# A Generalized Approximation Framework for Fractional Network Flow and Packing Problems*

Michael Holzhauser        Sven O. Krumke

University of Kaiserslautern, Department of Mathematics

We generalize the fractional packing framework of Garg and Koenemann [17] to the case of linear fractional packing problems over polyhedral cones. More precisely, we provide approximation algorithms for problems of the form $\max\{c^\mathsf{T} x : Ax \leqslant b, x \in C\}$, where the matrix $A$ contains no negative entries and $C$ is a cone that is generated by a finite set $S$ of non-negative vectors. While the cone is allowed to require an exponential-sized representation, we assume that we can access it via one of three types of oracles. For each of these oracles, we present positive results for the approximability of the packing problem. In contrast to other frameworks, the presented one allows the use of arbitrary linear objective functions and can be applied to a large class of packing problems without much effort. In particular, our framework instantly allows to derive fast and simple fully polynomial-time approximation algorithms (FPTASs) for a large set of network flow problems, such as budget-constrained versions of traditional network flows, multicommodity flows, or generalized flows. Some of these FPTASs represent the first ones of their kind, while others match existing results but offer a much simpler proof.

## 1 Introduction

In a fractional linear packing problem, one seeks to find a solution to the problem $\max\{c^\mathsf{T} x : Ax \leqslant b, x \geqslant 0\}$, where the matrix $A \in \mathbb{N}_{\geqslant 0}^{m \times n}$ contains no negative entries and, without loss of generality, the vectors $c \in \mathbb{N}_{>0}^n$ and $b \in \mathbb{N}_{>0}^m$ have positive entries. Many problems can be formulated as packing problems, possibly the most intuitive being the fractional knapsack problem (cf. [27]). More than this, many network flow problems can be seen as fractional packing problems if one allows exponential sized representations. For example, the traditional maximum flow problem can be seen as the problem of packing flows on $s$-$t$-paths without violating the capacities of the edges.

A large number of authors presented approximation frameworks for such fractional packing problems, including Plotkin et al. [32], Grigoriadis and Khachiyan [18, 19], Young [38], and Bienstock

---

and Iyengar [3] (cf. [2, 17] for an overview of these results). One of the most powerful frameworks among these was developed by Garg and Koenemann [17], who have shown that a $(1 - \varepsilon)$-approximate solution to a general fractional packing problem of the above form can be computed efficiently *provided* we are able to "handle" the dual problem appropriately. More precisely, in the dual formulation $\min\{b^\top y : A^\top y \geqslant c, y \geqslant 0\}$, we need to be able to determine a *most violated dual constraint* efficiently: For some given infeasible solution $y$ to the dual, we need to find a dual constraint[1] $(A^\top)_{l\cdot} y \geqslant c_l$ that minimizes the value $\frac{(A^\top)_{l\cdot} y}{c_l}$ among all dual constraints with a positive right-hand side value. This constraint reveals the largest degree of violation. Since the result may even hold if the number of variables in the primal formulation is of exponential size, the authors were able to provide efficient FPTASs for network flow problems such as multicommodity flow problems by using (exponential-sized) path-based formulations of the corresponding problems (cf. Garg and Koenemann [17]). Fleischer [14] later showed that it suffices to determine an *approximately most violated dual constraint*, whose fraction $\frac{(A^\top)_{l\cdot} y}{c_l}$ may be up to a factor $1 + \varepsilon$ away from the largest violation.

In this paper, we generalize the result of Garg and Koenemann [17] to the case of packing problems over polyhedral cones. More precisely, we are interested in approximate solutions to problems of the form

$$\max c^\top x \tag{1a}$$

$$\text{s.t. } Ax \leqslant b, \tag{1b}$$

$$x \in C, \tag{1c}$$

where the matrix $A \in \mathbb{N}_{\geqslant 0}^{m \times n}$ contains no negative numbers and where the cone $C$ is finitely generated by a set $S$ of non-negative vectors. While the vector $b \in \mathbb{N}_{>0}^m$ is still assumed to contain positive entries (without loss of generality), we allow the entries of the vector $c \in \mathbb{N}^n$ to have arbitrary signs. We will thereby combine a large set of well-known techniques such as the fractional packing framework of Garg and Koenemann [17], the extension of Fleischer [14], the parametric search technique of Megiddo [29], geometric-mean binary search due to Hassin [21], and transformation strategies for fractional objectives as described in Lawler [28].

The chosen formulation is motivated by the following observation: Network flow problems often allow a characterization by some kind of *flow decomposition*, i.e., each feasible flow is representable by the sum of flows on much simpler structures, which we call *basic components* in the following. Viewed from the other side, we can express each such feasible flow as a conic combination of flows on these basic components. Hence, if $C$ describes the cone that is generated by flows on basic components, we can express structural properties of each flow by the containment in the cone. What usually remains are packing constraints that bound the total flow on each edge, the flow that leaves some node, or the overall costs of the flow. As it will be shown, most common network flow problems can be modeled in such a way.

One major advantage of the presented framework is that we do not assume the cone $C$ or the set $S$ that generates it to be given explicitly. Instead, we only assume that we have some kind of oracle access

---

[1] We use the notation $B_{l\cdot}$ for a matrix $B$ to denote the $l$-th row of the matrix.

to the cone, which allows us to derive polynomial-time algorithms for problems that require cones with an exponential number of extreme rays. In addition to this benefit, our framework provides the following advantages:

- ▷ The framework allows to derive fast and simple combinatorial FPTASs for a large class of packing problems and network flow problems. In many cases, we are even able to derive first strongly polynomial-time FPTASs.

- ▷ As our problem is formulated as a packing problem, the addition of further budget-constraints does not influence the applicability of the procedure, whereas such constraints usually make the design of exact algorithms significantly harder.

- ▷ The application of the framework only requires two "ingredients", namely the existence of some kind of flow decomposition theorem and a decent amount of control over the resulting basic components.

- ▷ As the framework is based on the approximation scheme of Garg and Koenemann [17] in its core, it works without considering some kind of residual network in case of network flow problems. As a consequence, the framework can be applied to problems that do not offer a concept of residual networks. One example is the maximum flow problem in generalized processing networks that will be discussed later. Moreover, it maintains properties of the underlying network such as cycle freeness or signs of costs.

- ▷ In contrast to the framework of Garg and Koenemann [17], our formulation allows to stick to the natural edge-based formulations of the corresponding network flow problems and does not require an explicit reformulation of the problem as a packing problem.

- ▷ Moreover, the presented framework is the first application of the procedure of Garg and Koenemann [17] that natively supports the use of arbitrary linear objective functions, which allows the application to minimum cost flow problems. To the best of our knowledge, all prior applications instead transformed the objective functions into budget-constraints and searched for the optimal budget, which requires the restriction to positive costs and which results in weakly polynomial running times (cf. [17, 14, 13]).

The paper is structured as follows: In Section 2, we briefly introduce the concepts that are inherent to our framework such as the procedure of Garg and Koenemann [17] and Megiddo's parametric search technique [29, 30]. In Section 3, we reformulate the given problem (1) as a packing problem and identify a subproblem that needs to be solved in each iteration of the algorithm. Moreover, we introduce three oracle types that will be investigated in the rest of the paper: a *minimizing oracle* returning a cost-minimal vector in the ground set, a *sign oracle* only returning a vector with the same sign as a cost minimal vector, and a *separation oracle* either returning a vector with negative costs or stating that there is no such vector. Based on these considerations, we describe the general procedure in Section 4 and show that we can approximate problem (1) efficiently if we are able to find a sufficiently good initial lower bound on the most violated dual constraint. In Section 5, we provide both weakly polynomial-time and strongly polynomial-time approaches to find such a lower bound. Finally, we apply our framework to a large class of network flow and packing problems in Section 6, including the maximum/minimum cost flow problem, generalized minimum cost flow problem, and the maximum/minimum cost flow problem in processing networks as well as budget-constrained versions

of these. Moreover, we apply our framework to the maximum (weighted) spanning tree packing problem and the maximum (weighted) matroid base packing problem as examples of "pure" combinatorial problems. In Table 1, we give an overview of the results that will be derived in Section 6.

| Problem | Previous best FPTAS | Our FPTAS |
|---|---|---|
| Budget-Constrained Maximum Flow Problem | — | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (m + n \log n)\right)$ |
| Budget-Constrained Minimum Cost Flow Problem | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot (nm^2 + n^3 m))\right)$ [22] | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot nm^2\right)$ |
| Budget-Constrained Minimum Cost Generalized Flow Problem | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot nm^2 \cdot (\log \frac{1}{\varepsilon} + \log \log M)\right)$ (positive costs) [13] | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot n^2 m^2\right)$ (arbitrary costs) |
| Maximum Flows in Generalized Processing Networks | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right)$ [24] | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right)$ (simpler proof) |
| Minimum Cost Flows in Generalized Processing Networks | — | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right)$ |
| Maximum Concurrent Flow Problem | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot (m^2 + kn)\right)$ [25] | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2 \cdot \min\{k, n\}\right)$ (simpler proof) |
| Maximum Weighted Multicommodity Flow Problem | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2 \cdot \min\{\log C, k\}\right)$ [14] | $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2\right)$ |
| Maximum Spanning Tree Packing Problem | $\mathcal{O}\left(n^3 m \log(n^2/m)\right)$ (exact algorithm) [15] | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot \alpha(m, n)\right)$ |
| Maximum Weighted Spanning Tree Packing Problem | — | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot \alpha(m, n)\right)$ |
| Maximum Matroid Base Packing Problem | $\mathcal{O}\left(m^7 \cdot F(m)\right)$ (exact algorithm) [33, p. 734] | $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot (F(m) + \log m)\right)$ |
| Maximum Weighted Matroid Base Packing Problem | — | $\widetilde{\mathcal{O}}\left(m \cdot F(m) \cdot \left(\frac{1}{\varepsilon^2} \cdot m + \log \log M\right)\right)$ |

**Table 1:** A summary of our results. Here, $m$ and $n$ denote the number of edges and nodes, respectively. The variable $k$ denotes the number of commodities and $C$ denotes the largest ratio of any two weights of commodities. $M$ is the largest integer given in the input. The notation $\widetilde{\mathcal{O}}(\cdot)$ ignores poly-logarithmic factors in $m$, so $\widetilde{\mathcal{O}}(f(n, m)) = \mathcal{O}(f(n, m) \cdot \log^q m)$ for some constant $q$. $\alpha(m, n)$ denotes the inverse Ackermann function. $F(m)$ is the running time of a independence testing oracle for a given matroid with $m$ elements in its ground set.

## 2 Preliminaries

### 2.1 Approximation Algorithms

An algorithm $A$ is called a (polynomial-time) *approximation algorithm with performance guarantee* $\alpha \in [1, \infty)$ or simply an $\alpha$–*approximation* for a maximization problem $\Pi$ with objective function $c$ if, for each instance $I$ of $\Pi$ with optimum solution $x^*$, it computes a feasible solution $x$ with objective value

4

$c(x) \geqslant \frac{1}{\alpha}c(x^*)$ in polynomial time. An algorithm $A$ that receives as input an instance $I \in \Pi$ and a real number $\varepsilon \in (0, 1)$ is called a *polynomial-time approximation scheme (PTAS)* if, on input $(I, \varepsilon)$, it computes a feasible solution $x$ with objective value $c(x) \geqslant (1 - \varepsilon) \cdot c(x^*)$ with a running-time that is polynomial in the encoding size $|I|$ of $I$. If this running-time is additionally polynomial in $\frac{1}{\varepsilon}$, the algorithm is called a *fully polynomial-time approximation scheme (FPTAS)*.

## 2.2 Garg and Koenemann's Fractional Packing Framework

Consider a fractional packing problem of the form $\max\{c^\mathsf{T}x : Ax \leqslant b, x \geqslant 0\}$ with non-negative entries in the matrix $A \in \mathbb{N}_{\geqslant 0}^{m \times n}$. For example, we can model the maximum flow problem in such a way by interpreting the variables as flows on $s$-$t$-paths and requiring that the sum of flows on all paths that share some specific edge $e$ is bounded by the edge's capacity. Hence, the vector $b$ corresponds to the capacities of the edges and the vector $c$ equals the all-one vector. Note that we need to stick to the path-based formulation of the problem since we are not allowed to introduce flow conservation constraints as they require negative coefficients. The dual formulation of the general primal problem is given as $\min\{b^\mathsf{T}y : A^\mathsf{T}y \geqslant c, y \geqslant 0\}$. In the example, the dual problem is to find small edge-lengths such that each path has length at least one. Although both the primal and the dual formulation of the problem are of exponential size in general, the fractional packing framework of Garg and Koenemann [17] allows us to obtain approximate solutions for the primal problem by using these formulations only implicitly, which will be shown in the following.

