Fair Matchings and Related Problems

Chien-Chung Huang¹ · Telikepalli Kavitha² · Kurt Mehlhorn³ · Dimitrios Michail⁴

Received: 21 November 2013 / Accepted: 24 March 2015 / Published online: 8 April 2015
© Springer Science+Business Media New York 2015

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.

**Keywords**  Matching under preferences · Profile-based matching · Bipartite graphs

1 Introduction

Let $G = (A \cup B, E)$ be a bipartite graph on $n$ vertices and $m$ edges, where each vertex $u \in A \cup B$ has a list ranking its neighbors in an order of preference (ties are allowed here). Such an instance is usually referred to as 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* [3]. 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 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 [4–7]. The running time will be $O(rmn)$ or $\tilde{O}(r^2m\sqrt{n})$. Note that these complexities follow from the customary assumption that an arithmetic operation takes $O(r)$ time on weights of order $nr^\omega$. 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 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

---

1 We note that one can easily compute the value of $r^*$ in $\tilde{O}(r^*m\sqrt{n})$ or in $\tilde{O}(r^*n^\omega)$ time with high probability (see Sect. 2.2 for details). Thus the brute-force reduction to weighted matching can be improved to have running times of $O(r^*mn)$ or $\tilde{O}(r^*^2m\sqrt{n})$. 

---

\(\text{Springer}\)
$K_i \subseteq A \cup B$ satisfy the property that there is a matching in $G$ that matches all $v \in K_i$. Find a cover \{ $y^i_u$ $|$ $u \in A \cup B$ \} so that $\sum_{u \in A \cup B} y^i_u$ is minimized subject to (1) for each $e = (a, b)$ in $E_i$, we have $y^i_a + y^i_b \geq w_i(e)$, and (2) $y^i_u \geq 0$ if $u \notin K_i$.

When $K_i = \emptyset$, the above problem reduces to the standard weighted vertex cover problem. We show that the generalized minimum weighted vertex cover problem, where $y^i_v$ for $v \in K_i$ 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 [8].

**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 general 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. Fair matchings have not received much attention in the literature so far. Except the two preprints [1,2] on which this work is based, the only work dealing with fair matchings is the Ph.D. thesis of Sng [10], which has an algorithm to find a fair $b$-matching\(^2\) in a bipartite graph $G = ( A \cup B, E )$ where each vertex $b \in B$ has capacity $q(b)$ while each vertex $a \in A$ has capacity 1. The running time of Sng’s algorithm is $O(r \cdot Q \cdot \min \{ m \log n, n^2 \})$, where \( Q = \sum_{b \in B} q(b) \).

The first profiled-based matching problem was introduced by Irving [11] and is called the “rank-maximal matching” problem.\(^3\) This problem has been well-studied [12–15].

**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 the more restricted model of one-sided preference lists. In this model, only vertices of $A$ have

\(^2\) Sng used the term “generous maximum matching”.

\(^3\) Irving originally called it the “greedy matching” problem.
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 $O\left(\min\{r^*m\sqrt{n}, mn, r^*n^\omega\}\right)$ [12], 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 $E_1 \cup E_2 \cup \cdots \cup E_r$—this partition enables the problem of computing a rank-maximal matching to be reduced to computing $r^*$ maximum cardinality matchings in certain subgraphs of $G$.

Our fair matching algorithm can be easily modified to compute a rank-maximal matching in the two-sided preference lists model. Thus the latter problem can be solved in time $\tilde{O}(r^*m\sqrt{n})$ or in time $\tilde{O}(r^*n^\omega)$ 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 contains our combinatorial algorithm for fair matchings along with the algorithm for solving the generalized minimum weighted vertex cover problem. Section 3 gives the scaling algorithm for fair b-matchings.

2 The Fair Matching Problem: Our Combinatorial Technique

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 $1 \leq i \leq r$.

For $i \in \{1, \ldots, r\}$, the value $w_i(e)$, where $e = (a, b)$, is defined as follows:

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

**Definition 3** For any matching $M$ in $G$, let $\text{signature}(M) = (w_1(M), w_2(M), \ldots, w_r(M))$, where $w_i(M) = \sum_{e \in M} w_i(e)$, for $1 \leq i \leq r$.

Thus $\text{signature}(M)$ is an $r$-tuple, where the first coordinate is twice the size of $M$ (as $r$ is the largest rank used in the input instance, $\sum_{e \in M} w_1(e) = 2|M|$), the second coordinate is the number of vertices that get matched to neighbors ranked $r - 1$ or better, and so on. Let OPT denote a fair matching. Then $\text{signature}(\text{OPT}) \succeq \text{signature}(M)$ for any matching $M$ in $G$, where $\succeq$ is the lexicographic order on signatures.

For any matching $M$ and $1 \leq j \leq r$, let $\text{signature}_j(M) = (w_1(M), \ldots, w_j(M))$ denote the $j$-tuple obtained by truncating $\text{signature}(M)$ to its first $j$ coordinates.

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

\[ \odot \text{ Springer] } \]
For convenience, all matchings are by default 0-optimal. A fair matching is the same as an \(r\)-optimal matching, thus what we seek is an \(r\)-optimal matching \(M\).

We first present an algorithm that runs for \(r\) iterations and computes an \(r\)-optimal matching. For any \(j \geq 1\), the \(j\)-th iteration of this algorithm computes a \(j\)-optimal matching \(M_j\). This involves solving the minimum weighted vertex cover problem in a subgraph \(G_j\) (which is to be defined later), where the edge weights are given by the weight function \(w_j\), and computing a maximum weight matching \(M_j\) in the graph \(G_j\), under the constraint that all vertices of a critical subset \(K_j \subseteq A \cup B\) have to be matched. In the first iteration which corresponds to \(j = 1\), we have \(G_1 = G\) and \(K_1 = \emptyset\).

We will show that each iteration of this algorithm runs in time \(O(m \sqrt{n} \log n)\) or in time \(\tilde{O}(n^\omega)\) with high probability. Thus we have an algorithm for computing a fair matching that runs in time \(O(rm \sqrt{n} \log n)\) or \(\tilde{O}(rn^\omega)\) with high probability. We then show how to improve this algorithm so that the modified algorithm runs for only \(r^*\) iterations, where \(r^* \leq r\) is the largest rank used in a fair matching.

