Network Problems with Non-Polynomial Weights and Applications

(Submitted to Track A)

Kurt Mehlhorn and Dimitrios Michail

Max-Planck-Institut für Informatik, Saarbrücken, Germany
{mehlhorn,michail}@mpi-sb.mpg.de

Abstract. The most efficient algorithms for several network problems like minimum cost flow and the maximum weight matching problem follow the primal-dual paradigm. These algorithms perform arithmetic (additions and subtractions) on numbers of magnitude $O(nC)$ when the edge weights (also called costs) are integers bounded by C and n denotes the number of vertices. Under the standard assumption that arithmetic on numbers of magnitude $O(n)$ has constant cost, arithmetic on numbers of this size has cost $O((\log C)/\log n)$. We show for the scaling versions of these algorithms that arithmetic on numbers of size polynomial in n suffices without increasing the asymptotic number of arithmetic operations. In this way, we obtain an $O(T+\sqrt{n}m\log(nC))$ time and $O(S+m)$ space algorithm for the maximum weight matching problem on bipartite graphs where T and S are the time and space bound of an algorithm to sequentially enumerate the sets $\mathcal{E}_i$, $1 \le i \le \lceil \log C \rceil$, of all edges having a one in the i-th bit of their weight. The previously best algorithm had running time $O(\sqrt{n}m(\log(nC))^2/\log n)$. We obtain similar improvements for the capacitated transshipment problem with polynomially bounded demands.

Some matching problems on bipartite graphs reduce to weighted matching problems with $C = n^r$, $r \le m$. For these problems, we have $T = O(r\sqrt{n}m\log n)$ and $S = O(m)$ and hence our algorithm improves upon previous solutions by a factor $\Theta(r)$.

1 Introduction

This paper was inspired by a study of rank-optimal matching problems. Let $G = (V,E)$ be a bipartite graph and let $E = E_1 \dot\cup E_2 \dot\cup \ldots \dot\cup E_r$, $r \le m$ be a partition of the edge set. We call the edges in class E_i the edges of rank i. For a matching M in G, let $s_i(M) = |M \cap E_i|$ be the number of edges in M of rank i. We call a matching *rank-maximal* if it maximizes the vector $(s_1, s_2, \ldots, s_r)$, *maximum cardinality rank-maximal* if it maximizes the vector $(\sum_i s_i, s_1, s_2, \ldots, s_r)$, and *fair* if it maximizes $(\sum_i s_i, -s_r, -s_{r-1}, \ldots, -s_1)$. All three versions of the rank-optimal matching problems reduce to weighted matching problems. For example, if an edge of rank i is given weight $2^{\lceil \log n \rceil \cdot (r-i)}$ then maximum weight matchings correspond to rank-maximal matchings. Observe that the binary representation of these weights has a simple structure.

The running times of the best strongly and weakly polynomial algorithms for the maximum weight matching problem in bipartite graphs are usually stated as $O(nm + n^2 \log n)$ [1] and $O(\sqrt{n}m \log(nC))$ [2], respectively. Here n and m denote the number of vertices and edges, respectively, and C denotes the maximal edge weight. Edge weights are assumed to be integral. The bounds just stated assume that arithmetic operations have constant cost. The algorithms perform arithmetic (additions and subtractions) on numbers of magnitude $O(nC)$; arithmetic on numbers of this size has cost $O((\log C)/\log n)$ under the customary assumption that arithmetic on numbers of magnitude $O(n)$ has constant cost. Thus, the true time bounds are a factor $(\log C)/\log n$ larger than the bounds stated above. In particular, the running time of the weakly polynomial algorithm is $O(\sqrt{n}m(\log(nC))^2/\log n)$. In the application to rank-maximal matchings $C = 2^{\lceil \log n \rceil \cdot (r-1)}$ and hence the time bounds of the strongly and weakly polynomial algorithms are $O(rnm)$ and $O(r^2\sqrt{n}m \log n)$, respectively. *We show that the use of numbers of magnitude $O(nC)$ can be avoided in the weakly polynomial algorithm; it suffices to handle numbers of polynomial size. In this way, we avoid the penalty factor $(\log C)/\log n$.*