Suppose that we want to find an $(1 - \varepsilon)$-approximate solution for the primal problem with $\varepsilon \in (0, 1)$ and let $\varepsilon' := \frac{\varepsilon}{2}$. The procedure described in [17] starts with the feasible primal solution $x = 0$ and the infeasible dual solution given by $y_i := \frac{\delta}{b_i} > 0$ for each $i \in \{1, \dots, m\}$, where $\delta := \frac{(1+\varepsilon')}{((1+\varepsilon')m)^{\frac{1}{\varepsilon'}}}$. In each step of the algorithm, the *most violated dual constraint* is determined based on the current dual solution $y$, i.e., we determine a row $j$ in the dual formulation that minimizes $\frac{(A^\mathsf{T})_j \cdot y}{c_j}$ among all rows with negative right-hand side value. For example, although there are exponentially many constraints in the dual formulation of the maximum flow problem, we can find the most violated constraint in $\mathcal{O}(m + n \log n)$ time by computing a shortest $s$-$t$-path with edge lengths given by the dual vector $y$. We then increase $x_j$ by the (in terms of the primal problem) maximum allowed value $\nu := \min_{i \in \{1, \dots, m\}:A_{ij}>0} \frac{b_i}{A_{ij}}$ (i.e., in the example, we send $\nu$ units of flow on the shortest path *without* considering flow that has been sent in previous iterations), which will most likely make the primal solution infeasible. At the same time, each variable $y_i$ will be multiplied by a factor of $\left(1 + \varepsilon' \cdot \frac{\nu}{\frac{b_i}{A_{ij}}}\right)$. Intuitively, for the maximum flow problem, the "congested" edges will get "longer" over time and will, thus, be used less likely in future iterations, which somehow balances the flow among all paths.

The algorithm stops as soon as the dual solution fulfills $\sum_{i \in \{1, \dots, m\}} b_i \cdot y_i \geqslant 1$. As noted above, the primal solution will most likely be infeasible since, in each iteration, the primal variables are increased regardless of the previous values. However, Garg and Koenemann [17] show that we obtain a feasible primal solution by scaling down the solution $x$ by $\log_{1+\varepsilon'} \frac{1+\varepsilon'}{\delta}$ and that this solution is within a factor $(1 - 2\varepsilon') = (1 - \varepsilon)$ of the optimal solution. Moreover, they prove that the described

procedure terminates within $\frac{1}{\varepsilon'} \cdot m \cdot (1 + \log_{1+\varepsilon'} m) = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m\right)$ iterations. We refer to [17, 14, 13] for further details on the procedure.

In a follow-up publication, Fleischer [14] showed that it suffices to determine an *approximately most violated dual constraint* in each iteration of the problem: The claimed time bound and approximation guarantee continue to hold even if we choose a dual constraint $(A^\mathsf{T})_j.y \geqslant c_j$ only fulfilling $\frac{(A^\mathsf{T})_j.y}{c_j} \leqslant (1+\varepsilon) \cdot \min_{l \in \{1,\dots,n\}: c_l > 0} \frac{(A^\mathsf{T})_l.y}{c_l}$. We will make use of this observation in Section 4.

## 2.3 Megiddo's Parametric Search Technique

In Section 6, we make use of Megiddo's parametric search technique (cf. [29]). Since we will go into the very heart of the method, we will briefly describe its idea in the following. We refer the reader to [29] and [30] for further details on the method.

Assume that we want to solve an optimization problem $\Pi$ for which we already know an (exact) algorithm $A$ that solves the problem, but in which some of the input values are now *linear parametric values* that depend linearly on some real parameter $\lambda$. Moreover, suppose that an algorithm $C$ is known (in the following called *callback*) that is able to decide if some candidate value for $\lambda$ is smaller, larger, or equal to the value $\lambda^*$ that leads to an optimum solution to the underlying problem $\Pi$. The idea of the parametric search technique is to extend the input of $A$ from constants to affine functions depending on $\lambda$ and to simulate the execution of algorithm $A$ step by step. Note that each variable remains its affine structure as long as the algorithm only performs additions, subtractions, multiplications with constants, and comparisons. We call such an algorithm *strongly combinatorial* in the following. Throughout the execution, an interval $I$ is maintained that is known to contain the optimal value $\lambda^*$. As soon as the simulation reaches the comparisons of two linear parametric values, since both values depend linearly on $\lambda$, it either holds that one of the variables is always larger than or equal to the other one in $I$ (in which case the result of the comparison is independent from $\lambda$) or that there is a unique intersection point $\lambda'$. For this intersection point, we evaluate the callback $C$ in order to determine if $\lambda' < \lambda^*$, $\lambda' > \lambda^*$, or $\lambda' = \lambda^*$ and, thus, resolve the comparison, update the interval $I$, and continue the execution. Hence, as soon as the simulation of $A$ finishes, we have obtained an optimum solution to $\Pi$. The overall running-time is given by the running-time of $A$ times the running-time of $C$ and can be further improved using parallelization techniques described in [30]. We refer to [29, 30] for details on the parametric search technique. Further applications and extensions of parametric search techniques can moreover be found in [8, 34, 35].

# 3 Packing over Cones

In this section, we transform the problem (1) to a general (possibly exponential-sized) fractional packing problem in a first step and then reduce this problem to a more simple subproblem by incorporating the fractional packing framework of Garg and Koenemann [17]. Moreover, we introduce three types of oracles that enable us access to the cone $C$ and that will be used throughout the rest of the paper.

Let $S := \{x^{(1)}, \ldots, x^{(k)}\}$ denote a finite set of $k$ non-negative $n$-dimensional vectors $x^{(l)} \in \mathbb{R}^n_{\geqslant 0}$ with $x^{(l)} \neq 0$. The cone that is spanned by these vectors is given by

$$C := \left\{ x \in \mathbb{R}^n : x = \sum_{l=1}^{k} \alpha_l \cdot x^{(l)} \text{ with } \alpha_l \geqslant 0 \text{ for all } l \in \{1, \ldots, k\} \right\}. \tag{2}$$

The main result of this paper is that we are able to compute $(1 - \varepsilon)$-approximate solutions for the problem to maximize a linear function over the cone $C$ subject to a set of packing constraints under specific assumptions that will be investigated in the following. As we will see in Section 6, this extended framework is especially useful in the case of network flow problems for which some kind of flow decomposition theorem is known.

Note that we do *neither* require the set $S$ to be of polynomial size *nor* assume the set $S$ or the cone $C$ to be given explicitly. Instead, as it is common when dealing with implicitly given polyhedra, we only assume the cone to be *well-described*, which implies that it has an encoding length of at least $n + 1$ (cf. [20] for further details). Moreover, we make decisions over $S$ and $C$ via a given *oracle* $\mathcal{A}$ (to be specified later) that yields information about $S$ based on a given cost vector $d$. We assume that the running-time $T_{\mathcal{A}}$ of each oracle $\mathcal{A}$ fulfills $T_{\mathcal{A}} \in \Omega(n)$ since it would not be able to investigate each component of $d$ or return a vector $x \in C$ otherwise.

In the following, let $A \in \mathbb{N}^{m \times n}_{\geqslant 0}$ denote a constraint matrix with non-negative entries, $b \in \mathbb{N}^m_{>0}$ a positive right-hand side vector, and $c \in \mathbb{Z}^n$ a cost vector with arbitrary signs. Without loss of generality, we assume that at least one entry in each row and each column of $A$ is positive. Moreover, we define $N$ to be the number of non-zero entries contained in the matrix $A$.

As described above, the problem we want to approximate is given as follows:

$$\max c^\mathsf{T} x \tag{1a}$$

$$\text{s.t. } Ax \leqslant b, \tag{1b}$$

$$x \in C. \tag{1c}$$

Using the definition of the cone $C$ based on equation (2), we obtain the following equivalent formulation of the problem (1):

$$\max c^\mathsf{T} \sum_{l=1}^{k} \alpha_l \cdot x^{(l)}$$

$$\text{s.t. } A \left( \sum_{l=1}^{k} \alpha_l \cdot x^{(l)} \right) \leqslant b,$$

$$\alpha_l \geqslant 0 \qquad\qquad\qquad \text{for all } l \in \{1, \ldots, k\}.$$

In particular, we replaced the original variables $x$ by the weight vector $\alpha$ and, in doing so, incorporated the constraints of the cone. As noted above, this formulation might be of exponential size.

However, in the following, we will never need to state it explicitly but will derive results based on its implicit structure. By rearranging the objective function and the packing constraints, we obtain the following equivalent formulation of the problem:

$$\max \sum_{l=1}^{k} \alpha_l \cdot \left( c^T x^{(l)} \right) \tag{4a}$$

$$\text{s.t.} \sum_{l=1}^{k} \alpha_l \cdot \left( A_i . x^{(l)} \right) \leqslant b_i \qquad\qquad \text{for all } i \in \{1, \ldots, m\}, \tag{4b}$$

$$\alpha_l \geqslant 0 \qquad\qquad \text{for all } l \in \{1, \ldots, k\}. \tag{4c}$$

Clearly, we can neglect vectors $x^{(l)}$ for which $c^T x^{(l)} \leqslant 0$ since, without loss of generality, it holds that $\alpha_l = 0$ for each such $l$ in an optimal solution. Hence, for the moment, we restrict[2] our considerations on vectors $x^{(l)}$ with $c^T x^{(l)} > 0$ such that the primal problem (4) becomes in fact a fractional packing problem (again, possibly of exponential size). The dual formulation of this problem is given as follows:

$$\min \sum_{i=1}^{m} y_i \cdot b_i \tag{5a}$$

$$\text{s.t.} \sum_{i=1}^{m} y_i \cdot \left( A_i . x^{(l)} \right) \geqslant c^T x^{(l)} \qquad\qquad \text{for all } l \in \{1, \ldots, k\} \text{ with } c^T x^{(l)} > 0, \tag{5b}$$

$$y_i \geqslant 0 \qquad\qquad \text{for all } i \in \{1, \ldots, m\}. \tag{5c}$$

As it was shown in Section 2.2, we can apply the fractional packing framework of [17] to the original problem (1) *provided* we are able to determine the most violated dual constraint in equation (5b) efficiently. Hence, given a dual solution $y > 0$, we need to be able to solve the following subproblem in polynomial time:

$$\min_{\substack{l \in \{1,\ldots,k\} \\ c^T x^{(l)} > 0}} \frac{\sum_{i=1}^{m} y_i \cdot \left( A_i . x^{(l)} \right)}{c^T x^{(l)}} = \min_{\substack{l \in \{1,\ldots,k\} \\ c^T x^{(l)} > 0}} \frac{\sum_{i=1}^{m} y_i \cdot \sum_{j=1}^{n} A_{ij} \cdot x_j^{(l)}}{c^T x^{(l)}}$$

$$= \min_{\substack{l \in \{1,\ldots,k\} \\ c^T x^{(l)} > 0}} \frac{\sum_{j=1}^{n} x_j^{(l)} \cdot \sum_{i=1}^{m} y_i \cdot A_{ij}}{c^T x^{(l)}}.$$

With $a_j := \sum_{i=1}^{m} y_i \cdot A_{ij}$ for $j \in \{1, \ldots, n\}$ and $a = (a_1, \ldots, a_n)^T$, this subproblem reduces to

$$\min_{\substack{l \in \{1,\ldots,k\} \\ c^T x^{(l)} > 0}} \frac{a^T x^{(l)}}{c^T x^{(l)}}. \tag{6}$$

Note that the vector $a$ depends on $y$ and, thus, changes throughout the course of the procedure of Garg and Koenemann [17]. However, it always holds that $a_j > 0$ for each $j \in \{1, \ldots, n\}$ since $y_i > 0$

---

[2]We will see how we can "filter out" vectors $x^{(l)}$ with negative costs in the following sections.

for each $i \in \{1, \ldots, m\}$ throughout the procedure and since the matrix $A$ has at least one positive and no negative entry in each row as assumed above. Since $x^{(l)} \neq 0$ and $x^{(l)} \in \mathbb{R}_{\geq 0}^n$ for each $l \in \{1, \ldots, k\}$, this also yields that $a^\top x^{(l)} > 0$, so the minimum in equation (6) is always strictly positive.