We will now focus on the first version of our algorithm that runs for \(r\) iterations. The problem of computing the matching \(M_j\) will be referred to as the primal program of the \(j\)-th iteration and the minimum weighted vertex cover problem becomes its dual. 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 \(v\).)

\[
\begin{align*}
\max & \sum_{e \in E} w_j(e)x^j_e \\
\text{s.t.} & \sum_{e \in \delta(v)} x^j_e \leq 1 & \forall v \in A \cup B \\
& \sum_{e \in \delta(v)} x^j_e = 1 & \forall v \in K_j \\
& x^j_e \geq 0 & \forall e \in G_j.
\end{align*}
\]

\[
\begin{align*}
\min & \sum_{v \in V} y^j_v \\
\text{s.t.} & y^j_a + y^j_b \geq w_j(e) & \forall e = (a, b) \text{ in } G_j \\
& y^j_v \geq 0 & \forall v \in (A \cup B) \setminus K_j.
\end{align*}
\]

**Proposition 1** \(M_j\) and \(y^j\) are the optimal solutions to the primal and dual programs respectively of the \(j\)-th iteration, iff the following conditions hold:

1. if \(u\) is unmatched in \(M_j\) (thus \(u\) has to be outside \(K_j\)), then \(y^j_u = 0\);
2. if \(e = (u, v) \in M_j\), then \(y^j_u + y^j_v = w_j(e)\).

**Proposition 1** 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 \(j\)-th iteration.

- to prune “irrelevant” edges: if \(e = (u, v)\) and \(y^j_u + y^j_v > w_j(e)\), then no optimal solution to the primal program of the \(j\)-th iteration 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 \( K_j \): if \( y_u^j > 0 \) and \( u \not\in K_j \), then \( u \) has to be matched in every optimal solution of the primal program of the \( j \)-th iteration. Hence \( u \) should be added to the critical set. Adding such vertices \( u \) to \( K_j \) yields the critical set \( K_{j+1} \) 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 \( 1 \leq j \leq r \), let \( G_j = (A \cup B, E_j) \) be the subgraph that we work with in the \( j \)-th iteration and let \( K_j \subseteq A \cup B \) 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 \( K_j \). We now describe our algorithm to solve the dual program using \( M_j \). Our idea is built upon that of Iri [8], who solved the special case of \( K_j = \emptyset \). Recall that if a vertex \( v \) is not matched in \( M_j \), then \( v \not\in K_j \).

- 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 \( B \setminus K_j \). 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 Fig. 1.

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

![Fig. 1](image_url) 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 \( B \).
Lemma 1 By the above transformation,
1. $B \setminus K_j \subseteq R$.
2. There is no edge between $A' \cap R$ and $B \setminus R$.
3. $M_j$ projects on to a perfect matching between $A' \setminus R$ and $B \setminus R$.

Proof Part (1) holds because there is a directed edge from $s$ to $z$ and directed edges from $z$ to every vertex in $B \setminus K_j$. To show part (2), it is trivial to see that there can be no edge from $A' \cap R$ to $B \setminus R$ (by the definition of $B \setminus R$). If there is an edge $(b, a)$ from $B \setminus R$ 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 $b \in B \setminus R$.

For part (3), observe that if $b \in B \setminus R$, then by (1), $b \in K_j$ and this implies that $b$ is matched in $M_j$. If $a \in A' \setminus R$ is unmatched in $M_j$, then such a vertex can be reached from $s$, contradicting the assumption that $a \in A' \setminus R$. So all vertices in $(A' \cup B) \setminus R$ are matched in $M_j$. By (2), a vertex $b \in B \setminus R$ cannot be matched to vertices in $A' \cap R$. If a vertex $a \in A' \setminus R$ is matched to a vertex $B$ in $R$, then $a$ is also in $R$, a contradiction. This proves part (3). $\square$

Note that there may exist some edges in $E_j \setminus M_j$ that are directed from $A' \setminus R$ to $B \cap R$. Furthermore, some vertices of $A \setminus K_j$ can be contained in $A \setminus R$. Delete all edges from $A' \setminus R$ to $B \cap R$ from $G_j$; let $H_j$ denote the resulting graph. By Lemma 1.3, 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 $(A' \cup B) \setminus R$, which is isolated from the second part $(A' \cup B) \setminus R$. See Fig. 2.

Next add a directed edge from the source vertex $s$ to each vertex in $B \setminus R$. Each of these edges $e$ has weight $d(e) = 0$. By Lemma 1.3, 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 [16–19] 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 $v \in A' \cup B$. 

Fig. 2 The set $A' \cup B$ in the graph $H_j$ is split into two parts: $(A' \cup B) \cap R$ and $(A' \cup B) \setminus R$. 

\[ \begin{array}{c}
A' \\
A' \cap R
\end{array} \quad \begin{array}{c}
A' \setminus R
\end{array} \quad \begin{array}{c}
B
\end{array} \quad \begin{array}{c}
B \cap R
\end{array} \quad \begin{array}{c}
B \setminus R
\end{array} \]

\[ \begin{array}{c}
A' \\
A' \cap R
\end{array} \quad \begin{array}{c}
A' \setminus R
\end{array} \quad \begin{array}{c}
B
\end{array} \quad \begin{array}{c}
B \cap R
\end{array} \quad \begin{array}{c}
B \setminus R
\end{array} \]
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 2** 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 \( v \in ((A \cup B) \cap \mathcal{R}) \setminus K_j, \) \( \tilde{y}_v \geq 0. \)
2. If \( v \in (A \cup B) \setminus K_j \) is unmatched in \( M_j \), then \( \tilde{y}_v = 0. \)
3. For each edge \( e = (a, b) \in M_j, \) we have \( \tilde{y}_a + \tilde{y}_b = w_j. \)
4. For each edge \( e = (a, b) \in H_j, \) we have \( \tilde{y}_a + \tilde{y}_b \geq w_e. \)

**Proof** For part (1), suppose that \( a \in (A \cap \mathcal{R}) \setminus K_j \) and \( \tilde{y}_a < 0. \) By Lemma 1.2 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 \( a' \in (A' \cap \mathcal{R}) \setminus K_j \) 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 \( b \in (B \cap \mathcal{R}) \setminus K_j. \) In this case, \( d_e = 0 \) and \( b \) is not an element of the critical set \( K_j. \) 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 \( b \in (B \cap \mathcal{R}) \setminus K_j. \) Recall that \( \tilde{y}_b = -d_b. \) We claim that \( d_b \leq 0. \) 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 weight exactly 0. This completes the proof of part (1).