Our work is based on the weakly polynomial algorithm of Gabow and Tarjan. This algorithm uses scaling. Let $c(e)$ be the weight (cost) of edge e and let $K = \lceil \log C \rceil$. For $0 \le i \le K$, let $c_i(e) = \lfloor c(e)/2^{K-i} \rfloor$ be the weight obtained by considering only the first i-bits of the binary representation of $c(e)$ when viewed as numbers with K bits. The algorithm operates in K phases. In the i-th phase, it computes a nearly optimal matching M_i for the cost function c_i and a potential function π_i proving the near-optimality of the matching. A potential function assigns an integer to each vertex. It guides the construction of the matching M_{i+1} in the next phase through the use of reduced costs. For a potential function $\pi : V \mapsto \mathbb{Z}$ and a cost function $c : E \mapsto \mathbb{Z}_{\ge 0}$, the *reduced cost* of an edge $e = (v, w)$ is defined as $\overline{c}(e) = \pi(v) + \pi(w) - c(e)$. In the Gabow/Tarjan algorithm, potential values and reduced costs may become as large as $\Theta(nC)$. Our improved algorithm is based on the following observations:

- Edges whose reduced cost becomes sufficiently large (= larger than $8n$) in some phase can be safely ignored ("priced out" in the terminology of combinatorial optimization) in later phases. They will never be part of an optimal matching.
- If potential values are not stored explicitly, but only the reduced cost of edges, only numbers polynomially bounded in n need to be handled and hence arithmetic has constant cost.

In this way, we obtain an $O(T + \sqrt{n}m \log(nC))$ time and $O(S + m)$ space algorithm for the maximum weight matching problem on bipartite graphs where T and S are the time and space bound of an algorithm to sequentially enumerate the sets $\mathcal{E}_i$, $1 \le i \le \lceil \log C \rceil$, of all edges having a one in the i-th bit of their weight (each weight is assumed to be a signed integer $\pm b_1 b_2 \ldots b_{\lceil \log C \rceil}$, where b_1

is the most-significant bit). We obtain similar improvements for the capacitated transshipment problem with polynomially bounded demands.

In the application to rank-optimal matching problems, we have $T = O(r\sqrt{n}m \log n)$ and $S = O(m)$ and hence we obtain running time $O(r\sqrt{n}m \log n)$ and space requirement $O(m)$. Similarly for rank-optimal b-matching problems we obtain $O(rnm \log(n^2/m) \log n)$ time and space $O(m)$. For some of these problems r can be replaced by $r^* \leq r \leq m$, which for a particular instance is the maximal rank used in an optimal solution.

The paper is organized as follows. In Section 2 we discuss the maximum weight matching problem and in Section 4 the minimum cost flow problem. In Sections 3 and 5 we give several applications. We summarize and offer conclusions in Section 6.

2 Maximum weight bipartite matching

We give a variant of the Gabow/Tarjan scaling algorithm [2] for weighted bipartite matchings which runs in time $O(T + \sqrt{n}m \log nC)$ where C is the largest weight of any edge on the graph. The algorithm ensures that any arithmetic performed will be on numbers of size polynomial in n. The space requirement is $O(S + m)$.

2.1 Preliminaries

We are given a bipartite graph $G = (V, E)$, $V = A \dot\cup B$ and a non-negative weight function on the edges, $c : E \mapsto \mathbb{Z}_{\geq 0}$. We assume that all weights are divisible by $2^{\lceil 1+\log n \rceil}$; this assumption can always be ensured by multiplying the weights by $2^{\lceil \log n \rceil}$. We are looking for a maximum weight matching M. A well-known transformation allows us to assume that $|A| = |B|$ and that there is a maximum weight matching which is also perfect.

We create a graph $\hat{G}$ with two copies of G, namely G_1 and G_2. For any vertex $v \in G$, let $v_1 \in G_1$ and $v_2 \in G_2$ be the two copies of v. We add a zero cost edge between v_1 and v_2. Observe that $\hat{G}$ is bipartite if G is and that both sides of the bipartition of $\hat{G}$ have the same cardinality. For any matching M in G there is perfect matching of twice the cost in $\hat{G}$, use a copy of M each in G_1 and G_2 and match vertices free in M with their copy by the new zero cost edges. Conversely, any matching $\hat{M}$ in $\hat{G}$ induces matchings M_1 and M_2 in G_1 and G_2, respectively, whose cost adds up to the cost of $\hat{M}$. Thus, in $\hat{G}$, a maximum weight matching can be assumed to be perfect. We assume from now on that the transformation has been performed and use G to denote the transformed graph and n to denote the common cardinality of the two sides of the bipartition of G.