**Observation 1:**
It always holds that $a_j > 0$ for each $j \in \{1, \ldots, n\}$. Moreover, $a^\top x^{(l)} > 0$ for each $x^{(l)} \in S$. ◁

Clearly, if the vectors in $S$ are given explicitly, we immediately obtain an FPTAS for the original problem (1) using the arguments given in Section 2.2. In the following, we discuss the more elaborate case that we can access the set $S$ and the cone $C$ only via given oracles. Throughout this paper, we investigate three kinds of oracles with decreasing strength. The most powerful oracle to be considered can be defined as follows:

**Definition 2** (Minimizing Oracle)**:**
For a given vector $d \in \mathbb{R}^n$, a *minimizing oracle* for the set $S$ returns a vector $x^{(l^*)} \in S$ that minimizes $d^\top x^{(l)}$ among all vectors $x^{(l)} \in S$. ◁

Clearly, the notion of minimizing oracles requires very powerful algorithms. For example, if $S$ is the set of unit-flows on simple cycles in a given graph $G$, a minimizing oracle would need to be able to determine a most negative simple cycle, which is $\mathcal{NP}$-complete in general (see Section 6.2). In many cases, it suffices to consider a much weaker type of oracle given as follows:

**Definition 3** (Sign Oracle)**:**
For a given vector $d \in \mathbb{R}^n$, a *sign oracle* for the set $S$ returns a vector $x^{(l)} \in S$ with[3] $\operatorname{sgn} d^\top x^{(l)} = \operatorname{sgn} d^\top x^{(l^*)}$, where $x^{(l^*)}$ minimizes $d^\top x^{(i)}$ among all vectors in $S$. ◁

Rather than determining a vector in $S$ with minimum cost, a sign oracle only returns *any* vector whose cost have the same sign as a minimum-cost vector, which may be much easier to find. In the example above, we can easily find a cycle with the same costs as a most negative cycle by computing a minimum mean cycle in $\mathcal{O}(nm)$ time (cf. [26] and Section 6.2). An even simpler kind of oracle is given as follows:

**Definition 4** (Separation Oracle)**:**
For a given vector $d \in \mathbb{R}^n$, a *separation oracle* for the set $S$ either states that $d^\top x^{(i)} \geq 0$ for all vectors $x^{(i)} \in S$ or returns a *certificate* $x^{(l)} \in S$ that fulfills $d^\top x^{(l)} < 0$. ◁

Clearly, the notion of separation oracles yields the least powerful yet most natural definition of an oracle. The name "separation oracle" is based on the fact that such an oracle can be seen as a traditional separation oracle for the *dual cone* $C^* := \{w \in \mathbb{R}^n : w^\top x \geq 0 \text{ for all } x \in C\}$ of the cone $C$ (cf. [20]).

Note that each minimizing oracle also induces a sign oracle and that each sign oracle induces a separation oracle, so the considered oracles have in fact decreasing strength. In particular, each approximation algorithm that is based on a sign oracle (separation oracle) is also valid for the case of a minimizing oracle (sign oracle).

---

[3]The *sign function* $\operatorname{sgn} \colon \mathbb{R} \mapsto \{-1, 0, 1\}$ returns $-1$, $0$, or $1$ depending on whether the argument is negative, zero, or positive, respectively.

For the special case of uniform costs $c^T x^{(l)}$ for all vectors $x^{(l)} \in S$, we get the first approximation result based on the procedure of Garg and Koenemann [17]:

**Theorem 5:**
Suppose that $c^T x^{(l)} = \hat{c}$ for all $x^{(l)} \in S$ and some constant $\hat{c} > 0$. Given a minimizing oracle $\mathcal{A}$ for $S$ running in $T_{\mathcal{A}}$ time, a $(1 - \varepsilon)$-approximate solution for the problem (1) can be computed in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (N + T_{\mathcal{A}})\right)$ time.

**Proof:** Since $c^T x^{(l)} = \hat{c}$ for each $x^{(l)} \in S$, the subproblem given in equation (6) reduces to the problem of finding a vector $x^{(l)}$ with minimum cost $\left(\frac{1}{\hat{c}} \cdot a\right)^T x^{(l)}$ among all vectors in $S$. Using the minimizing oracle, we can compute a minimizer for (6) in $\mathcal{O}(T_{\mathcal{A}})$ time based on the cost vector $d := \frac{1}{\hat{c}} \cdot a$. Note that this cost vector can be built in $\mathcal{O}(N)$ time as each entry $a_j$ of $a$ is defined to be $\sum_{i=1}^{m} y_i \cdot A_{ij}$, where each $y_i$ stems from the framework of Garg and Koenemann [17]. Consequently, we need look at each of the $N$ entries of $A$ once in order to build $d$. Hence, we are able to determine a most violated dual constraint of (5) in $\mathcal{O}(N + T_{\mathcal{A}})$ time, so the claim follows immediately from the arguments outlined in Section 2.2. $\square$

Note that the vector $d$ that is constructed in the above procedure is always positive in each component according to Observation 1. As a consequence, if for example we use the vector $d$ to denote the length of edges in a graph, we can use Dijkstra's (1959) algorithm to compute a shortest path. This will be used in Section 6.

In the following sections, we will focus on the more general cases in which the costs are no longer uniform and in which we may only have access to the cone via weaker types of oracles.

## 4 General Algorithm

Throughout this section, we assume that there is a separation oracle $\mathcal{A}$ for $S$ running in $T_{\mathcal{A}}$ time. Hence, the presented results are valid for the case of minimizing oracles and sign oracles as well. In the subsequent section, we will see where the different strengths of the oracles come into play.

The procedure of the upcoming algorithm is based on an idea introduced by Fleischer [14], which was originally developed for the maximum multicommodity flow problem: For $\lambda^*$ to denote the optimal value of the most violated dual constraint in equation (6), we let $\underline{\lambda}$ denote a positive lower bound for $\lambda^*$. We will show in Section 5 how we can find a good initial value for this lower bound efficiently. In each iteration of the procedure of Garg and Koenemann [17] as described in Section 2.2, we need to determine an approximately most violated dual constraint corresponding to some vector $x^{(j)} \in S$ fulfilling

$$\frac{a^T x^{(j)}}{c^T x^{(j)}} \leqslant (1 + \varepsilon) \cdot \lambda^* = (1 + \varepsilon) \cdot \min_{\substack{l \in \{1, \dots, k\} \\ c^T x^{(l)} > 0}} \frac{a^T x^{(l)}}{c^T x^{(l)}}.$$

For $\lambda \in \mathbb{R}$, let $d(\lambda) := a - \lambda c$ and $D(\lambda) := \min\{d(\lambda)^T x^{(l)} : l \in \{1, \dots, k\} \text{ with } c^T x^{(l)} > 0\}$. Similar to the minimum ratio cycle problem [28, 29, 30], we get the following characterization of the relation between the sign of $D(\lambda)$ and the sign of $\lambda^* - \lambda$:

**Lemma 6:**

For some given value of $\lambda \in \mathbb{R}$, it holds that $\text{sgn}(D(\lambda)) = \text{sgn}(\lambda^* - \lambda)$.

**Proof:** Let $L := \{l \in \{1, \ldots, k\} : c^T x^{(l)} > 0\}$. First, consider the case that $D(\lambda) > 0$. The claim follows by simple arguments:

$$D(\lambda) > 0 \iff d(\lambda)^T x^{(l)} > 0 \qquad \text{for all } l \in L$$

$$\iff (a - \lambda c)^T x^{(l)} > 0 \qquad \text{for all } l \in L$$

$$\iff \frac{a^T x^{(l)}}{c^T x^{(l)}} > \lambda \qquad \text{for all } l \in L$$

$$\iff \lambda^* > \lambda.$$

Conversely, if $D(\lambda) < 0$, we get the following equivalences by similar arguments:

$$D(\lambda) < 0 \iff d(\lambda)^T x^{(l)} < 0 \qquad \text{for some } l \in L$$

$$\iff (a - \lambda c)^T x^{(l)} < 0 \qquad \text{for some } l \in L$$

$$\iff \frac{a^T x^{(l)}}{c^T x^{(l)}} < \lambda \qquad \text{for some } l \in L$$

$$\iff \lambda^* < \lambda.$$

Finally, in the remaining case $D(\lambda) = 0$, it follows by continuity that $\lambda^* = \lambda$, which shows the claim. $\qquad \square$

Lemma 6 implies that $\lambda^*$ is the maximum value of $\lambda$ such that $D(\lambda) \geqslant 0$, i.e., such that $d(\lambda)^T x^{(l)} \geqslant 0$ for each $x^{(l)} \in S$ with $c^T x^{(l)} > 0$. In each iteration of our general procedure, we call the given separation oracle $\mathcal{A}$ with the vector $d((1 + \varepsilon)\underline{\lambda})$. We distinguish between the two possible outcomes of one such call:

**Case 1:** The oracle returns some certificate $x^{(l)} \in S$ with $d((1 + \varepsilon)\underline{\lambda})^T x^{(l)} < 0$. In this case, we get that

$$(a - (1 + \varepsilon) \cdot \underline{\lambda} \cdot c)^T x^{(l)} < 0$$

$$\iff \qquad\qquad a^T x^{(l)} < (1 + \varepsilon) \cdot \underline{\lambda} \cdot c^T x^{(l)}$$

$$\implies \qquad\qquad \frac{a^T x^{(l)}}{c^T x^{(l)}} < (1 + \varepsilon) \cdot \underline{\lambda}$$

$$\implies \qquad\qquad \frac{a^T x^{(l)}}{c^T x^{(l)}} < (1 + \varepsilon) \cdot \lambda^*.$$

The third inequality follows from the fact that $a^T x^{(l)} > 0$ (cf. Observation 1) and that $\underline{\lambda} > 0$ such that it also holds that $c^T x^{(l)} > 0$. The returned vector $x^{(l)}$ yields an approximately most

violated dual constraint. We use this dual constraint and continue the procedure of Garg and Koenemann [17]. Note that, during an iteration of the procedure, it holds that $c^\mathsf{T} x^{(j)}$ remains constant for each vector $x^{(j)} \in S$ (since it does not depend on the dual solution $y$) and that $a^\mathsf{T} x^{(l)}$ does not decrease (since both the entries in $x^{(l)}$ and the entries in $A$ are non-negative). Hence, $\underline{\lambda}$ continues to be a lower bound for the (possibly increased) new value $\lambda^*$ of (6).

**Case 2:** The oracle states that all vectors $x^{(l)}$ fulfill $d((1+\varepsilon)\underline{\lambda})^\mathsf{T} x^{(l)} \geqslant 0$. It now holds that

$$(a - (1+\varepsilon) \cdot \underline{\lambda} \cdot c)^\mathsf{T} x^{(l)} \geqslant 0 \qquad\qquad \forall x^{(l)} \in S$$

$$\Longleftrightarrow \qquad a^\mathsf{T} x^{(l)} \geqslant (1+\varepsilon) \cdot \underline{\lambda} \cdot c^\mathsf{T} x^{(l)} \qquad \forall x^{(l)} \in S$$

$$\Longleftrightarrow \qquad \frac{a^\mathsf{T} x^{(l)}}{c^\mathsf{T} x^{(l)}} \geqslant (1+\varepsilon) \cdot \underline{\lambda} \qquad\qquad \forall x^{(l)} \in S \text{ with } c^\mathsf{T} x^{(l)} > 0$$

$$\Longleftrightarrow \qquad \lambda^* \geqslant (1+\varepsilon) \cdot \underline{\lambda}.$$

In this case, we can update the lower bound $\underline{\lambda}$ to $(1+\varepsilon) \cdot \underline{\lambda}$ and continue.

Hence, in each iteration of the algorithm, we either proceed in the procedure of Garg and Koenemann [17] or we increase the lower bound by a factor of $1 + \varepsilon$. Again, we want to stress that $\underline{\lambda}$ continues to be a lower bound for $\lambda^*$ after an iteration of the above algorithm. The following theorem shows that this yields an efficient approximation algorithm for the problem (1) provided we are given a sufficiently good initial estimate for $\lambda^*$:

**Theorem 7:**
Given a separation oracle $\mathcal{A}$ for $S$ running in $T_{\mathcal{A}}$ time and given an initial lower bound $\underline{\lambda}$ for the initial value of $\lambda^*$ fulfilling $\underline{\lambda} \leqslant \lambda^* \leqslant m^{\frac{1}{\varepsilon}m} \cdot \underline{\lambda}$, a $(1-\varepsilon)$-approximate solution for problem (1) can be determined in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (N + T_{\mathcal{A}})\right)$ time.

**Proof:** The correctness of the procedure follows from the arguments outlined in Section 2.2, the preceding discussion, and the fact that the initial value of $\underline{\lambda}$ is a valid lower bound for $\lambda^*$.

