Add an edge from $t$ to $s$ with zero cost and capacity larger than $n^2/4$.
   - **b.** Direct any edge $(a, b)$ where $a \in A$ and $b \in B$ from $A$ to $B$, set its capacity to one and cost to $-(4n^r - n^{i-1} - n^{j-1})$.
   - **c.** Set the demand/supply values of all vertices to zero. Add, if required, additional edges to ensure that $G$ is symmetric.
2. *Initialization Phase.*
   - **a.** Multiply all edge costs by $2^{1+\lceil \log n \rceil}$ to make them divisible by the same amount.
   - **b.** Let $K = \lceil \log C \rceil$ where $C$ is the magnitude of the largest edge cost and let $\mathcal{E}_i$, $1 \leq i \leq K$ denote the set of all edges having a 1 in the $i$-th bit of their cost.
   - **c.** Initialize $x_0$ to any feasible flow and reduced cost $c_0(e) = 0$ for any $e \in E$.
3. *Scaling Phase.* For $i = 1$ to $K$ do
   - **a.** Let $\tilde{c}_i(e) = 2c_{i-1}(e) + (1 \text{ if } e \in \mathcal{E}_i \text{ else } 0) \times sign(e)$, where $sign(e) = \pm 1$ depending on the sign of the original cost $c(e)$. The flow $x_{i-1}$ is 3-optimal with respect to the cost function $\tilde{c}_i$ and the zero potential function, i.e., the potential of all the vertices is 0.
   - **b.** Use the results of [9] with input (i) the flow $x_{i-1}$, (ii) $\tilde{c}_i$ as the edge cost function and (iii) the zero potential function, to compute a 1-optimal flow and a potential function $\tilde{\pi}$ which proves the 1-optimality. Let $x_i$ be this flow.
     Potentials are only decreased, starting from zero, during the computation and $\tilde{\pi}(v) \geq -d \cdot n$ for some constant $d$ and all $v$. Constant $d$ depends on the way the techniques of [9] are applied to refine a 3-optimal flow to an 1-optimal flow.
   - **c.** Compute new reduced costs as $c_i(u, v) = \tilde{c}_i(u, v) + \tilde{\pi}(u) - \tilde{\pi}(v)$.
   - **d.** If any edge $e \in E$ has $|c_i(e)| > d \cdot n + 1$ where $d$ is the constant from step 3b fix it to empty or saturated by removing it (and its reversal) from the graph and modifying the imbalances of both its endpoints accordingly.
4. Return the b-matching induced by the flow $x_K$ and any flow on edges which were fixed to either empty or saturated.

**Figure 3** The scaling algorithm for the fair b-matching problem.

▶ **Theorem 13.** *Given $G = (A \cup B, E)$ and a capacity function $q : A \cup B \to \mathbb{Z}_{>0}$, the fair b-matching problem can be solved in time $O(rmn \log(n^2/m) \log n)$ using space $O(m)$.*

## Acknowledgements

We are grateful to the anonymous reviewers for their careful comments. Special thanks to the reviewer who pointed out Sng's thesis [23].

## References

1 J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. In *Mathematical Programming* 54(1): 41-56, 1992.

2 R. K. Ahuja, T. L. Magnanti and J. B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993).

3 R. Duan and H.-H. Su. A Scaling Algorithm for Maximum Weight Matchings in Bipartite Graphs. In 23rd *SODA*: 1413-1424, 2012.

4 M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *J.ACM* 34(3), 596-615, 1987.

5 H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. In *SIAM J. Comput.* 18: 1013-1036, 1989.

6 D. Gale and L.S. Shapley. College admissions and the stability of marriage. In *American Mathematical Monthly* 69: 9-15, 1962.

7 A. V. Goldberg. Scaling Algorithms for the Shortest Paths Problem. In *SIAM J. Comput.* 24(3): 494-504, 1995.

8 A. V. Goldberg, E. Tardos and R. E. Tarjan. Network Flow Algorithms. In *Paths, Flows and VLSI-Design*: 101-164, Springer Verlag, 1990.

9 A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. In *Math. Oper. Res.* 15: 430-466, 1990.

10 N. J. A. Harvey. Algebraic Structures and Algorithms for Matching and Matroid Problems. In *SIAM J. Comput.* 39(2): 679-702, 2009.

11 C.-C. Huang and T. Kavitha. Weight-maximal Matchings. In the 2nd International Workshop on Matching under Preferences, July 2012.

12 J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *SIAM J. Comput.* 2: 225-231, 1973.

13 M. Iri. A new method of solving transportation-network problems. In *Journal of the Operations Research Society of Japan* 3: 27-87, 1960.

14 R. W. Irving. Greedy Matchings. University of Glasgow, Computing Science Department Research Report, TR-2003-136, 2003.

15 R.W. Irving, T. Kavitha, K. Mehlhorn, D. Michail and K. E. Paluch. Rank-maximal matchings. In *ACM Transactions on Algorithms* 2(4): 602-610, 2006.

16 T. Kavitha and C. D. Shah. Efficient Algorithms for Weighted Rank-Maximal Matchings and Related Problems. In *17th ISAAC*: 153-162, 2006.

17 K. Mehlhorn and D. Michail. Network Problems with Non-Polynomial Weights and Applications. Available at `www.mpi-sb.mpg.de/~mehlhorn/ftp/HugeWeights.ps`

18 D. Michail. Reducing rank-maximal to maximum weight matching. In *Theoretical Computer Science* 389(1-2): 125-132, 2007.

19 M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *45th FOCS*: 248-255, 2004.

20 K. Paluch. Capacitated Rank-Maximal Matchings. In *8th CIAC* 410: 324-335, 2013.

21 P. Sankowski. Shortest Paths in Matrix Multiplication Time. In *13th ESA*: 770-778, 2005.

22 P. Sankowski. Maximum weight bipartite matching in matrix multiplication Time. In *Theoretical Computer Science* 410: 4480-4488, 2009.

23 C. Sng. Efficient Algorithms for bipartite matching problems with preferences Ph.D. thesis, University of Glasgow, 2008.

24 R. Yuster and U. Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *46th FOCS*: 90-100, 2005.