We need a few definitions. A matching M is called *1-feasible* matching for cost function c if there are dual variables $\pi(v), v \in V$ such that:

$$\pi(u) + \pi(v) \geq c(e) - 1 \qquad \forall e = (u, v) \in E \tag{1a}$$

$$\pi(u) + \pi(v) = c(e) \qquad \forall e = (u, v) \in M \tag{1b}$$

A matching M is *1-optimal* if it is 1-feasible and it is perfect. The *reduced cost* of an edge $e = (v, w) \in E$ with respect to a cost function c and a potential function π is defined as $\bar{c}(e) = \pi(v) + \pi(w) - c(e)$.

Bertsekas [3, 4] observed that, under the assumption that weights are divisible by $2^{1+\lceil \log n \rceil}$, a 1-optimal matching is optimal. Indeed, let M^* be 1-optimal, let π be a potential function proving 1-optimality, and let P be any perfect matching. Then

$$c(M^*) = \sum_{e \in M^*} c(e) = \sum_{v \in V} \pi(v) \geq c(P) - n.$$

On the other hand, $c(P) - c(M^*)$ is divisible by $2^{1+\lceil \log n \rceil}$ and hence $c(P) \leq c(M^*)$.

2.2 The algorithm

The algorithm operates in phases 0 to K. In phase zero all edges have cost zero. We set M_0 to an arbitrary perfect matching, e.g., the edges added in the transformation presented in Section 2.1. We also set the reduced cost of all edges to zero and, implicitly, the potential of all nodes to zero. During the algorithm we will only maintain the reduced costs. With these settings, M_0 is an 1-optimal matching and phase zero ends.

Consider any latter phase. At the beginning of phase i we have a 1-optimal matching M_{i-1} and reduced costs $\bar{c}_{i-1}(e)$ for $e \in E$. Our goal it to compute a 1-optimal matching M_i for the cost function c_i. Let $b_i(e) = 1$ if $e \in \mathcal{E}_i$ and 0 otherwise and define an auxiliary cost function $\tilde{c}$ by

$$\tilde{c}(e) = 2\bar{c}_{i-1}(e) + 2 - b(e).$$

Then $\tilde{c}(e) \geq 2 \cdot (-1) + 2 - 1 = -1$ and $\tilde{c}(M_{i-1}) = \sum_{e \in M_{i-1}} 2 - b(e) \leq 2n$. We are now in a position to use a result of Gabow and Tarjan.

Lemma 1. *Let $b > 0$ be a constant and let G be a bipartite graph with cost function $\tilde{c}$. If $\tilde{c}(e) \geq -1$ and there is a perfect matching with cost at most bn, a perfect matching M and a potential function π with (1) $\tilde{c}(e) = \pi(u) + \pi(v)$ for all $e = (u, v) \in M$, (2) $\tilde{c}(e) \geq \pi(u) + \pi(v) - 1$ for all $e = (u, v) \in E$, and (3) $|\pi(v)| \leq (b + 2)n$ for all $v \in A \cup B$ can be determined in time $O(\sqrt{n}m)$.*

We can use this Lemma with $b = 2$. Let M and π be as in this Lemma. We implicitly set $\pi_i(v) = 2\pi_{i-1}(v) - \pi(v) + 2 \cdot (v \in A)$. Then for $e = (u, v)$

$$\begin{aligned} \bar{c}_i(e) &= \pi_i(u) + \pi_i(v) - c_i(e) \\ &= 2\pi_{i-1}(u) - \pi(u) + 2 + 2\pi_{i-1}(v) - \pi(v) - 2c_{i-1}(e) - b(e) \\ &= 2\bar{c}_{i-1}(e) + 2 - b(e) - \pi(u) - \pi(v) = \tilde{c}(e) - \pi(u) - \pi(v) \end{aligned}$$

and hence $\bar{c}_i(e) = 0$ for $e \in M$ and $\bar{c}_i(e) \geq -1$ for all $e \notin M$. Thus $M_i = M$ is 1-optimal. We need one more fact. Let $M' = M_{i-1}$ be the matching with $\tilde{c}(M') \leq bn$. Then

$$\tilde{c}(M) = \sum_{e=(u,v) \in M} \pi(u) + \pi(v) \leq \sum_{e=(u,v) \in M'} \tilde{c}(e) + 1 \leq (b + 1)n$$

and hence $\tilde{c}(e) \le (b+2)n$ for any $e \in M$.