To show part (2), by Lemma 1.3, an unmatched vertex must be in \( \mathcal{R}. \) First, assume that this unmatched vertex is \( a \in (A \cap \mathcal{R}) \setminus K_j. \) By our construction, there is only one path from \( s \) to \( a, \) which is simply the directed edge from \( s \) to \( a \) and its weight is 0. So \( \tilde{y}_a = d_a = 0. \) Next assume that this unmatched vertex is \( b \in (B \cap \mathcal{R}) \setminus K_j. \) By (1), \( \tilde{y}_b \geq 0. \) So let us assume that \( \tilde{y}_b > 0. \) Then \( d_b = -\tilde{y}_b < 0. \) By Lemma 1.2 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 \( a \in (A' \cap \mathcal{R}) \setminus K_j, \) 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 part (3), 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, \(-\tilde{y}_b + w_j = d_b + d(e) = d_a = \tilde{y}_a.\)
Finally, for part (4), consider an edge $e = (a, b)$ outside $M_j$ in $H_j$. Such an edge is directed from $a$ to $b$. So $\delta_a - w_e = d_a + d(e) \geq d_b = -\delta_b$. This, together with part (3), completes the proof. \qed

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

- Some vertex $a \in A \setminus K_j$ has $\delta_a < 0$. (However it cannot happen that some vertex $b \in B \setminus K_j$ has $\delta_b < 0$, since Lemma 1.1 states that such a vertex is in $R$ and Lemma 2.1 states that $\delta_b$ must be non-negative.)
- The edges deleted from $G_j$ (to form $H_j$) are not properly covered by the initial vertex cover $\{\delta_v\}_{v \in A \cup B}$.

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

where $\delta_1 = \max_{e = (a, b) \in E} \{w_e - \delta_a - \delta_b\}$ and $\delta_2 = \max_{a \in A \setminus K_j} \{-\delta_a\}$.

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 1.) We build the final vertex cover as follows.

1. For each vertex $u \in (A \cup B) \cap R$, let $y_u = \delta_u$;
2. For each vertex $a \in A \setminus R$, let $y_a = \delta_a + \delta$;
3. For each vertex $b \in B \setminus R$, let $y_b = \delta_b - \delta$.

**Theorem 1** The final vertex cover $\{y_v\}_{v \in A \cup B}$ is an optimal solution for the dual program.

**Proof** We first argue that $\{y_v\}_{v \in A \cup B}$ is a feasible dual solution. By Lemma 2.1 and the choice of $\delta$, all vertices $a \in A \setminus K_j$ have $y_a \geq 0$. By Lemma 1.1 and Lemma 2.1, all vertices $b \in B \setminus K_j$ have $y_b \geq 0$. Also by Lemma 1.2 and Lemma 2.3, and the choice of $\delta$, all edges in $E_j$ are properly covered. So $\{y_v\}_{v \in A \cup B}$ is feasible. We have:

$$w_j(M_j) = \sum_{e \in M_j} w_e^j = \sum_{e = (a, b) \in M_j, b \in R} \delta_a + \delta_b + \sum_{e = (a, b) \in M_j, b \notin R} (\delta_a + \delta) + (\delta_b - \delta)$$

where the equality holds because if a vertex $u$ is unmatched, Lemma 2.2 states that $\delta_u = 0$ and since $u$ must be in $R$, we have $y_u = \delta_u = 0$. Now by the linear programming duality theorem, we conclude that the cover $\{y_v\}_{v \in A \cup B}$ is optimal. \qed

Given $M_j$, it follows that the dual problem can be solved in time $O(m \sqrt{n})$ or $O(n^{\omega})$. We now show that $M_j$ can be computed in time $O(m \sqrt{n} \log n)$ or in time $O(n^{\omega})$ time with high probability. This follows from the following folklore technique: let $A_0$ and

© Springer
Let $A_1$ be two copies of the vertex set $A$, i.e., for $i = 0, 1$, let $A_i = \{a_i : a \in A\}$; similarly let $B_0$ and $B_1$ be two copies of the vertex set $B$ and let $E_0$ and $E_1$ be two copies of the edge set $E$.

Form the new bipartite graph $\tilde{G}_j = (X \cup Y, E'_j)$, where $X = A_0 \cup B_1$ and $Y = B_0 \cup A_1$. The edge set $E'_j = E_0 \cup E_1 \cup \{(u_0, u_1) : u \notin K_j\}$. The weight of an edge in $E_0 \cup E_1$ is the same as the weight of the corresponding original edge in $E$ while the weight of any edge $(u_0, u_1)$, where $u \notin K_j$, is 0.

Consider any maximum weight perfect matching $M^*_j$ in the graph $\tilde{G}_j$. By restricting $M^*_j$ to the edges of $E_0$, we obtain a matching $M_j$ in $G_j$ that matches all vertices in $K_j$. Also the weight of $M_j$ is exactly half the weight of $M^*_j$. We now show that $M_j$ has to be a maximum weight matching in $G_j$ that matches all vertices in $K_j$.

Suppose not and let $M'_j$ be a maximum weight matching in $G_j$ that matches all in $K_j$. So $w_j(M'_j) > w_j(M_j)$. By duplicating the edges of $M'_j$ (one copy for $A_0 \cup B_0$ and another for $A_1 \cup B_1$) and using the edges of weight 0 between the two copies of a vertex $u \notin K_j$, we get a perfect matching $\tilde{M}_j$ in $\tilde{G}_j$ whose weight is exactly twice that of $M'_j$. In other words, the weight of $\tilde{M}_j$ is more than that of $M^*_j$ and this contradicts our assumption that $M^*_j$ is a maximum weight perfect matching in $\tilde{G}_j$.

Recall that in the context of fair matchings, $c \leq 2 = O(1)$. Thus 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 [4,5,7], or in $\tilde{O}(n^\omega)$ time with high probability by Sankowski’s algorithm [18].

### 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. 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, where $r^*$ is the worst rank in a fair matching.