In each step of the algorithm, we evaluate the given separation oracle and – based on its result – either perform one iteration of the procedure of Garg and Koenemann [17] and Fleischer [14] or update the lower bound $\underline{\lambda}$. As noted in Section 2.2, the former case occurs up to $\mathcal{O}(\frac{1}{\varepsilon^2} \cdot m \log m)$ times. In order to determine the number of updates to $\underline{\lambda}$, let $(\lambda^*)^{(k)}$, $\underline{\lambda}^{(k)}$, $y_i^{(k)}$ denote the values of the corresponding variables after the $k$-th iteration of the overall algorithm and let $\tau$ denote the number of iterations until the algorithm stops. Note that the procedure stops as soon as $\sum_{i=1}^{m} b_i \cdot y_i^{(k)} \geqslant 1$. Hence, after the $(\tau-1)$-th iteration, it holds that $y_i^{(\tau-1)} < \frac{1}{b_i}$ for each $i \in \{1, \ldots, m\}$. Since each variable $y_i^{(\tau-1)}$ will be increased by a factor of at most $1+\varepsilon$ in the final iteration, it holds that $y_i^{(\tau)} < (1+\varepsilon) \cdot \frac{1}{b_i}$. Since the initial value of each variable $y_i$ was set to $y_i^{(0)} := \frac{\delta}{b_i}$, every dual variable increases by a factor of at most $\frac{1+\varepsilon}{\delta}$ during the execution of the algorithm, so $y_i^{(\tau)} \leqslant \frac{1+\varepsilon}{\delta} \cdot y^{(0)}$. However, this also implies that $(\lambda^*)^{(\tau)} \leqslant \frac{1+\varepsilon}{\delta} \cdot (\lambda^*)^{(0)}$. Since the lower bound $\underline{\lambda}$ remains to be a lower bound after every step of the algorithm as discussed above, it holds that

$$\underline{\lambda}^{(\tau)} \leqslant (\lambda^*)^{(\tau)} \leqslant \frac{1+\varepsilon}{\delta} \cdot (\lambda^*)^{(0)} \leqslant \frac{1+\varepsilon}{\delta} \cdot m^{\frac{1}{\varepsilon}m} \cdot \underline{\lambda}^{(0)},$$

where the third inequality follows from the requirement that $(\lambda^*)^{(0)} \leqslant m^{\frac{1}{\varepsilon}m} \cdot \underline{\lambda}^{(0)}$. Since $\underline{\lambda}$ is increased by a factor of $1 + \varepsilon$ in each update step, we get that the number of such steps is bounded by

$$
\begin{aligned}
\log_{1+\varepsilon} \frac{\underline{\lambda}^{(\tau)}}{\underline{\lambda}^{(0)}} &\leqslant \log_{1+\varepsilon} \left( \frac{1+\varepsilon}{\delta} \cdot m^{\frac{1}{\varepsilon}m} \right) = \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} + \log_{1+\varepsilon} m^{\frac{1}{\varepsilon}m} \\[2mm]
&= \log_{1+\varepsilon} \frac{1+\varepsilon}{\frac{1+\varepsilon}{((1+\varepsilon)m)^{\frac{1}{\varepsilon}}}} + \frac{1}{\varepsilon} \cdot m \log_{1+\varepsilon} m \\[2mm]
&= \log_{1+\varepsilon} ((1+\varepsilon)m)^{\frac{1}{\varepsilon}} + \frac{1}{\varepsilon} \cdot m \log_{1+\varepsilon} m \\[2mm]
&= \frac{1}{\varepsilon} \cdot (1 + \log_{1+\varepsilon} m) + \frac{1}{\varepsilon} \cdot m \log_{1+\varepsilon} m \\[2mm]
&\in \mathcal{O} \left( \frac{1}{\varepsilon} \cdot m \log_{1+\varepsilon} m \right) = \mathcal{O} \left( \frac{1}{\varepsilon} \cdot m \cdot \frac{\log m}{\log(1+\varepsilon)} \right) = \mathcal{O} \left( \frac{1}{\varepsilon^2} \cdot m \log m \right)
\end{aligned}
$$

and, thus, matches the number of iterations of the procedure of Garg and Koenemann [17]. The claim then follows by the fact that, in each step of the algorithm, we need $\mathcal{O}(N)$ time to compute the entries of the vector $d((1+\varepsilon)\underline{\lambda})$ and $T_{\mathcal{A}}$ time to evaluate the oracle. $\qquad\square$

Note that the allowed deviation of the initial lower bound $\underline{\lambda}$ to $\lambda^*$ in Theorem 7 is chosen in a way such that the number of update steps to $\underline{\lambda}$ does not dominate the $\mathcal{O}(\frac{1}{\varepsilon^2} \cdot m \log m)$ steps of the overall procedure.

# 5 Determining a lower bound

The proof of Theorem 7 shows that the strongly polynomial number of oracle calls depends on the assumption that the initial value for the lower bound $\underline{\lambda}$ is not "too far away" from the real value $\lambda^*$ of the most violated dual constraint. In this section, we present a weakly polynomial-time and a strongly polynomial-time approach to find such a sufficiently good initial value.

## 5.1 Weakly Polynomial-Time Approach for Separation Oracles

We start by providing a general approach that is valid for all three types of oracles. The running time will depend (logarithmically) on the largest number given in the input, denoted by $M :=$ $\max\{(\max_i b_i), (\max_j c_j), (\max_{i,j} A_{i,j}), n, m\} \in \mathbb{N}$.

**Lemma 8:**
Suppose that we are given a separation oracle $\mathcal{A}$ running in $T_{\mathcal{A}}$ time. An initial lower bound $\underline{\lambda}$ for $\lambda^*$ fulfilling $\underline{\lambda} \leqslant \lambda^* \leqslant m^{\frac{1}{\varepsilon}m} \cdot \underline{\lambda}$ can be determined in weakly polynomial time $\mathcal{O}((T_{\mathcal{A}} + N) \cdot (\log \log M - (\log \frac{1}{\varepsilon} + \log m + \log \log m)))$.

**Proof:** Let $x^{(l)} \in S$ denote a vector with $\frac{a^\mathsf{T} x^{(l)}}{c^\mathsf{T} x^{(l)}} = \lambda^*$ that determines the minimum in equation (6). Using that $y_i := \frac{b_i}{\delta}$ for each $i \in \{1, \dots, m\}$ at the beginning of the procedure, we get that

$$\frac{a^\mathsf{T} x^{(l)}}{c^\mathsf{T} x^{(l)}} = \frac{\sum_{j=1}^n a_j \cdot x_j^{(l)}}{\sum_{j=1}^n c_j \cdot x_j^{(l)}} = \frac{\sum_{j=1}^n \left( \sum_{i=1}^m y_i \cdot A_{ij} \right) \cdot x_j^{(l)}}{\sum_{j=1}^n c_j \cdot x_j^{(l)}} = \frac{\sum_{j=1}^n \sum_{i=1}^m \frac{\delta}{b_i} \cdot A_{ij} \cdot x_j^{(l)}}{\sum_{j=1}^n c_j \cdot x_j^{(l)}}. \tag{7}$$

Without loss of generality, we can assume the separation oracle $\mathcal{A}$ to always return a vector $x^{(l)} \in S$ with $\max_{j \in \{1, \dots, n\}} x_j = 1$ (whenever it returns a vector at all): For each $x \in C$, it also holds that $\beta \cdot x \in C$ for some positive constant $\beta$. Hence, if the oracle does not fulfill the required property, we can wrap it into a new oracle $\mathcal{A}'$ which divides the vector returned by $\mathcal{A}$ by $\max_{j \in \{1, \dots, n\}} x_i > 0$. Using this fact in equation (7), we get the following lower and upper bound on $\lambda^*$:

$$\lambda^* \geqslant \frac{\delta}{M} \cdot \frac{\sum_{j=1}^n \left( \sum_{i=1}^m A_{ij} \right) \cdot x_j^{(l)}}{\sum_{j=1}^n M \cdot x_j^{(l)}} \geqslant \frac{\delta}{M} \cdot \frac{\sum_{j=1}^n 1 \cdot x_j^{(l)}}{n \cdot M} \geqslant \frac{\delta}{n \cdot M^2} \geqslant \frac{\delta}{M^3} =: \lambda_1$$

and

$$\lambda^* \leqslant \frac{\delta}{1} \cdot \frac{\sum_{j=1}^n \left( \sum_{i=1}^m A_{ij} \right) \cdot x_j^{(l)}}{\sum_{j=1}^n 1 \cdot x_j^{(l)}} \leqslant \delta \cdot \frac{\sum_{j=1}^n m \cdot M \cdot x_j^{(l)}}{1} \leqslant \delta \cdot nm \cdot M \leqslant \delta \cdot M^3 =: \lambda_2.$$

According to Lemma 6, each feasible lower bound $\underline{\lambda}$ for $\lambda^*$ is characterized by the fact that $D(\underline{\lambda}) \geqslant 0$, so an oracle call with the vector $d(\underline{\lambda})$ results in the answer that there are no vectors in $S$ with negative costs. Since $\underline{\lambda}$ is required to fulfill $\underline{\lambda} \leqslant \lambda^* \leqslant m^{\frac{1}{\varepsilon} m} \cdot \underline{\lambda}$, we only need to consider values for $\underline{\lambda}$ of the form $\lambda_1 \cdot (m^{\frac{1}{\varepsilon} m})^k$ in $[\lambda_1, \lambda_2]$ for integral values of $k$. Moreover, since the oracle returns a vector if and only if $\underline{\lambda} > \lambda^*$, we can perform a binary search on these values in order to find the best possible lower bound for $\lambda^*$. In total, we get the following number of iterations:

$$\mathcal{O}\left( \log \log_{m^{\frac{1}{\varepsilon} m}} \frac{\lambda_2}{\lambda_1} \right) = \mathcal{O}\left( \log \log_{m^{\frac{1}{\varepsilon} m}} \frac{\delta \cdot M^3}{\frac{\delta}{M^3}} \right) = \mathcal{O}\left( \log \log_{m^{\frac{1}{\varepsilon} m}} M \right)$$

$$= \mathcal{O}\left( \log \frac{\log M}{\log m^{\frac{1}{\varepsilon} m}} \right) = \mathcal{O}\left( \log \frac{\log M}{\frac{1}{\varepsilon} \cdot m \log m} \right)$$

$$= \mathcal{O}\left( \log \log M - \left( \log \frac{1}{\varepsilon} + \log m + \log \log m \right) \right).$$

In combination with the overhead of $N + T_{\mathcal{A}}$ to call the oracle (as in the proof of Theorem 7), we get the claimed time bound. $\qquad \square$

Note that the time bound given in Lemma 8 is in fact only weakly polynomial for very large values of $M$: The binary search only has an effect on the overall running time if the encoding length $\log M$ of $M$ fulfills $\log M \in \omega(\frac{1}{\varepsilon} \cdot m \log m)$, i.e., if $M$ is exponential in $\frac{1}{\varepsilon} \cdot m \log m$.

Theorem 7 in combination with Lemma 8 yields the following theorem:

**Theorem 9:**

Given a separation oracle $\mathcal{A}$ for $S$ running in $T_{\mathcal{A}}$ time, a $(1 - \varepsilon)$-approximate solution for problem (1) can be determined in weakly polynomial time $\mathcal{O}((T_{\mathcal{A}} + N) \cdot (\frac{1}{\varepsilon^2} \cdot m \log m + \log \log M - (\log \frac{1}{\varepsilon} + \log m + \log \log m)))$. $\qquad\square$

In particular, if the oracle $\mathcal{A}$ runs in polynomial time, we immediately obtain an FPTAS for problem (1) according to Theorem 9.

## 5.2 Strongly Polynomial-Time Approach for Sign Oracles

In the previous subsection, we introduced a method to determine an initial lower bound for $\lambda^*$ that is valid for each of the investigated types of oracles. However, although the general procedure that was described in Section 4 performs a strongly polynomial number of steps, the overall procedure would not yield a strongly polynomial FPTAS, in general, even if the oracle runs in strongly polynomial time due to the weakly polynomial overhead of the binary search. In this section, we present an alternative method for minimizing and sign oracles running in strongly polynomial time. In the subsequent subsection, we generalize the result to separation oracles.

According to Lemma 6, we can decide about the direction of the deviation between some candidate value $\lambda$ and the desired value $\lambda^*$, if we are able to determine the sign of $D(\lambda)$. Clearly, this task is strongly related to the definition a sign oracle for $S$. However, the value $D(\lambda)$ is defined to be the minimum of $d(\lambda)^\mathsf{T} x^{(l)}$ among all vectors $x^{(l)}$ that *additionally* fulfill $c^\mathsf{T} x^{(l)} > 0$ whereas the sign oracle is not required to consider only such vectors according to Definition 3. Nevertheless, as it will be shown in the following lemma, we can neglect this additional restriction when evaluating the sign oracle:

**Lemma 10:**

For any positive value of $\lambda$, it holds that $\mathrm{sgn}(D(\lambda)) = \mathrm{sgn}(d(\lambda)^\mathsf{T} x^{(l)})$ where $x^{(l)}$ is a vector returned by a sign oracle for $S$.