We compute $\bar{c}_i(e)$ as $\bar{c}_i(e) = 2\bar{c}_{i-1}(e) + 2 - b(e) - \pi(u) - \pi(v)$ and so the update is a change by a number of polynomial size. The next lemma shows how to ensure that reduced costs are in $O(n)$ as well.

Lemma 2. *Consider any edge e with $\bar{c}_{i-1}(e) \ge 8n$. Then e is not part of the 1-optimal matching computed in any phase $j \ge i$.*

Proof. If $\bar{c}_{i-1}(e) \ge 8n$, $\tilde{c}_i(e) \ge 16n$. Thus $e \notin M_i$ by the discussion above. Also $\bar{c}_i(e) = 2\bar{c}_{i-1}(e) + 2 - b(e) - \pi(u) - \pi(v) \ge 16n - 8n \ge 8n$.

During the algorithm we maintain the reduced costs. When an edge reaches reduced cost $8n$ or more, we delete it. In this way, reduced costs stay in $O(n)$. Summarizing, we have the following theorem.

Theorem 1. *The maximum weight matching problem on a bipartite graph $G(A \dot\cup B, E)$ can be solved in $O(T + \sqrt{n}m \log nC)$ time and $O(S + m)$ space using arithmetic on numbers of size polynomial in n only. Here, T and S are the time and space bounds of an algorithm to sequentially enumerate the sets $\mathcal{E}_i$, $1 \le i \le \lceil \log C \rceil$, of all edges having a one in the i-th bit of their weight.*

3 An Application to Rank-Optimal Matchings

Let $G = (V, E), V = A \dot\cup B$ where $|V| = n$ and $|E| = m$ be a bipartite graph and let $E = E_1 \dot\cup E_2 \dot\cup \ldots \dot\cup E_r$, $r \le m$, be a partition of the edge set. We call the edges in class E_i the edges of rank i. For a matching M in G, let $s_i(M) = |M \cap E_i|$ be the number of edges in M of rank i. We call a matching:

- *rank-maximal*, if it maximizes the vector $(s_1, s_2, \ldots, s_r)$,
- *maximum cardinality rank-maximal*, if it maximizes the vector $(\sum_i s_i, s_1, s_2, \ldots, s_r)$, and
- *fair*: if it maximizes $(\sum_i s_i, -s_r, -s_{r-1}, \ldots, -s_1)$.

We show how to reduce all three versions to weighted matching problems, where the binary representation of the edge weights has a very simple structure. The reductions involve weights as large as n^r.

The *rank-maximal* matching problem has been previously studied in [5], where two algorithms were presented. The first with $O(r^* \sqrt{n}m)$ running time and $O(m)$ space and the second with $O(r^* nm)$ running time and the same space. Here r^* is the maximal rank used in any optimal solution for a particular instance. We present $O(r^* \sqrt{n}m \log n)$ algorithms for all three problems. No polynomial algorithm was known for the maximum cardinality rank-maximal and the fair matching problem; in particular, the $O(r^* \sqrt{n}m)$ algorithm from [5] does not seem to generalize.

Rank-maximal matchings Given an edge of rank i we assign to it a weight of $2^{\lceil \log n \rceil \cdot (r-i)}$. Then $C = 2^{\lceil \log n \rceil \cdot (r-1)}$ and the sets $\mathcal{E}_j$ for $1 \leq j \leq \lceil \log C \rceil$ are disjoint. This allows us to enumerate them efficiently. The enumeration algorithm maintains a list of buckets, where bucket i contains the edges which have a 1 in their i-th bit. Since only r such buckets are required the space requirement is $O(m + r)$. The total running time of the algorithm to compute a rank-maximal matching is therefore $O(r\sqrt{n}m \log n)$ with space requirement $O(m)$ since $r \leq m$.

In this time bound, we can replace r by r^* in the following way. Let $\phi(i) = (i-1)\lceil \log n \rceil + 1$ for $1 \leq i \leq r$ be the phase that we change the weight of rank i edges from 0 to 1. Instead of setting only the weight of i-ranked edges to 1 we set the weight of all edges with rank $\geq i$ to 1. We then execute $O(\log n)$ phases where we only scale by 2 every time (not adding anything on the edge weights). The resulting matching will be optimal w.r.t the edge weights at phase $\phi(i)$ (due to integrality). If the resulting matching does not use any edge of rank $\geq i$, then we know that there exists a rank-maximal matching using only edges of rank $< i$. Otherwise, we reset the weights to their original values and restart phase $\phi(i)$. This way the total number of phases will be $O(r^* \log n)$ instead of $O(r \log n)$.