1. **Initialization.** Let $G_1 = G$ and $K_1 = \emptyset$.
2. For $j = 1$ to $r$ do
   
   (a) Find the optimal solution $\{y^j_u\}_{u \in A \cup B}$ to the dual program of the $j$-th 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., $K_{j+1} = K_j \cup \{u\}_{y^j_u > 0}$.
3. Return the optimal solution to the primal program of the last iteration.

The solution returned by our algorithm is a maximum weight matching in the graph $G_r$ under the weight function $w_r$ such that this matching matches all vertices in $K_r$. By Proposition 1, this is, in fact, a matching in the subgraph $G_{r+1}$ that matches all vertices in $K_{r+1}$. Lemma 4 proves the correctness of our algorithm. Lemma 3 guarantees that our algorithm is never “stuck” in any iteration due to the infeasibility of the primal or dual problem.
Lemma 3 The primal and dual programs of the $j$-th iteration are feasible, for $1 \leq j \leq r$.

Proof By linear programming duality, if the primal program of the $j$-th iteration is bounded from above and admits a feasible solution, then there is also a feasible solution for its dual. It is obvious that the primal program is bounded from above since it is upper bounded by $\sum_{e \in E} w_j(e) \leq 2m$. Therefore, to prove this lemma, we just need to show the feasibility of the primal program.

The base case is $j = 1$. Since $K_1 = \emptyset$, any matching in $G_1 = G$ is a feasible solution for the first primal program. For $j \geq 2$, we need to show that the primal program of the $j$-th iteration is feasible. By induction hypothesis, assume that the primal program of the $(j - 1)$-st iteration is feasible. Let $M_{j-1}$ denote its optimal solution. Since $M_{j-1}$ is a feasible point of the primal program of the $(j - 1)$-st iteration, $M_{j-1}$ uses only edges in $G_{j-1}$ and matches all vertices in $K_{j-1}$. Since $M_{j-1}$ is, in fact, an optimal solution to the primal program of the $(j - 1)$-st iteration, we will show that $M_{j-1}$ has to be a feasible point of the primal program of the $j$-th iteration by arguing that $M_{j-1}$ does not lose any of the edges pruned from $G_{j-1}$ and all vertices in $K_j$ are matched in $M_{j-1}$.

In step 2(b) of the $(j - 1)$-st iteration, we remove only those edges $e = (a, b)$ such that $y_{j-1}^a + y_{j-1}^b > w_{j-1}(e)$ from $G_{j-1}$ to form $G_j$. By the optimality of $M_{j-1}$, we know from Proposition 1.2 that $M_{j-1}$ has no slack edges, thus all edges in $M_{j-1}$ are retained in $G_j$.

We also know from Proposition 1.1 that if $y_{j-1}^u > 0$, then $u$ must be matched in $M_{j-1}$. Therefore, all vertices in $K_j \setminus K_{j-1}$ are matched in $M_{j-1}$. Moreover, as $M_{j-1}$ is also a feasible solution in the $(j - 1)$-st primal program, all vertices in $K_{j-1}$ are matched in $M_{j-1}$. This completes the proof of Lemma 3. \qed

Lemma 4 For every $0 \leq j \leq r$, the following hold:

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

Proof We proceed by induction. The base case is $j = 0$. Recall that $K_1 = \emptyset$ and $G_1 = G$. As all matchings are by default 0-optimal, the lemma holds vacuously.

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

$$w_j(M) = \sum_{e=(a,b)\in M} w_j(e) = \sum_{e=(a,b)\in M} y_{j}^a + y_{j}^b \geq \sum_{u \in A \cup B} y_{j}^u,$$

where the final inequality holds because all vertices $v$ with positive $y_{j}^v$ 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 $w_j(M)$.
Recall that by definition, $\text{OPT}$ is also $(j-1)$-optimal. By (2) of the induction hypothesis, $\text{OPT}$ is a matching in $G_j$ and $\text{OPT}$ matches all vertices in $K_j$. So $\text{OPT}$ is a feasible solution of the primal program in the $j$-th iteration. Thus $w_j(\text{OPT}) \leq w_j(M)$. However, it cannot happen that $w_j(\text{OPT}) < w_j(M)$, otherwise, signature($M$) $>$ signature($\text{OPT}$), since both $\text{OPT}$ and $M$ have the same first $j-1$ coordinates in their signatures. So we conclude that $w_j(\text{OPT}) = w_j(M)$, 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 $G_j$ that matches all vertices in $K_j$. So $M'$ is a feasible solution to the primal program of the $j$-th iteration. As signature($M'$) has $w_j(\text{OPT})$ in its $j$-th coordinate, $M'$ must be an optimal solution to the primal program of the $j$-th iteration; otherwise there is a $(j-1)$-optimal matching with a value larger than $w_j(\text{OPT})$ in the $j$-th coordinate of its signature, contradicting the optimality of $\text{OPT}$. By Proposition 1.2, all edges of $M'$ are present in $G_{j+1}$; by Proposition 1.1, all vertices $u \notin K_j$ with $y_u > 0$, in other words, all vertices in $K_{j+1}\backslash K_j$ have to be matched by the optimal solution $M'$. This completes the proof of (2).

Since our algorithm returns a matching in $G_{r+1}$ that matches all vertices in $K_{r+1}$, we know from Lemma 4.1 that this matching is $r$-optimal, thus the matching returned by our algorithm is fair. 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. Thus the above algorithm runs in time $O(rm\sqrt{n}\log n)$ or in time $\tilde{O}(rn^{\omega})$ time with high probability.

**Improving the running time** As mentioned earlier, our algorithm can be modified so that it terminates in $r^*$ iterations, where $r^*$ is the largest rank used in $\text{OPT}$. For that, we need to know the value of $r^*$ and this value can be computed right at the start of our algorithm as follows.

- Let $M_{\text{max}}$ be a maximum cardinality matching in $G$. The value $r^*$ is the smallest index $j$ such that the subgraph $\tilde{G}_j$ admits a matching of size $|M_{\text{max}}|$, where $\tilde{G}_j$ 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 $M_{\text{max}}$ and then computing a maximum cardinality matching in $\tilde{G}_1, \tilde{G}_2, \ldots$ and so on till we see a subgraph $\tilde{G}_j$ that admits a matching of size $|M_{\text{max}}|$. This index $j = r^*$ and it can be found in $O(r^*m\sqrt{n})$ time [20] or in $O(r^*n^{\omega})$ time [21,22].