**Proof:** First consider the case that $\mathrm{sgn}(d(\lambda)^\mathsf{T} x^{(l)}) = -1$, i.e., that $d(\lambda)^\mathsf{T} x^{(l)} < 0$. Using the definition of $d(\lambda)$, we get that $(a - \lambda c)^\mathsf{T} x^{(l)} = a^\mathsf{T} x^{(l)} - \lambda \cdot c^\mathsf{T} x^{(l)} < 0$. Since both $a^\mathsf{T} x^{(l)} > 0$ according to Observation 1 and $\lambda > 0$, it must hold that $c^\mathsf{T} x^{(l)} > 0$ as well. Thus, we can conclude that $D(\lambda) \leqslant d(\lambda)^\mathsf{T} x^{(l)} < 0$.

Now consider the case that $\mathrm{sgn}(d(\lambda)^\mathsf{T} x^{(l)}) = 0$. According to Definition 3, it holds that there are no vectors $x^{(j)} \in S$ with $d(\lambda)^\mathsf{T} x^{(j)} < 0$, so $D(\lambda) \geqslant 0$. As in the previous case, we get that $(a - \lambda c)^\mathsf{T} x^{(l)} = a^\mathsf{T} x^{(l)} - \lambda \cdot c^\mathsf{T} x^{(l)} = 0$ if and only if $c^\mathsf{T} x^{(l)} > 0$ since both $a^\mathsf{T} x^{(l)} > 0$ and $\lambda > 0$. Hence, we also get that $D(\lambda) \leqslant d(\lambda)^\mathsf{T} x^{(l)} = 0$, so $D(\lambda) = 0$.

Finally, if $\mathrm{sgn}(d(\lambda)^\mathsf{T} x^{(l)}) = 1$, there are no vectors $x^{(i)} \in S$ with $d(\lambda)^\mathsf{T} x^{(i)} \leqslant 0$. Thus, it also holds that $D(\lambda) > 0$, which shows the claim. $\qquad\square$

Lemma 10 now allows us to determine a sufficiently good initial lower bound $\underline{\lambda}$. In fact, as it will be shown in the following lemma, we are even able to determine an *exact* most violated dual constraint in *each* iteration of the procedure:

**Lemma 11:**

Given a strongly combinatorial and strongly polynomial-time sign oracle $\mathcal{A}$ for S running in $T_{\mathcal{A}}$ time, a most violated dual constraint can be determined in $\mathcal{O}\left(N + T_{\mathcal{A}}^2\right)$ time.

**Proof:** Lemma 6 and Lemma 10 imply that $\lambda^*$ is the unique value for $\lambda$ for which the sign oracle returns a vector $x^{(l)} \in S$ with $d(\lambda)^\mathsf{T} x^{(l)} = 0$. In particular, the returned vector $x^{(l)}$ is a minimizer for (6). Hence, since the values $a_i$ can be computed in $\mathcal{O}(N)$ time, we are done if we are able to determine such a vector $x^{(l)}$ in $\mathcal{O}(T_{\mathcal{A}}^2)$ time.

Let $d(\lambda)$ be defined as above, where $\lambda$ is now treated as a *symbolic* value that is known to be contained in an interval I. Initially, we set I to $(0, +\infty)$ since the optimal value $\lambda^*$ is known to be strictly positive (cf. equation (6)). Note that the costs $(d(\lambda))_i = a_i - \lambda \cdot c_i$ fulfill the linear parametric value property. We simulate the execution of the sign oracle $\mathcal{A}$ at input $d(\lambda)$ using Megiddo's (1979) parametric search technique as described in Section 2.3. The underlying idea is to "direct" the control flow during the execution of $\mathcal{A}$ in a way such that it eventually returns the desired vector minimizing (6).

Whenever we need to resolve a comparison between two linear parametric values that intersect at some point $\lambda' \in I$, we call the sign oracle itself with the cost vector $d := d(\lambda')$ in order to determine the sign of $D(\lambda')$. If $D(\lambda') = 0$, we found a minimizer for equation (6) and are done. If $D(\lambda') < 0$ ($D(\lambda') > 0$), the candidate value $\lambda'$ for $\lambda^*$ was too large (too small) according to Lemma 6 and Lemma 10 such that we can update the interval I to $I \cap (-\infty, \lambda')$ $(I \cap (\lambda', +\infty))$, resolve the comparison, and continue the simulation of the oracle algorithm. After $\mathcal{O}(T_{\mathcal{A}})$ steps, the simulation terminates and returns a vector $x^{(l)} \in S$ that fulfills $d(\lambda^*)^\mathsf{T} x^{(l)} = 0$ for the desired value $\lambda^* \in I$. Hence, this vector yields a most violated constraint in (5b). Since the described simulation runs in $\mathcal{O}(T_{\mathcal{A}}^2)$ time, the claim follows. $\qquad\square$

Note that we actually still obtain a polynomial running-time of the above algorithm even if we do not assume the sign oracle to run in *strongly* polynomial time but only to run in *weakly* polynomial time. However, the running-time of the resulting algorithm might exceed the stated time bound since the candidate values $\lambda'$ that determine the input to the callback oracle are rational numbers whose representation might involve exponential-size numbers of the form $H^{T_{\mathcal{A}}}$ for some H with polynomial encoding length. Although the running-time of a weakly polynomial-time oracle algorithm depends only logarithmically on the size of these numbers, the running-time might still increase by a large (polynomial) factor.

Lemma 11 can be incorporated into the procedure of Garg and Koenemann [17] to obtain an FPTAS for problem (1) running in $\mathcal{O}(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (N + T_{\mathcal{A}}^2))$ time. However, it can also be used to find an initial lower bound $\underline{\lambda}$ for $\lambda^*$ (which, in fact, equals $\lambda^*$), which yields the following theorem in combination with Theorem 7:

**Theorem 12:**

Given a strongly combinatorial and strongly polynomial-time sign oracle $\mathcal{A}$ for S running in $T_{\mathcal{A}}$ time, there is a strongly polynomial FPTAS for the problem (1) running in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (N + T_{\mathcal{A}}) + T_{\mathcal{A}}^2\right)$ time. $\qquad\square$

## 5.3 Strongly polynomial-time approach for Separation Oracles

Although separation oracles are probably the most natural kind of oracle, they are also the weakest of the considered oracle types. The proof of Lemma 11 relies on the fact that we are able to decide if some candidate value $\lambda$ is too small, too large, or equal to the desired value. In the case of a separation oracle, however, the case that $d^\mathsf{T} x^{(i)} \geqslant 0$ for all vectors $x^{(i)} \in S$ does no longer include the information whether there is a vector $x^{(l)} \in S$ with $d^\mathsf{T} x^{(l)} = 0$ (in which case we have found the desired vector in the parametric search as described above) or if $d^\mathsf{T} x^{(i)} > 0$ for all $x^{(i)} \in S$. For example, if we come across a comparison of the form $a_0 + \lambda \cdot a_1 \leqslant b_0 + \lambda \cdot b_1$ during the simulation where $a_1 > b_1$, we are actually interested in the information whether or not the optimal value $\lambda^*$ fulfills $\lambda^* \leqslant \lambda' := \frac{b_0 - a_0}{a_1 - b_1}$. However, if we use the separation oracle with the cost vector $d(\lambda')$, we only obtain the information whether $\lambda^* < \lambda'$ (in case that the oracle returns a certificate) or if $\lambda^* \geqslant \lambda'$. Hence, in the latter case, the outcome of the comparison is not yet resolved since we need the additional information whether or not $\lambda^* = \lambda'$, so we cannot continue the simulation without any further ado. Nevertheless, as it will be shown in the following lemma, we can gather this additional information by a more sophisticated approach:

**Lemma 13:**
Given a strongly combinatorial and strongly polynomial-time separation oracle $\mathcal{A}$ for S running in $T_\mathcal{A}$ time, a most violated dual constraint can still be determined in $\mathcal{O}\left(N + T_\mathcal{A}^2\right)$ time.

**Proof:** The claim directly follows from Lemma 11 if we can show that we can extend the given separation oracle into a sign oracle for S. As in the proof of Lemma 11, we simulate the execution of the separation oracle using the parametric cost vector $d(\lambda) := a - \lambda c$. Assume that the execution halts at a comparison that needs to be resolved, resulting in a candidate value $\lambda'$ for the desired value $\lambda^*$. We invoke the separation oracle with the cost vector $d := d(\lambda')$. Clearly, if the oracle returns a certificate $x^{(l)}$ with $d^\mathsf{T} x^{(l)} < 0$, we can conclude that $D(\lambda') < 0$ such that the value $\lambda'$ was too large according to Lemma 6 and the result of the comparison is determined. Conversely, if the oracle states that $d^\mathsf{T} x^{(i)} \geqslant 0$ for all $x^{(i)} \in S$, we can conclude that $D(\lambda') \geqslant 0$. However, we may not yet be able to resolve the comparison since its result may rely on the additional information whether $D(\lambda') = 0$ or $D(\lambda') > 0$ as shown above. Nevertheless, we can extract this information by one additional call to the oracle as it will be shown in the following.

First suppose that $\lambda' = \lambda^*$. In this situation, it holds that $d(\lambda')^\mathsf{T} x^{(i)} \geqslant 0$ for all $x^{(i)} \in S$ and there is at least one vector $x^{(l)} \in S$ that fulfills $d(\lambda')^\mathsf{T} x^{(l)} = 0$. Since all the functions $f^{(i)}(\lambda) := d(\lambda)^\mathsf{T} x^{(i)} = a^\mathsf{T} x^{(i)} - \lambda \cdot c^\mathsf{T} x^{(i)}$ are linear functions of $\lambda$ with negative slope (in case that $c^\mathsf{T} x^{(i)} > 0$; otherwise, the function has no positive root at all), it holds that several functions $f^{(l)}$ evaluate to zero at $\lambda'$ while every other function attains its root at a larger value for $\lambda$ (cf. Figure 1a). Hence, for every larger value of $\lambda$, the separation oracle changes its outcome and returns a certificate. In particular, for a sufficiently small but positive value of $\delta$, the separation oracle called with the cost vector $d(\lambda' + \delta)$ returns a vector $x^{(l)} \in S$ with $d(\lambda' + \delta)x^{(l)} < 0$ that additionally fulfills $d(\lambda')^\mathsf{T} x^{(l)} = 0$ (so $x^{(l)}$ yields a most violated constraint in the overall procedure). Clearly, the value of $\delta$ must be small enough to guarantee that we do not reach the root of another function $f^{(i)}$ (i.e., smaller than the distance between the dashed and the dotted line in Figure 1a).
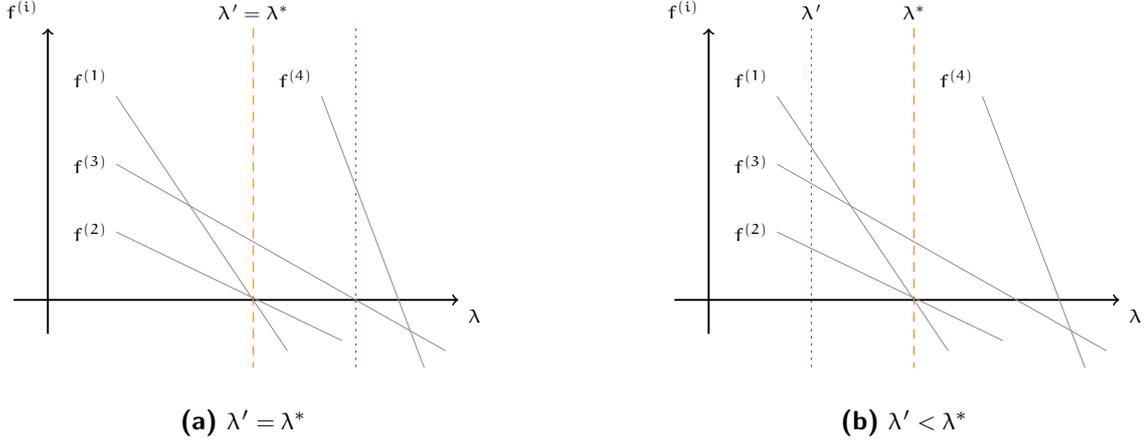
**(a)** $\lambda' = \lambda^*$          **(b)** $\lambda' < \lambda^*$

**Figure 1:** Illustration of the two cases that may occur during the simulation of the separation oracle in case that the separation oracle did not return a certificate when evaluated for a candidate value $\lambda'$.