Theorem 2. *The* rank-maximal *matching problem can be solved on a bipartite graph in $O(r^*\sqrt{n}m \log n)$ time using space $O(m)$, where $r^* \leq r \leq m$ is the maximal rank of an edge used in an optimal solution.*

Maximum cardinality rank-maximal matchings We modify the previous reduction by adding a sufficient large weight to each edge weight. This way we can make sure than any maximum weight matching has maximum cardinality. The largest weight we might have is $2^{\lceil \log n \rceil (r-1)}$ and therefore adding a weight larger than $2n2^{\lceil \log n \rceil (r-1)}$ such as $2^{\lceil \log n \rceil (r-1) + \lceil \log n \rceil + 2} = 2^{\lceil \log n \rceil r + 2}$ is sufficient. This is true since any matching may contain less than n edges. The new edge weights do not increase the complexity of enumerating the sets $\mathcal{E}_i$, since we simply add every edge to a constant number of new sets.

Fair matchings The *fair* matching problem can be similarly solved by assigning a weight of $2^{\lceil \log n \rceil (i-1)}$ to edges of rank i and finding a minimum weight maximum cardinality matching.

Theorem 3. *The* maximum cardinality rank-maximal *and* fair *matching problems can be solved on a bipartite graph in time $O(r\sqrt{n}m \log n)$ using space $O(m)$.*

4 Minimum cost flow

In this section we deal with the *capacitated transshipment* version of the minimum cost flow problem. Let $G = (V, E)$ be a directed network with a cost $c : E \mapsto \mathbb{Z}$ and capacity $u : E \mapsto \mathbb{Z}_{\geq 0}$ associated with each edge. With each $v \in V$ a number $b(v)$ is associated; $\sum_{v \in V} b(v) = 0$. If $b(v) > 0$, v is a supply node, and if $b(v) < 0$, 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 original and artificial edges.

A *pseudoflow* is a function $x : E \mapsto \mathbb{Z}$ 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 by $imb_x(v) = \sum_{(w,v)\in E} x(w,v) + b(v)$. 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 $\pi : V \mapsto \mathbb{Z}$. For a potential function π, 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 π such that $c^\pi(e) \geq 0$ for all $e \in E(x)$. For a constant $\epsilon \geq 0$ a flow is ϵ-optimal if $c^\pi(e) \geq -\epsilon$ for all $e \in E(x)$ for some potential function π. Consider an ϵ-optimal flow x and any original edge e. If $c^\pi(e) < -\epsilon$, the residual capacity of e must be zero and hence e is saturated, i.e., $x(e) = u(e)$. If $c^\pi(e) > \epsilon$, we have $c^\pi(e^R) = -c^\pi(e) < -\epsilon$ 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. As in Section 2, we assume that all costs are divisible by $2^{1+\lceil \log n \rceil}$. This guarantees that a 1-optimal flow is optimal. We refer the reader to [1] for more details.

Goldberg and Tarjan [6, 7] gave a scaling algorithm for the capacitated transshipment problem. It scales the costs and runs in $O(\log(nC))$ phases. We assume that capacities, demands and supplies are polynomially bounded in n, but costs are arbitrary. We modify the algorithm of Goldberg and Tarjan so that all numbers handled by the algorithm are polynomially bounded. The main ideas are to maintain node potentials implicitly and to fix the flow across edges of large (positive or negative) reduced cost. Fixing flow or node potentials has been previously used in network problems in order to obtain strongly polynomial algorithms [1, page 397].

4.1 The algorithm

The algorithm works in phases; in phase i it computes a 1-optimal flow with respect to the cost function c_i. In phase 0 we set the cost of every edge to 0 and calculate any feasible flow x_0. The zero potential function proves the 1-optimality of x_0. We therefore set the reduced cost of any edge to zero.