Instead of the edge weight functions $w_1, \ldots, w_r$, we will use the weight functions $\tilde{w}_1, \ldots, \tilde{w}_{r^*}$ in our algorithm where for $1 \leq i \leq r^*$, $\tilde{w}_i$ is defined as follows:

- 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.
Note that for any \( j \in \{1, \ldots, r^*\} \), we have \( \tilde{w}_j(M) = w_{r-r^*+j}(M) \). For any matching \( M \), let \( \text{signature'}(M) = (\tilde{w}_1(M), \tilde{w}_2(M), \ldots, \tilde{w}_{r^*}(M)) \). In other words, for any matching \( M \), \( \text{signature'}(M) \) is the projection of the \( r^* \)-tuple \( \text{signature}(M) \) on to its \( \text{last} \ r^* \text{ coordinates}. \) We now show the following lemma.

**Lemma 5** A matching \( M \) such that the \( r^* \)-tuple \((\tilde{w}_1(M), \tilde{w}_2(M), \ldots, \tilde{w}_{r^*}(M))\) is lexicographically maximal is a fair matching in \( G \).

**Proof** The subgraph \( \tilde{G}_{r^*} \) admits a matching of size \( |M_{\text{max}}| \). Since \( \text{OPT} \) is a fair matching, it follows from the definition of a fair matching that \( \text{OPT} \) belongs to \( \tilde{G}_{r^*} \). So \( w_1(\text{OPT}) = w_2(\text{OPT}) = \cdots = w_{r-r^*+1}(\text{OPT}) = 2|M_{\text{max}}| \). So \( \tilde{w}_1(\text{OPT}) = 2|M_{\text{max}}| \).

Since \( M \) is lexicographically maximal under \( \text{signature'} \), we also have \( \tilde{w}_1(M) = 2|M_{\text{max}}| \). Hence, by the definition of the weight functions \( w_1, \ldots, w_{r-r^*} \), we have \( w_1(M) = \cdots = w_{r-r^*}(M) = 2|M_{\text{max}}| \). So \( M \) is \((r-r^*)\)-optimal (see Definition 4).

We have \( \text{signature'}(M) \succeq \text{signature}(\text{OPT}) \) since \( M \) is lexicographically maximal under \( \text{signature'} \). However if \( \text{signature}(M) \succeq \text{signature}(\text{OPT}) \), then \( \text{signature}(M) \succ \text{signature}(\text{OPT}) \) since \( M \) is \((r-r^*)\)-optimal. Thus \( \text{signature'}(M) = \text{signature'}(\text{OPT}) \), i.e., \( \text{signature}(M) \) and \( \text{signature}(\text{OPT}) \) agree on their last \( r^* \) coordinates. Along with the fact that \( M \) is \((r-r^*)\)-optimal, this implies \( \text{signature}(M) = \text{signature}(\text{OPT}) \). Hence \( M \) is a fair matching in \( G \).

Our new algorithm is the same as our earlier algorithm, except that we will use the edge weight functions \( \tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_{r^*} \) instead of \( w_1, w_2, \ldots, w_r \). As before, we initialize \( G_1 = G \) and \( K_1 = \emptyset \). For each \( j \), where \( 1 \leq j \leq r^* \), the primal program of the \( j \)-th iteration is a maximum weight matching in the subgraph \( G_j \) under the edge weight function \( \tilde{w}_j \) that matches all in \( K_j \). As before, in the \( j \)-th iteration, we first solve the dual program of this iteration, then prune the subgraph \( G_j \) to \( G_{j+1} \) and grow the critical set \( K_j \) to \( K_{j+1} \).

Lemmas 3 and 4 hold here as well (where optimality is defined with respect to \( \text{signature'} \)) and thus the matching returned by the above algorithm after \( r^* \) iterations will be a matching \( M \) such that \((\tilde{w}_1(M), \tilde{w}_2(M), \ldots, \tilde{w}_{r^*}(M))\) is lexicographically maximal. It follows from Lemma 5 that \( M \) is a fair matching. Thus we can conclude Theorem 2.

**Theorem 2** A fair matching \( M \) in \( G = (A \cup B, E) \) can be computed in \( \tilde{O}(r^* m \sqrt{n}) \) time, or in \( O(r^* n^\omega) \) 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.

### 2.3 Two-Sided Rank-Maximality

In this section we show how our fair matching algorithm can be modified to solve the rank-maximal matching problem and the maximum cardinality rank-maximal matching problem in the domain of two-sided preference lists. For both these problems, we use the same algorithm that we designed for fair matchings—the only difference is in how the edge weight functions \( w_1, \ldots, w_r \) are defined.
Consider the maximum cardinality rank-maximal matching problem. Here the function \( w_1: E \to \{1\} \) is a constant function, i.e., \( w_1(e) = 1 \) for all \( e \in E \). For \( 2 \leq i \leq r \) and edge \( e = (a, b) \), we have \( w_i(e) = 2 \) if both \( a \) and \( b \) rank each other as rank \( i - 1 \) neighbors, it is 1 if exactly one of \( \{a, b\} \) ranks the other as a rank \( i - 1 \) neighbor, otherwise it is 0.

Let \( \text{OPT} \) be the optimal matching under the above weight functions, in other words, \( \text{OPT} \) has maximum weight under \( w_1 \), subject to this, it has maximum weight under \( w_2 \), and so on. It is easy to see that \( \text{OPT} \) is a maximum cardinality rank-maximal matching in \( G \). When the algorithm given in Sect. 2.2 is run using these edge weight functions, by Lemmas 3 and 4, the matching \( M \) returned satisfies \( w_j(M) = w_j(\text{OPT}) \) for all \( 1 \leq j \leq r \). That is, \( \text{signature}(M) = \text{signature}(\text{OPT}) \), so \( M \) is a maximum cardinality rank-maximal matching.

In the rank-maximal matching problem, we define the functions \( w_1, \ldots, w_r \) from \( E \) to \( \{0, 1, 2\} \) as follows. For \( 1 \leq i \leq r \) and \( e = (a, b) \), we have \( w_i(e) = 2 \) if both \( a \) and \( b \) rank each other as rank \( i \) neighbors, it is 1 if exactly one of \( \{a, b\} \) ranks the other as a rank \( i \) neighbor, otherwise it is 0. Using these edge weight functions and our algorithm from Sect. 2.2, we get a rank-maximal matching in the two-sided preference lists model.