Now suppose that $\lambda' < \lambda^*$ (cf. Figure 1b). In this case, for a sufficiently small but positive value of $\delta$, the separation oracle returns the *same* answer when called with the cost vector $d(\lambda' + \delta)$ as long as $\lambda' + \delta \leqslant \lambda^*$ (i.e., as long as $\delta$ is smaller than the distance between the dotted and the dashed line in Figure 1b). Consequently, in order to separate this case from the former case, it suffices to specify a value for $\delta$ that is smaller than the distance between any two roots of the functions that occur both in the instance and during the simulation of $\mathcal{A}$. We can then use a second call to the decision oracle in order to decide whether a candidate value $\lambda'$ is smaller than or equal to the optimal value $\lambda^*$.

First note that the root of each function $f^{(i)}$ is given by the rational number $\frac{a^T x^{(i)}}{c^T x^{(i)}}$. Since the coefficients $c_j$ are part of the instance I of the problem (1) and since the values $a_j = \sum_{i=1}^{m} y_i \cdot A_{ij}$ are generated within the framework of Garg and Koenemann [17], the encoding length of both values is polynomial in the problem size. Similarly, as noted in Section 3, we can assume that the encoding lengths of all vectors $x^{(i)}$ returned by the oracle are in $\Omega(n)$ and, since the oracle runs in polynomial time, polynomially bounded in the instance size. Consequently, there is some bound $M_f$ with polynomial encoding length such that the root of each function $f^{(i)}$ can be represented by a fraction $\frac{p_i}{q_i}$ with $p_i, q_i \in \mathbb{N}$ and $q_i \leqslant M_f$.

Now consider the root $-\frac{a_0 - b_0}{a_1 - b_1}$ of some function $g$ of the form $g(\lambda) := (a_0 - b_0) + \lambda \cdot (a_1 - b_1)$ that stems from a comparison of two linear parametric values of the forms $a_0 + \lambda \cdot a_1$ and $b_0 + \lambda \cdot b_1$. Assume that we are in the k-th step of the simulation. Since the oracle algorithm is strongly combinatorial, the values $a_0 + \lambda \cdot a_1$ and $b_0 + \lambda \cdot b_1$ result from one or more of the input values $d_j := a_j - \lambda \cdot c_j$ (which are the only linear parametric values at the beginning of the simulation) as well as a sequence of up to k additions or subtractions with other linear parametric values and multiplications with constants. Hence, since $k \in \mathcal{O}(T_{\mathcal{A}})$ and $\mathcal{A}$ runs in (strongly) polynomial time, there is a bound $M_g$ with polynomial encoding length such that the root $-\frac{a_0 - b_0}{a_1 - b_1}$ of each such function $g$ considered up to the k-th step of the simulation can be represented by a fraction of the form $\frac{p}{q}$ with $p, q \in \mathbb{N}$ and $q \leqslant M_g$.

Now let $\mu_1 = \frac{p_1}{q_1}$ and $\mu_2 = \frac{p_2}{q_2}$ with $\mu_1 \neq \mu_2$ denote the roots of two of the above functions of the

form $f^{(i)}$ or $g$. Since $q_1, q_2 \leqslant M_f \cdot M_g$, we get that

$$|\mu_1 - \mu_2| = \left| \frac{p_1}{q_1} - \frac{p_2}{q_2} \right| = \left| \frac{p_1 \cdot q_2 - p_2 \cdot q_1}{q_1 \cdot q_2} \right| \geqslant \frac{1}{M_f^2 \cdot M_g^2} =: \mu$$

Hence, choosing $\delta := \frac{\mu}{2}$, we are able to differentiate between the three cases $D(\lambda) < 0$, $D(\lambda) = 0$, and $D(\lambda) > 0$. Moreover, by returning *any*[4] vector in $S$ in the case of $D(\lambda) > 0$ and returning the certificate in every other case, the separation oracle is extended into a sign oracle and the correctness follows from the proof of Lemma 11. Note that the running time remains to be $\mathcal{O}\left(N + T_{\mathcal{A}}^2\right)$ (as in the case of a sign oracle in Lemma 11) since the encoding length of the number $\delta$ is polynomially bounded and since the oracle algorithm is assumed to run in strongly polynomial time. $\qquad\square$

Lemma 13 now yields one of the main results of this paper:

**Theorem 14:**

Given a strongly combinatorial and strongly polynomial-time sign oracle $\mathcal{A}$ for $S$ running in $T_{\mathcal{A}}$ time, there is a strongly polynomial FPTAS for the problem (1) running in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (N + T_{\mathcal{A}}) + T_{\mathcal{A}}^2\right)$ time. $\qquad\square$

# 6 Applications

In this section, we present several applications of the introduced framework. We will be able to derive new strongly polynomial-time FPTASs for several well-known network flow and packing problems and complement or even improve upon well-known results. All graphs considered in this section are assumed to be connected, such that the number of nodes $n$ fulfills $n \in \mathcal{O}(m)$.

## 6.1 Budget-Constrained Maximum Flows

In the *budget-constrained maximum flow problem*, the aim is to determine a flow with maximum value in an $s$-$t$-network that is additionally restricted by a *budget-constraint* of the form $\sum_{e \in E} b_e \cdot x_e \leqslant B$ for non-negative integers $b_e \in \mathbb{N}$ for each $e \in E$ a budget $B \in \mathbb{N}$. The problem is known to be efficiently solvable by combinatorial algorithms, both in weakly polynomial-time [1, 4, 5, 6] and in strongly polynomial-time [23]. In the following, we present a strongly polynomial-time FPTAS for the problem, which is both much more simple and efficient than the exact strongly polynomial-time algorithm.

In order to apply our framework, we need to show that each feasible solution is decomposable in some kind of basic component and that we are able to handle these components appropriately. Without loss of generality, since each budget-constrained maximum flow $x$ is also a traditional $s$-$t$-flow and since flows on cycles do not contribute to the flow value, it holds that $x$ can be decomposed into $m' \leqslant m$ flows $\overline{x}^{(j)}$ on $s$-$t$-paths $P_j$ such that $x = \sum_{j=1}^{m'} \overline{x}^{(j)}$. Hence, if $x^{(l)}$ denotes the flow

---

[4]Actually, since we do not have direct access to the set $S$, we need to obtain such a vector via an oracle access. However, by calling the oracle once more with a very large value for $\lambda$ or by returning *some* vector found before, we obtain a certificate in $S$, which we can return.

with unit flow value on some path $P_l$ in the set of $s$-$t$-paths $\{P_1, \ldots, P_k\}$, it holds that each (budget-constrained) maximum flow $x$ is contained in the cone $C$ that is generated by the vectors in the set $S := \{x^{(l)} : l \in \{1, \ldots, k\}\}$. Consequently, we can formulate the budget-constrained maximum flow problem as follows:

$$\max \sum_{e \in E} c_e \cdot x_e \tag{8a}$$

$$\text{s.t.} \sum_{e \in E} b_e \cdot x_e \leqslant B, \tag{8b}$$

$$x_e \leqslant u_e \qquad \qquad \text{for each } e \in E, \tag{8c}$$

$$x \in C, \tag{8d}$$

where $c_e = 1$ if $e \in \delta^-(t)$, and $c_e = 0$, else. Note that the flow conservation constraints are now modeled by the containment in the cone $C$, such that a packing problem over a polyhedral cone remains, i.e., a problem of the form (1).

In the above formulation, it holds that $c^{\mathsf{T}} x^{(l)} = \hat{c} := 1$ for each $x^{(l)} \in S$ since each $s$-$t$-path contributes equally to the value of the flow. Hence, we can apply Theorem 5 if we can show that there is a minimizing oracle for $S$, i.e., that we can determine a vector $x^{(l)}$ minimizing $d^{\mathsf{T}} x^{(l)}$ for a given cost vector $d$. This simply reduces to the determination of a shortest $s$-$t$-path with respect to the edge lengths $d$. Note that, since the vector $a$ is always positive in each component according to Observation 1 and since $\hat{c} = 1$, we need to search for a shortest path with non-negative edge lengths in $\mathrm{SP}(m, n) \in \mathcal{O}(m + n \log n)$ time according to the proof of Theorem 5. Thus, we get that there is an FP-TAS for the budget-constrained maximum flow problem running in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot \mathrm{SP}(m, n)\right)$ time since the number $N$ of non-zero entries in the constraint matrix in (8) is bounded by $2m \in \mathcal{O}(\mathrm{SP}(m, n))$. Note that this running time is still obtained even if we add (a constant number of) different budget-constraints.

We want to stress that our framework allows to stick to the commonly used edge-based formulation of the problem, in which there is a linear number of variables defining the flow on single edges. In contrast, one is required to use the path-based formulation of the problem when using the original framework of Garg and Koenemann [17]: The flow conservation constraints, which define the "shape" of a feasible flow, cannot be directly used in a formulation as a packing problem. These constraints, however, are now modeled by the containment in the cone $C$. Moreover, note that the only ingredients that we used are that (1) each flow decomposes into flows on some type of basic components ($s$-$t$-paths) and (2) that we are able to handle these basic components efficiently, which allowed us to apply the framework.

## 6.2 Budget-Constrained Minimum Cost Flows

In the *budget-constrained minimum cost flow problem*, the aim is to determine a minimum cost flow subject to a budget constraint of the form $\sum_{e \in E} b_e \cdot x_e \leqslant B$, similarly to the budget-constrained maximum flow problem that was studied above. The problem is known to be efficiently solvable

in weakly and strongly polynomial-time [22, 23]. In [22], a strongly polynomial-time FPTAS was presented for the budget-constrained minimum cost flow problem, which runs in

$$\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot (nm \log m \log \log m + n^3 \log n + nm \log^2 n \log \log n)\right)$$

time and which uses similar ideas as the ones presented above. In the following, we improve upon this result.

When considering the (equivalent) circulation based version of the problem in which there are no demands and flow conservation holds at each node, it is easy to see that each optimal flow can be decomposed into flows on simple cycles. Hence, we can restrict our considerations to flows that are contained in the cone C that is spanned by flows on simple cycles with unit flow value. The result of Theorem 5 cannot be applied to this problem for two reasons: On the one hand, since we are dealing with arbitrary costs, it clearly does no longer hold that $c^\mathsf{T} x^{(l)}$ is constant among all flows on cycles with unit flow value. On the other hand, any minimizing oracle would be required to return a vector $x^{(l)}$ that minimizes $d^\mathsf{T} x^{(l)}$ for a given cost vector $d$, so it would be necessary to find a most negative cycle $C^*$ in the underlying graph. However, this problem is known to be $\mathcal{NP}$-complete in general since finding a most negative simple cycle in a graph with edge costs $d_e = -1$ for each $e \in E$ is equivalent to deciding if the graph contains a Hamiltonian cycle (cf. Garey and Johnson [16]). Nevertheless, we are able to determine a cycle $C$ with the same *sign* as the most negative cycle $C^*$ efficiently by computing a minimum mean cycle in $\mathcal{O}(nm)$ time (cf. [26]). Hence, we can apply both Theorem 9 and Theorem 12 to the budget-constrained minimum cost flow problem in order to obtain a weakly polynomial-time FPTAS running in

$$\mathcal{O}\left(nm \cdot \left(\frac{1}{\varepsilon^2} \cdot m \log m + \log \log M - \log \frac{1}{\varepsilon} - \log m - \log \log m\right)\right)$$

time and, since the minimum mean cycle algorithm of Karp [26] is both strongly polynomial and strongly combinatorial, a strongly polynomial-time FPTAS with a time bound of

$$\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot nm + (nm)^2\right) = \mathcal{O}\left(nm \cdot \left(\frac{1}{\varepsilon^2} \cdot m \log m + nm\right)\right).$$

The latter running time can be improved by making use of the following observation: As it was shown in Lemma 11, the sign oracle is incorporated into Megiddo's parametric search in order to determine a minimizer of

$$\min_{\substack{l \in \{1,\dots,k\} \\ c^\mathsf{T} x^{(l)} > 0}} \frac{a^\mathsf{T} x^{(l)}}{c^\mathsf{T} x^{(l)}} \tag{6}$$

for a positive cost vector $a$ and a vector $c$. In the case of the budget-constrained minimum cost flow problem, this reduces to the determination of a *minimum ratio cycle* $C$. Megiddo [30] derived an algorithm that determines a minimum ratio cycle in a simple graph in $\mathcal{O}(n^3 \log n + nm \log^2 n \log \log n)$ time by making use of a parallel algorithm for the all-pairs shortest path problem in combination with Karp's minimum mean cycle algorithm [26] as a negative cycle detector in his parametric search. This running time was later improved by Cole [10] to $\mathcal{O}(n^3 \log n + nm \log^2 n)$. Hence, the strongly

polynomial FPTAS can be improved to run in

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot m\log m\cdot nm+n^3\log n+nm\log^2 n\right)=\mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot nm^2\log n\right)$$

time on simple graphs. In the case of multigraphs, one can use a technique introduced in [22] in order to transform the graph into an equivalent simple graph in $\mathcal{O}(nm\log m\log\log m)$ time before applying Cole's minimum ratio cycle algorithm, yielding an FPTAS running in

$$\mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot m\log m\cdot nm+nm\log m\log\log m+n^3\log n+nm\log^2 n\right)=\mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot nm^2\log m\right)$$

time. Hence, in both cases, the strongly polynomial-time FPTAS dominates the FPTASs introduced above. The claimed running time holds even if we add up to $\mathcal{O}(n)$ different budget constraints to the problem.