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 $c_{i-1}^{\pi_{i-1}}$. We scale up by multiplying the potentials and the edge costs by 2 and adding ± 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:

$$\tilde{c}(v,w) = 2c_{i-1}^{\pi_{i-1}}(v,w) + (1 \textit{ if } e \in \mathcal{E}_i \textit{ else } 0) \times \textit{sign}(e) \tag{2}$$

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

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, Equation (2) gives us $\tilde{c}(e) \geq -3$, i.e, x_{i-1} is 3-optimal with respect to the edge costs $\tilde{c}$ and the constant zero potential function. We use a variant of the algorithm of Goldberg and Tarjan [6, 7] to transform the 3-optimal flow into a 1-optimal flow.

Lemma 3. *Given the cost function $\tilde{c}$, a 3-optimal flow x_{i-1}, in time $O(nm \log(n^2/m))$ one can compute a potential function π and a flow x_i which is 1-optimal with respect to the reduced costs $c^\pi(u,v) = \tilde{c}(u,v) + \pi(u) - \pi(v)$. Potentials are only decreased during the computation and $\pi(v) \geq -dn$ for some constant d and all v.*

At the end of the phase, we implicitly set $\pi_i(v) = 2\pi_{i-1}(v) + \pi(v)$ for all v and explicitly recompute reduced costs as $c_i^{\pi_i}(u,v) = \tilde{c}(u,v) + \pi(u) - \pi(v)$. Thus all updates are by polynomially bounded numbers. We next show that edges of large reduced cost can be discarded; more precisely, we can fix the flow to either zero or the capacity.

We say that the reduced cost of an edge e is *large positive* from phase i on, if $c^\pi(e) > 1$ at the end of every phase after phase i, and is *large negative* from phase i on, if $c^\pi(e) < -1$ at the end of every phase after phase 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 4. *Let $f(n) = dn$ be the maximum potential change (decrease) of any vertex during any phase and let e be any edge. If $\tilde{c}_i(e) > 2f(n)+1$ at the beginning of phase i, e is large positive from phase i on, and if $\tilde{c}_i(e) < -2f(n) - 1$ at the beginning of phase i, e is large negative from phase i on.*

Proof. Assume $\tilde{c}_i(e) > 2f(n)+1$ at the beginning of some phase i. Then $c_i^{\pi_i}(e) > f(n) + 1$ since the reduced cost can change by at most $f(n)$. Therefore at the beginning of the next phase $\tilde{c}_{i+1} = 2c_i^{\pi_i}(v,w) + (1 \textit{ if } e \in \mathcal{E}_{i+1} \textit{ else } 0) \times \textit{sign}(e) > 2f(n) + 2 - 1 = 2f(n) + 1$. Thus e is large positive from phase i on.

Assume next that $\tilde{c}_i(e) < -2f(n) - 1$ at the beginning of some phase i. Then $c_i^{\pi_i}(e) < -f(n) - 1$ since the reduced cost can change by at most $f(n)$. Therefore at the beginning of the next phase $\tilde{c}_{i+1} = 2c_i^{\pi_i}(v,w) + (1 \textit{ if } e \in \mathcal{E}_{i+1} \textit{ else } 0) \times \textit{sign}(e) < -2f(n) - 2 + 1 = -2f(n) - 1$. Thus e is large negative from phase i on.

The Lemma above 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. In this way, all reduced costs stay in $O(n)$.

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

5 An Application to Rank-Optimal b-Matchings

The *rank-maximal* matchings problem can be generalized by introducing capacities on the vertices. We are given a function $q : V \mapsto \mathbb{Z}_{>0}$ denoting the capacity of each vertex. Each vertex $v \in V$ is allowed to be matched with at most $q(v)$ edges. Without loss of generality we may assume that $q(v) \le deg(v)$, where $deg(v)$ is the degree of $v \in G$.

We can transform such an instance into a minimum-cost flow problem. We add two additional vertices s and t. For each vertex $v \in A$ we add an edge (s, v) with capacity $q(v)$ and cost zero. For each vertex $v \in B$ we add an edge (v, t) again with capacity $q(v)$ and cost zero. Every edge (v, w) where $v \in A$ and $w \in B$ has capacity one and is directed from A to B. The resulting instance has a trivial upper bound on the maximum s-t flow of $n^2/4$. We need to assign costs to the edges in order for no collection of rank $\ge i + 1$ edges to overcome the cost of a rank i edge.

An edge of rank i will be assigned a cost of $-2^{2\lceil \log n \rceil (r-i)}$. In order to search for a circulation we also add an edge from t to s with cost zero and capacity $> n^2/4$. We set the supplies of all vertices to zero and search for a minimum-cost circulation with the algorithm presented in Section 4. The resulting circulation will correspond to a rank-maximal b-matching.