Both of the above algorithms can be made to terminate in \( r^* \) iterations, instead of \( r \) iterations, by adding an extra step at the end of Step 2 in every iteration of the main algorithm in Sect. 2.2:

Step 2(d) (in the \( j \)-th iteration). Define the graph \( G'_j \) as follows: the edge set is exactly the same as in \( G_{j+1} \); however the edge weight function \( w'_j \) is as follows. For each edge \( e \) in \( G'_j \): set \( w'_j(e) = 1 \) if \( \sum_{i=j+1}^r w_i(e) > 0 \), else set \( w'_j(e) = 0 \).

Now find a maximum \( w'_j \)-weight matching \( M'_{j+1} \) in \( G'_j \) that matches all vertices in \( K_{j+1} \). If the weight of \( M'_{j+1} \) is 0, then return \( M_j \) as the final solution, where \( M_j \) is the primal optimal solution of the \( j \)-th iteration.

We show the correctness of this additional step via the following lemma.

**Lemma 6** The following claims hold:

1. For any \( j \geq 1 \), if the matching \( M'_{j+1} \) has weight 0, then \( M_j \) is a rank-maximal/cardinality rank-maximal matching.
2. If \( r^* \) is the largest rank incident on an edge in \( \text{OPT} \), then \( M'_{r^*+1} \) has weight 0.

**Proof** We first show (1). Suppose \( w'_j(M'_{j+1}) = 0 \). Since all edge weights are non-negative, we claim that any matching \( M \) in \( G_{j+1} \) that matches all vertices in \( K_{j+1} \) must satisfy \( w_{j+1}(M) = \cdots = w_r(M) = 0 \); otherwise \( w'_j(M) > 0 \), a contradiction to the assumption that the maximum weight matching \( M'_{j+1} \) in \( G'_j \) that matches all vertices of \( K_{j+1} \) satisfies \( w'_j(M'_{j+1}) = 0 \).

As a result, both \( M_j \) and \( \text{OPT} \) have 0 in the last \( r - j \) coordinates in their signatures, since \( M_j \) (by its optimality in \( G_j \)) and \( \text{OPT} \) (by Lemma 4.2) are matchings in \( G_{j+1} \) that match all vertices of \( K_{j+1} \). By Lemma 4.1, \( M_j \) is \( j \)-optimal. Hence \( \text{signature}(M_j) = \text{signature}(\text{OPT}) \).
signature(OPT), thus $M_j$ is a rank-maximal/maximum cardinality rank-maximal matching and (1) is proved.

We prove (2) by contradiction. Suppose $r^*$ is the largest rank on an edge incident in OPT and we have $w'_{r^*}(M'_{r^*+1}) > 0$. Since $M'_{r^*+1}$ is a matching in $G_{r^*+1}$ that matches all vertices in $K_{r^*+1}$, it follows from Lemma 4.1 that $M'_{r^*+1}$ is $r^*$-optimal. While OPT has only 0 in its last $r - r^*$ coordinates, $M'_{r^*}$ has some positive values in its last $r - r^*$ coordinates; so we have $M'_{r^*} >$ OPT. This is a contradiction to the optimality of OPT.

We conclude this section with the following theorem.

**Theorem 3** A rank-maximal/maximum cardinality rank-maximal matching 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 $q : V \rightarrow \mathbb{Z}_{>0}$. 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 [23,24]. Let $G = (V, E)$ be a directed network with a cost $c : E \rightarrow \mathbb{Z}$ and capacity $u : E \rightarrow \mathbb{Z}_{\geq0}$ 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 \( x : E \rightarrow \mathbb{Z} \) 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 \( \text{cost}(x) = \sum_{e \in E} c(e)x(e) \).

The minimum-cost flow problem asks for a flow of minimum cost. Instead of using generalized cost-scaling, we use the traditional cost-scaling approach by solving a sequence of problems, \( P_0, P_1, \ldots, P_K \), on the network with original capacities but approximate costs. The cost function \( c_i \) for \( P_i \), is obtained by taking the \( i \)

### 3.1 Cost-Scaling

We assume that capacities, demands and supplies are polynomially bounded in \( n \), but costs may be exponential in the input size. We modify the algorithm of Goldberg and Tarjan [9] for the capacitated transshipment problem; see also [23, Section 10.3].

**Generalized cost-scaling** The idea of the generalized cost-scaling technique of Goldberg and Tarjan [24] is to view the maximum amount of violation of the complementary slackness conditions as an error parameter, and to reduce this parameter at each iteration, therefore improving the solution. Each iteration performs a refinement computation, which is similar to a maximum flow computation. The refinement starts with an \( \varepsilon \)-optimal flow and computes an \( \frac{\varepsilon}{2} \)-optimal flow, reducing the violation by a factor of 2. Initially the error is at most the magnitude of the largest edge cost \( C \), and if the error is less than \( 1/n \), then the current solution is optimal. Thus, the generalized cost-scaling method terminates in \( O(\log(nC)) \) iterations. Goldberg and Tarjan have shown how to perform such a refinement in \( O(mn \log(n^2/m)) \) time.

In their algorithm node potentials and reduced edge costs may become as large as \( O(C) \). We add two ingredients to keep the numbers small. First, we fix the flow across edges of large (positive or negative) reduced cost and remove them from the network. Second, we do not maintain the node potentials but only the reduced costs of the remaining edges. Note that these costs are small. The idea of fixing flows for edges of large reduced cost has been previously used in network problems in order to obtain strongly polynomial algorithms [23, page 397].

Figure 3 shows our algorithm in detail when applied to the fair b-matching problem. Instead of using generalized cost-scaling, we use the traditional cost-scaling approach by solving a sequence of problems, \( P_0, P_1, \ldots, P_K \), on the network with original capacities but approximate costs. The cost function \( c_i \) for \( P_i \), is obtained by taking the \( i \)
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 $-(4n - n^{-1} - n^{-2})$.
(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 $2^{\lfloor \log n \rfloor}$ to make them divisible by the same amount.
(b) Let $K = \lfloor \log C \rfloor$ 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 $1$-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 $\bar{c}_i(e) = 2c_{i-1}(e) + (1 \text{ if } e \in \mathcal{E}_i, \text{ else } 0) \times \text{sign}(e)$, where $\text{sign}(e) = \pm 1$ depending on the sign of the original cost $c(e)$. The flow $x_{i-1}$ is 3-optimal with respect to the cost function $\bar{c}_i$ and the zero potential function, i.e., the potential of all the vertices is 0.
(b) Refine the flow to be 1-optimal using the algorithm from [9] with input (i) the flow $x_{i-1}$, (ii) $\bar{c}_i$, as the edge cost function and (iii) the zero potential function. Let $x_i$ be the 1-optimal flow and $\tilde{\pi}$ be the potential function which proves the 1-optimality.