## 6.3 Budget-Constrained Minimum Cost Generalized Flows

The *generalized minimum cost flow problem* is an extension of the minimum cost flow problem, in which each edge $e\in E$ is denoted with an additional *gain factor* $\gamma_e$. The flow that enters some edge $e$ is multiplied by $\gamma_e$ as soon as it leaves the edge (cf. [36]). In the *budget-constrained minimum cost generalized flow problem*, the flow is additionally restricted by a budget-constraint of the form $\sum_{e\in E}b_e\cdot x_e\leqslant B$ as above.

The traditional minimum cost generalized flow problem (without an additional budget constraint) is known to be solvable by combinatorial algorithms in weakly polynomial-time [37]. Moreover, there is a strongly polynomial-time FPTAS running in $\widetilde{\mathcal{O}}\left(\log\frac{1}{\varepsilon}\cdot n^2m^2\right)$ time presented by Wayne [37]. However, this algorithm makes use of the inherent structure of traditional generalized flows and cannot be extended to the budget-constrained case without further ado. Earlier, Oldham [31] presented an FP-TAS for the related generalized minimum cost maximum flow problem with non-negative edge costs with a weakly polynomial running time of $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2}\cdot\log\frac{1}{\varepsilon}\cdot n^2m^2\cdot\log mCU\right)$, which, as well, cannot be easily extended to the budget-constrained case. Another weakly polynomial-time FPTAS for this problem running[5] in $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2}\cdot nm^2\cdot(\log\frac{1}{\varepsilon}+\log\log M)\right)$ time was presented by Fleischer and Wayne [13], which is also based on the procedure of Garg and Koenemann [17] and which can be extended to the budget-constrained version of the problem. Using our framework, we present two much simpler FPTASs that work for the generalized minimum cost flow with arbitrary edge costs and that complement the above ones by achieving better time complexities in specific cases.

Again, we consider the circulation based version of the problem in which the excess is zero at each node $v\in V$. As it was shown in [37], every such generalized circulation $x$ can be decomposed into at most $m$ flows on *unit-gain cycles* and *bicycles*, i.e., flows on cycles $C$ with $\prod_{e\in C}\gamma_e=1$ and flows on pairs of cycles $(C_1,C_2)$ with $\prod_{e\in C_1}\gamma_e<1$ and $\prod_{e\in C_2}\gamma_e>1$ that are connected by a path, respectively. Hence, every generalized circulation lies in the cone $C$ that is generated by flows on

---

[5]M denotes the largest absolute value of each number given in the problem instance, assuming gain factor are given as ratios of integers.

such unit-gain cycles and bicycles:

$$\max \sum_{e \in E} c_e \cdot x_e \tag{9a}$$

$$\text{s.t.} \sum_{e \in E} b_e \cdot x_e \leqslant B, \tag{9b}$$

$$x_e \leqslant u_e \qquad\qquad \text{for each } e \in E, \tag{9c}$$

$$x \in C. \tag{9d}$$

Note that this formulation does not differ from the models in the previous applications. Instead, the "structural complexity" of the problem that comes with the introduction of gain factors is modeled by the containment in the cone C. We are done if we are able to find a separation oracle for the set that generates this cone. Wayne [37] shows that there is a unit-gain cycle or bicycle with negative costs in a given network if and only if a specialized system with two variables per inequality (2VPI) is infeasible. Among others, Cohen and Megiddo [9] present a procedure that checks the feasibility of such a system and, in case that it is infeasible, provides a "certificate of infeasibility", which corresponds to a negative cost unit-gain cycle/bicycle [37]. This procedure runs in $\widetilde{\mathcal{O}}(n)$ time on $\mathcal{O}(nm)$ processors. Hence, when used as a separation oracle, we are able to apply Theorem 9. This yields an FPTAS running in

$$\widetilde{\mathcal{O}}\left(n^2 m \cdot \left(\frac{1}{\varepsilon^2} \cdot m + \log\log M' - \log\frac{1}{\varepsilon}\right)\right)$$

time, where $M'$ is an upper bound on the absolute costs $c_e$, fees $b_e$, and capacities $u_e$ of the edges $e \in E$ – independent of the numbers involved to represent the gain factors. Moreover, since the separation oracle is both strongly polynomial and strongly combinatorial [37], we can apply Theorem 12 in order to obtain a strongly polynomial-time FPTAS. Using parallelization techniques that are common when using Megiddo's parametric search [30], the time that is necessary to find an initial most violated dual constraint using Lemma 13 can be improved from $\widetilde{\mathcal{O}}((nm)^2)$ to $\widetilde{\mathcal{O}}(n \cdot (nm + nm\log(nm) + \log(nm) \cdot (n^2 m))) = \widetilde{\mathcal{O}}(n^3 m)$. This yields an FPTAS with a strongly polynomial running time in

$$\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m \cdot n^2 m + n^3 m\right) = \widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot n^2 m^2\right).$$

This algorithm embodies the first strongly polynomial-time FPTAS for the budget-constrained generalized minimum cost flow problem and improves upon the running time of the weakly polynomial-time FPTAS. Moreover, this FPTAS outperforms both the algorithm of Oldham [31] and, for large values of M or small values of $\varepsilon$, the algorithm of Fleischer and Wayne [13].

## 6.4 Maximum Flows in Generalized Processing Networks

*Generalized processing networks* extend traditional networks by a second kind of capacities, so called *dynamic capacities*, that depend on the flow itself. More precisely, the flow on each edge $e = (v, w) \in E$

is additionally constrained to be at most $\alpha_e \cdot \sum_{e' \in \delta^+(v)} x_{e'}$ for some edge-dependent constant $\alpha_e \in (0, 1]$, i.e., the flow on $e$ may only make up a specific fraction $\alpha_e$ of the total flow leaving the starting node $v$ of $e$. This extension allows to model manufacturing and distillation processes, in particular (cf. [24]).

Similar to s-t-paths, the "basic component" in the field of generalized processing networks is the notion of so-called *basic flow distribution schemes*. For each node $v \in V$ with $\delta^+(v) \neq \emptyset$, such a basic flow distribution scheme $\beta$ is a function that assigns a value in $[0, \alpha_e]$ to each edge $e \in \delta^+(v)$ such that $\sum_{e \in \delta^+(v)} \beta_e = 1$ and at most one edge $e \in \delta^+(v)$ fulfills $\beta_e \in (0, \alpha_e)$. Intuitively, a basic flow distribution scheme describes how flow can be distributed to the outgoing edges at each node without violating any dynamic capacity constraint.

In [24], the authors show that each flow in a generalized processing network can be decomposed into at most $2m$ flows on basic flow distribution schemes. Hence, we can conclude that each *maximum flow in a generalized processing network* is contained in the cone $C$ that is generated by unit-flows on basic flow distribution schemes. Moreover, for the problem on acyclic graphs and for a given cost vector $d$, we can determine a basic flow distribution scheme $\beta$ that allows a unit-flow $x$ with minimum costs $d(x) := \sum_{e \in E} d_e \cdot x_e$ in linear time $\mathcal{O}(m)$ (cf. [24]). By using Theorem 5, we get an FPTAS for the maximum flow problem in generalized processing networks with a strongly polynomial running-time of $\mathcal{O}(\frac{1}{\varepsilon^2} \cdot m^2 \log m)$. This result is in particular interesting since it is unknown whether an *exact* solution can be determined in strongly polynomial time since the problem is at least as hard to solve as any linear fractional packing problem (cf. [24] for further details).

## 6.5 Minimum Cost Flows in Generalized Processing Networks

Similar to the previous problem, each *minimum cost flow in a generalized processing network* is contained in the cone that is generated by flows with unit flow value on basic flow distribution schemes. On acyclic graphs, we have the same minimizing oracle as described above. Since the costs $c_e$ are now arbitrary, we can no longer apply Theorem 5. Nevertheless, since each minimizing oracle induces a sign oracle, we are able to apply Theorem 12, which yields an FPTAS for the problem running in strongly polynomial-time

$$\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m \log m \cdot m + m^2\right) = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m\right).$$

This matches the time complexity of the maximum flow variant of the problem described in Section 6.4.

## 6.6 Maximum Concurrent Flow Problem

The *maximum concurrent flow problem* is a variant of the maximum multicommodity flow problem, in which a demand $d_j$ is given for each commodity $j$ with source-sink-pair $(s_j, t_j) \in V \times V$. The task is to determine the maximum value of $\lambda$ such that a fraction $\lambda$ of all demands is satisfied without violating any edge capacity. While several FPTASs emerged for this problem, the best time bound at

present is given by $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot (m^2 + kn)\right)$ due to Karakostas [25], where $k \in \mathcal{O}(n^2)$ denotes the number of commodities.

The problem can be approximated efficiently with our framework by using the following novel approach: In order to improve the objective function value by one unit, we need to send $d_j$ units of each commodity. Hence, each concurrent flow decomposes into basic components of the following type: A set of flows on $k$ paths, containing a flow with value $d_j$ on an $(s_j, t_j)$ path for each commodity $j$. For a given (positive) cost vector, a basic component with minimum costs can be found by determining a shortest path between each commodity. Since Dijkstra's (1959) algorithm computes the shortest paths from one node to *every* other node, we only need to apply it $\min\{k, n\}$ times (once for each of the distinct sources of all commodities), which yields a minimizing oracle running in $\mathcal{O}(\min\{k, n\} \cdot (m + n \log n))$ time and an FPTAS running in $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2 \cdot \min\{k, n\}\right)$ time according to Theorem 5. This algorithm has a worse time complexity than the one of Karakostas [25]. Nevertheless, the application of the presented framework is much simpler than the algorithm given in [25] (and even matches its time complexity in the case of sparse graphs with a large number of commodities) and inherently allows the incorporation of additional budget-constraints.

## 6.7 Maximum Weighted Multicommodity flow Problem

The *maximum weighted multicommodity flow problem* is a generalization of the maximum multicommodity flow problem, in which a positive weight $c_j$ is denoted with each commodity and the aim is to maximize the weighted flow value. The problem is known to be solvable in $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2 \min\{\log C, k\}\right)$ time as shown by Fleischer [14], where $C$ denotes the largest ratio of any two weights of commodities.

Similar to the multicommodity flow problem, each feasible flow decomposes into flows with unit flow value on single $(s_j, t_j)$ paths. Moreover, the determination of such a path with minimal costs reduces to $\min\{k, n\}$ shortest path computations with possibly negative costs, similar to the maximum concurrent flow problem considered above. Using similar ideas as in the case of the budget-constrained minimum cost flow problem (Section 6.2), this would yield an FPTAS with a running time in $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot \min\{n, k\} \cdot nm + \min\{n, k\} \cdot n^3\right)$.

This running time can be improved as follows: As above, we can consider the cone $C$ to be spanned by flows on $(s_j, t_j)$ paths for each commodity, but where each flow between any $(s_j, t_j)$-pair now has flow value $\frac{1}{c_j}$. Each vector in the ground set $S$ then has uniform costs. In order to apply Theorem 5, we need to be able to determine a cost-minimal vector with respect to a given positive cost vector $d$. One straight-forward way to obtain such a minimizing oracle is to compute a shortest path for each commodity $j$ using the edge lengths $\frac{d_e}{c_j}$ for each $e \in E$ and to choose a shortest path among all commodities. This would result in a $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2} \cdot m^2 \cdot k\right)$ time FPTAS, similar to the previous application. However, it suffices to compute only $\min\{n, k\}$ shortest paths per iteration, which can be seen as follows: For each node $s$ out of the set of the $\min\{k, n\}$ distinct source nodes, we perform two steps: We first compute the shortest path distance to every other node using Dijkstra's (1959) algorithm. Afterwards, for each node that corresponds to the sink $t_j$ of a commodity $j$ with source $s_j = s$, we multiply the distance from $s_j$ to $t_j$ by $\frac{1}{c_j}$. By repeating this procedure for each source and keeping track of the overall shortest path, we obtain a minimizing oracle. This yields an FPTAS running in

$\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^2} \cdot m^2 \cdot \min\{n, k\}\right)$ time, which complements the result of Fleischer [14]. This example shows that more sophisticated definitions of the ground set S and the cone C may improve the running time of the procedure.