The sets $\mathcal{E}_j$ for $1 \le j \le 2\lceil \log n \rceil (r-1)$, are again disjoint and therefore can be enumerated efficiently using linear space. The resulting algorithm has running time $O(rnm \log(n^2/m) \log n)$ using space $O(m + r) = O(m)$ since $r \le m$.

We can again replace r by r^* in the following way. At the beginning of phase $\phi(i) = 2(i-1)\lceil \log n \rceil + 1$ for some $1 \le i \le r$ instead of setting only the weights of i-ranked edges to -1 we set the weight of all edges with rank $\ge i$ to -1. Now, we execute $O(\log n)$ more phases where we simply scale by 2 (not adding anything to the edge weights). The resulting flow will be optimal w.r.t the weights at phase $\phi(i)$ (due to integrality). If the resulting matching does not use any edges of rank $\ge i$, then we know that there exists a rank-maximal b-matching using only edges of rank $< i$. Otherwise, we reset the edges to their original weights and restart phase $\phi(i)$. The above imply that we require $O(r^* \log n)$ phases instead of $O(r \log n)$.

Theorem 5. *The* rank-maximal *b-matching problem can be solved on a bipartite graph in $O(r^* nm \log(n^2/m) \log n)$ time using space $O(m)$, where $r^* \le r \le m$ is the maximal rank used in the optimal solution.*

We can also solve the *maximum cardinality rank-maximal* b-matchings since we can find a minimum cost maximum s-t flow. In order to do so, we do not introduce the edge (t, s) and we set the demand of s to $|f|$ and t to $-|f|$, where f is a maximum s-t flow. Finally we can solve in the same way the *fair* matching problem with capacities, by assigning weights of $2^{2\lceil \log n \rceil (i-1)}$ on rank i edges and finding a maximum s-t flow of minimum cost.

Theorem 6. *The* maximum cardinality rank-maximal *and* fair *b-matching problems can be solved on a bipartite graph in $O(rnm \log (n^2/m) \log n)$ using space $O(m)$.*

6 Conclusions

This paper was motivated by studying *rank-optimal* matching problems. These problems reduce to weighted matching problems where the magnitude of the weights can be as large as $n^{O(m)}$. Therefore, a direct application of the fastest known network algorithms does not result on efficient algorithms.

By exploiting the fact that on the primal-dual algorithms there is no need to explicitly maintain the potential function, and by "pricing out" edges whose reduced cost has become large we obtained an algorithm for the maximum weight matching problem with running time $O(T + \sqrt{n}m \log nC)$ and space $O(S + m)$. Here T and S are the time and space respectively of an algorithm to sequentially enumerate the sets $\mathcal{E}_i$, $1 \leq i \leq \lceil \log C \rceil$ of all edges having a one in the i-th bit of their weight. These running times assume only that arithmetic on $O(\log n)$ bits take constant time. We also obtained a similar improvement for the capacitated transshipment problem with polynomially bounded demands.

Using the above algorithms we presented efficient algorithms for the *rank-optimal* matching problems with running time $O(r\sqrt{n}m \log n)$ and space $O(m)$ and for the *rank-optimal* b-matching problems with running time $O(rnm \log (n^2/m) \log n)$ and space $O(m)$. In some of these problems r can be replaced by $r^* \leq r \leq m$, the maximal rank for a particular instance used on an optimal solution.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
2. Gabow, H.N., Tarjan, R.: Faster scaling algorithms for network problems. SIAM Journal of Computing **18** (1989) 1013–1036
3. Bertsekas, D.P.: A distributed algorithm for the assignment problem. Laboratory for Information and Decision Sciences, Mass. Inst. of Technology, Cambridge, MA, Unpublished working paper (1979)
4. Bertsekas, D.P.: Distributed asynchronous relaxation methods for linear network flow problems. Technical Report LIDS Report P-1606, Mass. Inst. of Technology, Cambridge, MA (1986) preliminary version in Proc. 25th IEEE Conference of Decision and Control, December 1986.
5. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2004) 68–75

6. Goldberg, A., Tarjan, R.: Solving minimum-cost flow problems by successive approximation. In: Proceedings of the nineteenth annual ACM conference on Theory of computing, ACM Press (1987) 7–18
7. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by successive approximation. Math. Oper. Res. **15** (1990) 430–466