/* 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 a 1-optimal flow. See Lemma 7. */
(c) Compute new reduced costs as $c_i(u, v) = \bar{c}_i(u, v) + \tilde{\pi}(u) - \tilde{\pi}(v)$.
/* Implicitly, this defines $\pi(v)$ as $2\pi_{i-1}(v) + \tilde{\pi}(v)$. */
(d) Let $d$ be the constant from Lemma 7. If any edge $e \in E$ has $|c_i(e)| > d \cdot n + 1$, then 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.

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

most significant bits of the original cost function $c$. Each phase of our algorithm finishes with a 1-optimal flow. Due to cost-scaling this 1-optimal flow becomes 3-optimal at the beginning of the next phase. At this point we use the refinement procedure of Goldberg and Tarjan [9] as a black box in order to transform the 3-optimal flow into a 1-optimal flow. The refinement procedure for reducing the error parameter by a factor of 3 has the same asymptotic complexity as the version that reduces by a factor of 2, only constant factors are affected. We summarize the properties of the refinement algorithm using our own notation in Lemma 7.

3.2 The Main Algorithm

We now give the details. We assume that all costs are divisible by $2^{1+\lfloor \log n \rfloor}$. In case they are not, we can always multiply them by $2^{1+\lfloor \log n \rfloor}$. This guarantees that a 1-optimal flow is optimal\(^4\), see [23, Lemma 10.2]. Let $K = \lfloor \log C \rfloor$ where $C$ is the magnitude of the largest edge cost. We assume that each cost is a signed integer $\pm b_1b_2\ldots b_K$, where $b_1$ is the most significant bit. For $0 \leq i \leq K$, let $c_i(e) = \lfloor c(e)/2^{K-i} \rfloor$ be the

\(^4\) Let $\pi$ be the potential function and let $C$ be any directed cycle in the residual graph to the 1-optimal flow. Then $\sum_{e \in \mathcal{C}} c_e = \sum_{e \in \mathcal{C}} c_e \pi \geq n \cdot (-1)$. Since edge costs are multiples of $2^{1+\lfloor \log n \rfloor}$, the sums must be nonnegative and hence there is no negative cost cycle in the residual graph. This implies optimality.
cost obtained by considering only the first \( i \)-bits of the binary representation of \( c(e) \) when viewed as numbers with \( K \) bits.

Our scaling algorithm works in phases; in phase \( i \) it computes a 1-optimal flow with respect to the cost function \( c_i \) and some potential function \( \pi_i \). During the algorithm we store explicitly only the reduced costs and not the potential function. Nevertheless, we use \( \pi_i \) to refer to the potential function in phase \( i \). In phase zero we set the cost of every edge to zero and calculate any feasible flow \( x_0 \). The zero potential function on the vertices, i.e., the potential of all the vertices is 0, guarantees the 1-optimality of \( x_0 \). We therefore set the reduced cost \( c_0^{\pi_0}(e) = 0 \) for any \( e \in E \).

Consider next any phase \( i \) with \( i \geq 1 \). At the beginning of the phase, we have a 1-optimal flow \( x_{i-1} \) and a reduced cost function \( c_{i-1}^{\pi_{i-1}} \). 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. We scale up by multiplying the potentials and the edge costs by 2 and adding \( \pm 1 \) (after multiplying by 2) to the edge costs of edges in \( \mathcal{E}_i \) (we either add or subtract based on the sign of \( c(e) \)). The reduced costs scale up as follows:

\[
\tilde{c}_i(e) = 2c_{i-1}^{\pi_{i-1}}(e) + (1 \text{ if } e \in \mathcal{E}_i \text{ else } 0) \times \text{sign}(e) \tag{1}
\]

where \( \text{sign}(e) \) is 1 or \(-1\) depending on the sign of the original cost \( c(e) \) of \( e \).

At the end of phase \( i-1 \), the flow \( x_{i-1} \) is 1-optimal and therefore \( c_{i-1}^{\pi_{i-1}}(e) \geq -1 \) for any edge \( e \) with positive residual capacity. After scaling up, Eq. (1) gives us that any edge with positive residual capacity has \( \tilde{c}_i(e) \geq -3 \). This is equivalent to saying that \( x_{i-1} \) is 3-optimal with respect to the edge costs \( \tilde{c}_i \) and the zero potential function on the vertices. We use the refinement algorithm of Goldberg and Tarjan [9] to transform the 3-optimal flow into a 1-optimal flow. We summarize the properties and the running time of the refinement using our own notation into the following lemma.

**Lemma 7 (from [9], see also [24])** Given the edge cost function \( \tilde{c}_i \) and a 3-optimal flow \( x_{i-1} \) with respect to the zero potential function, in time \( O(mn \log (n^2/m)) \) one can compute a flow \( x_i \) and a potential function \( \tilde{\pi} \) such that \( x_i \) is 1-optimal with respect to the potential function \( \tilde{\pi} \). Potentials are only decreased, starting from zero, during the computation and \( \tilde{\pi}(v) \geq -d \cdot n \) for some constant \( d \) and all \( v \).

After the refinement we conceptually set \( \pi_i(v) = 2\pi_{i-1}(v) + \tilde{\pi}(v) \) and recompute the reduced costs as \( c_i^{\pi_i}(u,v) = \tilde{c}_i(u,v) + \tilde{\pi}(u) - \tilde{\pi}(v) \). We next show that edges of large reduced cost can be discarded; more precisely, we can fix their flow to either zero or the capacity.