Finally, using this approach, we can even further improve the algorithm to obtain a time bound of $\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^2} \cdot m^2\right)$ using an idea that was applied by Fleischer [14] to the (unweighted) multicommodity flow problem: For an initially tight lower bound $\underline{L}$ on the length of a shortest path for any commodity (which can be computed in $\widetilde{\mathcal{O}}(\min\{n, k\} \cdot m)$ time as above at the beginning), we can stick to one commodity j in each iteration of the overall procedure and compute a single shortest path from the source $s_j$ to the sink $t_j$. Once the length of this shortest path multiplied by $\frac{1}{c_j}$ becomes as large as $(1 + \epsilon) \cdot \underline{L}$, we go on to the next commodity and continue the procedure. After each commodity was considered, we update $\underline{L}$ to $(1 + \epsilon) \cdot \underline{L}$ and continue with the first commodity. Following the lines of Fleischer [14], this yields an FPTAS running in $\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^2} \cdot (m^2 + km)\right)$ time as there are $\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^2} \cdot k\right)$ shortest path computations that lead to a change of the commodity. However, since Dijkstra's (1959) algorithm computes the distance to every other node, we only need to consider $\min\{k, n\}$ different nodes by grouping commodities with the same source as above, which reduces the running time to $\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^2} \cdot m^2\right)$ (see [14] for details on the algorithm). Although Fleischer [14] both considered this technique and introduced the maximum weighted multicommodity flow problem, she refrained from applying this procedure to the problem.

## 6.8 Maximum Spanning Tree Packing Problem

In the *maximum spanning tree packing problem*, one is given an undirected graph $G = (V, E)$ with positive edge capacities $u_e$. Let $\mathcal{T}$ denote the set of all spanning trees in G. The aim is to find a solution to the problem

$$\max \sum_{T \in \mathcal{T}} x_T$$

$$\text{s.t.} \sum_{T \in \mathcal{T}: e \in T} x_T \leqslant u_e \qquad\qquad \forall\, e \in E,$$

$$x_T \geqslant 0 \qquad\qquad \forall T \in \mathcal{T},$$

i.e., one seeks to pack as many spanning trees as possible (in the fractional sense) without violating any edge capacity. While the problem was investigated in a large number of publications, the fastest (exact) algorithm for the problem is due to Gabow and Manu [15] and runs in $\mathcal{O}\left(n^3 m \log \frac{n^2}{m}\right)$ time.

Let S denote the set of incidence vectors $\chi_T$ of spanning trees $T \in \mathcal{T}$, where $(\chi_T)_e = 1$ if $e \in T$ and $(\chi_T)_e = 0$ else. Since each spanning tree contains exactly $n - 1$ edges, the problem can be stated in

an equivalent edge-based fashion as follows:

$$\max \frac{1}{n-1} \cdot \sum_{e \in E} x_e$$

$$\text{s.t. } x_e \leqslant u_e \qquad\qquad\qquad\qquad \forall\, e \in E,$$

$$x \in C.$$

In order to apply Theorem 5 (which is eligible since each spanning tree contributes equally to the objective function value), we need a minimizing oracle for the set $S$. However, this simply reduces to the determination of a minimum spanning tree, which can be done in $\mathcal{O}(m \cdot \alpha(m, n))$ time, where $\alpha(m, n)$ denotes the inverse Ackermann function (cf. [7]). This yields a strongly polynomial-time FPTAS for the problem running in $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot \alpha(m, n)\right)$ time.

Our framework also applies to a *weighted* version of the problem: Assume that each edge is labeled with an additional cost $c_e$ (with arbitrary sign) and assume that the weight $c(T)$ of each spanning tree $T \in \mathcal{T}$ is defined to be the sum of the weights of its edges, i.e., $c(T) := \sum_{e \in E} c_e$. The aim is then to maximize the objective function $\sum_{T \in \mathcal{T}} c(T) \cdot x_T$. As above, we can stick to an equivalent edge-based formulation using the objective function $\frac{1}{n-1} \cdot \sum_{e \in E} c_e \cdot x_e$. The minimum spanning tree algorithm can then be used as a sign oracle, which allows us to apply Theorem 12 to the problem. This yields an FPTAS for the maximum weighted spanning tree packing problem running in strongly polynomial time

$$\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot \alpha(m, n) + m^2 \cdot \alpha^2(m, n)\right) = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot \alpha(m, n)\right).$$

To the best of our knowledge, this is the first combinatorial approximation algorithm for this problem.

## 6.9  Maximum Matroid Base Packing Problem

Having a closer look at the results of Section 6.8, one might expect that they can be generalized to matroids: As spanning trees form the bases of graphic matroids, the presented ideas suggest that the framework can also be applied to packing problems over general matroids. In the *maximum matroid base packing problem*, a matroid $M(S, \mathcal{I})$ with ground set $S := \{1, \dots, m\}$ and independent sets in $\mathcal{I}$ is given as well as a positive capacity $u_i \in \mathbb{N}_{>0}$ for each $i \in S$. For $r$ to be the rank function of $M$, let $\mathcal{B} \subset \mathcal{I}$ denote the set of bases such that $I \in \mathcal{B}$ if and only if $r(I) = r(S)$. The aim of the problem is to pack as many bases of $M$ as possible (in the fractional sense) without violating any capacity

constraints:

$$\max \sum_{I \in \mathcal{B}} x_I$$

$$\text{s.t.} \sum_{I \in \mathcal{B}: i \in I} x_I \leqslant u_i \qquad\qquad \forall\, i \in S,$$

$$x_I \geqslant 0 \qquad\qquad \forall I \in \mathcal{B}.$$

As it is common when dealing with matroids, we assume that the matroid is described by an *independence testing oracle*, which checks if some set $S' \subseteq S$ is independent in $M$ (cf. [33]). Let $F(m)$ denote the running time of this oracle. As it is shown in [33], the problem can be solved in $\mathcal{O}(m^7 \cdot F(m))$ time using a result derived by Cunningham [11].

As it was the case in the maximum spanning tree packing problem in Section 6.8, the problem can be formulated in an equivalent element-based fashion as follows:

$$\max \frac{1}{r(S)} \cdot \sum_{i \in S} x_i$$

$$\text{s.t.}\; x_i \leqslant u_i \qquad\qquad \forall\, i \in S,$$

$$x \in C,$$

where the cone $C$ is spanned by the incidence vectors of bases in $\mathcal{B}$. In order to apply our framework, we need to be able to handle these bases efficiently. However, as we are dealing with matroids, we can find a cost-minimal basis $I \in \mathcal{B}$ of $M$ with respect to a given cost vector $d$ just by applying the Greedy algorithm (cf. [33]): Starting with $I = \emptyset$, we sort the elements by their costs and iteratively add each element in the sorted sequence unless the independence test fails. This yields a minimizing oracle for $\mathcal{B}$ running in $\mathcal{O}(m \cdot F(m) + m \log m)$ time. Hence, we immediately get an FPTAS for the maximum matroid base packing problem running in strongly polynomial time $\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot (F(m) + \log m)\right)$ according to Theorem 5.

As it was the case in Section 6.8, we can also extend our results to a *weighted* version of the problem: Assume we are additionally given costs $c_i \in \mathbb{Z}$ and want to maximize $\sum_{I \in \mathcal{B}} c(I) \cdot x_I$, where $c(I) := \sum_{i \in I} c_i$. Equivalently, we can also maximize $\frac{1}{r(S)} \cdot \sum_{i \in S} c_i \cdot x_i$ in the element-based formulation of the problem. Using the above minimizing oracle as a sign oracle, we can apply both Theorem 9 and, in case that the independence testing oracle is strongly polynomial and strongly combinatorial, Theorem 12 to the problem. This yields two FPTASs for the problem running in

$$\mathcal{O}\left((m \cdot F(m) + m \log m) \cdot \left(\frac{1}{\varepsilon^2} \cdot m \log m + \log \log M - \log \frac{1}{\varepsilon} - \log m - \log \log m\right)\right)$$

and

$$\mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot m^2 \log m \cdot (F(m) + \log m) + (m \cdot F(m) + m \log m)^2\right)$$

time, respectively. To the best of our knowledge, no other polynomial-time algorithm is known for this problem.

# 7 Conclusion

We investigated an extension of the fractional packing framework by Garg and Koenemann [17] that generalizes their results to fractional packing problems over polyhedral cones. By combining a large diversity of known techniques, we derived a framework that can be easily adopted to a large class of network flow and packing problems. This framework may in particular be applicable even if the cone has an exponential-sized representation as it only relies on a strongly polynomial number of oracle calls in order to gather information about the cone. In many cases, its application allows the derivation of approximation algorithms that are either the first ones with a strongly polynomial running time or the first combinatorial ones at all. For a large variety of applications, we were even able to complement or improve existing results.

The presented paper raises several questions for future research. On the one hand, we believe that our results can be applied to a much larger set of problems and can be used to obtain combinatorial FPTASs for complex problems without much effort. It may also be possible that the results continue to hold for even weaker kinds of oracles. On the other hand, as our framework is based on the one of Garg and Koenemann [17] in its core, all of the derived approximation algorithms have a running time in $\Omega(\frac{1}{\varepsilon^2} \cdot m \log m \cdot n)$ and, in particular, have a quadratic dependency on $\frac{1}{\varepsilon}$. It may be possible to achieve a subquadratic dependency on $\frac{1}{\varepsilon}$ by relying on other approaches such as the one of Bienstock and Iyengar [3]. Nevertheless, it seems that this trade comes with a worse dependence on other parameters, a worse practical performance, or a worse generality of the presented results.

# References

[1] R. K. Ahuja and J. B. Orlin. A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 25(2):89–98, 1995.

[2] D. Bienstock. *Potential function methods for approximately solving linear programming problems: theory and practice*, volume 53. Springer Science & Business Media, 2006.

[3] D. Bienstock and G. Iyengar. Approximating fractional packings and coverings in o (1/epsilon) iterations. *SIAM Journal on Computing*, 35(4):825–854, 2006.

[4] C. Çalışkan. A double scaling algorithm for the constrained maximum flow problem. *Computers & Operations Research*, 35(4):1138–1150, 2008.

[5] C. Çalışkan. On a capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 53(3):229–230, 2009.

[6] C. Çalışkan. A faster polynomial algorithm for the constrained maximum flow problem. *Computers & Operations Research*, 39(11):2634–2641, 2012.

[7] B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM (JACM)*, 47(6):1028–1047, 2000.

[8] E. Cohen and N. Megiddo. *Maximizing concave functions in fixed dimension*. World Scientific, 1990.

[9] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM Journal on Computing*, 23(6):1313–1347, 1994.

[10] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM (JACM)*, 34(1):200–208, 1987.

[11] W.H. Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36(2):161–188, 1984.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271, 1959.

[13] L. K. Fleischer and K. D. Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91(2):215–238, 2002.

[14] L.K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[15] H.N. Gabow and K.S. Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1-2):83–109, 1998.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, New York, 1979.

[17] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[18] M.D. Grigoriadis and L.G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.

[19] M.D. Grigoriadis and L.G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21(2):321–340, 1996.

[20] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, 1993.

[21] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.

[22] M. Holzhauser, S. O. Krumke, and C. Thielen. On the complexity and approximability of budget-constrained minimum cost flows. *submitted to Information Processing Letters*, 2015.

[23] M. Holzhauser, S. O. Krumke, and C. Thielen. Budget-constrained minimum cost flows. *Journal of Combinatorial Optimization*, 31(4):1720–1745, 2016.

[24] M. Holzhauser, S. O. Krumke, and C. Thielen. Maximum flows in generalized processing networks. *Journal of Combinatorial Optimization*, pages 1–31, 2016.

[25] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)*, 4(1):13, 2008.

[26] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.

[27] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[28] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[29] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.

[30] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983.

[31] J.D. Oldham. Combinatorial approximation algorithms for generalized flow problems. *Journal of Algorithms*, 38(1):135–169, 2001.

[32] S.A. Plotkin, D.B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[33] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[34] S. Toledo. Maximizing non-linear concave functions in fixed dimension. In *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on*, pages 676–685. IEEE, 1992.

[35] S. Toledo. Approximate parametric searching. *Information processing letters*, 47(1):1–4, 1993.

[36] K. D. Wayne. *Generalized Maximum Flow Algorithms*. PhD thesis, Cornell University, 1999.

[37] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research*, 27(3):445–459, 2002.

[38] N.E. Young. Randomized rounding without solving the linear program. In *SODA*, volume 95, pages 170–178, 1995.