**Handling large reduced costs** We say that the reduced cost of an edge \( e \) is large positive from phase \( i \) on, if \( c_j^{\pi_j}(e) > 1 \) at the end of every phase \( j \geq i \), and is large negative from phase \( i \) on, if \( c_j^{\pi_j}(e) < -1 \) at the end of every phase \( j \geq i \). If \( e \) is large positive from phase \( i \) on, we have \( x_j(e) = 0 \) for all \( j \geq i \) and if \( e \) is large negative from phase \( i \) on, we have \( x_j(e) = u(e) \) for all \( j \geq i \).

**Lemma 8** Let \( e = (u,v) \) and \( d \) be the constant from Lemma 7. If \( \tilde{c}_i(e) > 2dn + 1 \) at the beginning of phase \( i \), \( e \) is large positive from phase \( i \) on, and if \( \tilde{c}_i(e) < -2dn - 1 \) at the beginning of phase \( i \), \( e \) is large negative from phase \( i \) on.
Proof Assume $\tilde{c}_i(e) > 2dn + 1$ at the beginning of some phase $i$. Then, at the end of phase $i$ we have $c_\pi^\top_i(e) = \tilde{c}_i(u, v) + \tilde{\pi}(u) - \tilde{\pi}(v) > dn + 1$ since we know from Lemma 7 that $-dn \leq \tilde{\pi}(v) \leq 0$ for all $v \in V$. Therefore, at the beginning of the next phase $\tilde{c}_{i+1}(e) = 2c_\pi^\top_i(e) + (1$ if $e \in E_{i+1}$ else 0) $\times sign(e) > 2dn + 2 - 1 = 2dn + 1$. Thus, $e$ is large positive from phase $i$ onwards.

Assume next that $\tilde{c}_i(e) < -dn - 1$ at the beginning of some phase $i$. Then, at the end of phase $i$ we have $c_\pi^\top_i(e) = \tilde{c}_i(u, v) + \tilde{\pi}(u) - \tilde{\pi}(v) < -dn - 1$ since we know from Lemma 7 that $-dn \leq \tilde{\pi}(v) \leq 0$ for all $v \in V$. At the beginning of the next phase $\tilde{c}_{i+1}(e) = 2c_\pi^\top_i(e) + (1$ if $e \in E_{i+1}$ else 0) $\times sign(e) < -2dn - 2 + 1 = -2dn - 1$. Thus, $e$ is large negative from phase $i$ onwards. □

Lemmas 7 and 8 allows us to fix the flow across an edge $e$ once $|\tilde{c}_i(e)| > dn + 1$. We remove the edge $e$ (and its reversal $e^R$) from the graph and modify the imbalances of both endpoints accordingly, by adjusting supply and demand values. Thus, all reduced costs can be bounded by $O(n)$ and all arithmetic operations involve polynomially bounded numbers. Theorems 4 and 5 now follow.

Theorem 4 The minimum cost flow problem with polynomially bounded demands can be solved in $O(T + mn \log(n^2/m) \log(nC))$ time and $O(S + m)$ space using arithmetic only on numbers of size polynomial in $n$, where $C$ is the magnitude of the largest edge cost and $T$ and $S$ are the time and space bounds of an algorithm to sequentially enumerate the sets $E_i$, $1 \leq i \leq \lceil \log C \rceil$, of all edges having a 1 in the $i$-th bit of their weight.

Theorem 5 The fair $b$-matching problem can be solved in $\tilde{O}(rmn)$ time and $O(m)$ space.

References

1. Huang, C.-C., Kavitha, T.: Weight-maximal matchings. In: Proceedings of the the 2nd International Workshop on Matching under Preferences, July (2012)
2. Mehlhorn, K., Michail, D.: Network Problems with Non-Polynomial Weights and Applications. www.mpi-sb.mpg.de/mehlhorn/ftp/HugeWeights.ps
3. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. 69, 9–15 (1962)
4. Orlin, J.B., Ahuja, R.K.: New scaling algorithms for the assignment and minimum mean cycle problems. Math. Program. 54(1), 41–56 (1992)
5. Duan, R., Su, H.-H.: A scaling algorithm for maximum weight matchings in bipartite graphs. In: Proceedings of the 23rd SODA, pp. 1413–1424 (2012)
6. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34(3), 596–615 (1987)
7. Gabow, H., Tarjan, R.: Faster scaling algorithms for network problems. SIAM J. Comput. 18, 1013–1036 (1989)
8. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by successive approximation. Math. Oper. Res. 15, 430–466 (1990)
9. Sng, C.: Efficient Algorithms for bipartite matching problems with preferences Ph.D. thesis, University of Glasgow, (2008)
10. Irving, R.W.: Greedy Matchings. University of Glasgow, Computing Science Department Research Report, TR-2003-136, (2003)
12. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.E.: Rank-maximal matchings. ACM Trans. Algorithms 2(4), 602–610 (2006)
13. Kavitha, T., Shah, C.D.: Efficient algorithms for weighted rank-maximal matchings and related problems. In: Proceedings of the 17th ISAAC, pp. 153–162 (2006)
14. Michail, D.: Reducing rank-maximal to maximum weight matching. Theor. Comput. Sci. 389(1–2), 125–132 (2007)
15. Paluch, K.: Capacitated rank-maximal matchings. In Proceedings of the 8th CIAC, pp. 324–335 (2013)
16. Goldberg, A.V.: Scaling algorithms for the shortest paths problem. SIAM J. Comput. 24(3), 494–504 (1995)
17. Sankowski, P.: Shortest paths in matrix multiplication time. In: Proceedings of the 13th ESA, pp. 770–778 (2005)
18. Sankowski, P.: Maximum weight bipartite matching in matrix multiplication time. Theor. Comput. Sci. 410, 4480–4488 (2009)
19. Yuster, R., Zwick, U.: Answering distance queries in directed graphs using fast matrix multiplication. In: Proceedings of the 46th FOCS, pp. 90–100 (2005)
20. Hopcroft, J., Karp, R.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2, 225–231 (1973)
21. Harvey, N.J.A.: Algebraic structures and algorithms for matching and matroid problems. SIAM J. Comput. 39(2), 679–702 (2009)
22. Mucha, M., Sankowski, P.: Maximum matchings via gaussian elimination. In: Proceedings of the 45th FOCS, pp. 248–255 (2004)
23. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, USA (1993)
24. Goldberg, A.V., Tardos, E., Tarjan, R.E.: Network flow algorithms. In: Paths, Flows and VLSI-Design, pp. 101–164. Springer, Berlin (1990)