An Accelerated Newton–Dinkelbach Method and Its Application to Two Variables per Inequality Systems

Daniel Dadush
CWI, Amsterdam, The Netherlands

Zhuan Khye Koh
Department of Mathematics, London School of Economics and Political Science, UK

Bento Natura
Department of Mathematics, London School of Economics and Political Science, UK

László A. Végh
Department of Mathematics, London School of Economics and Political Science, UK

Abstract

We present an accelerated, or "look-ahead" version of the Newton–Dinkelbach method, a well-known technique for solving fractional and parametric optimization problems. This acceleration halves the Bregman divergence between the current iterate and the optimal solution within every two iterations. Using the Bregman divergence as a potential in conjunction with combinatorial arguments, we obtain strongly polynomial algorithms in three applications domains: *(i)* For linear fractional combinatorial optimization, we show a convergence bound of $O(m \log m)$ iterations; the previous best bound was $O(m^2 \log m)$ by Wang et al. (2006). *(ii)* We obtain a strongly polynomial label-correcting algorithm for solving linear feasibility systems with two variables per inequality (2VPI). For a 2VPI system with n variables and m constraints, our algorithm runs in $O(mn)$ iterations. Every iteration takes $O(mn)$ time for general 2VPI systems, and $O(m + n \log n)$ time for the special case of deterministic Markov Decision Processes (DMDPs). This extends and strengthens a previous result by Madani (2002) that showed a weakly polynomial bound for a variant of the Newton–Dinkelbach method for solving DMDPs. *(iii)* We give a simplified variant of the parametric submodular function minimization result by Goemans et al. (2017).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Mathematical optimization

Keywords and phrases Newton–Dinkelbach method, fractional optimization, parametric optimization, strongly polynomial algorithms, two variables per inequality systems, Markov decision processes, submodular function minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.36

Related Version *Full Version*: `https://arxiv.org/abs/2004.08634`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement nos. 757481–ScaleOpt and 805241–QIP).

Acknowledgements The fourth author would like to thank Neil Olver for several inspiring discussions on 2VPI systems, in particular, on symmetries of the problem.

1 Introduction

Linear fractional optimization problems are well-studied in combinatorial optimization. Given a closed domain $\mathcal{D} \subseteq \mathbb{R}^m$ and $c, d \in \mathbb{R}^m$ such that $d^\top x > 0$ for all $x \in \mathcal{D}$, the problem is

$$\inf c^\top x / d^\top x \quad \text{s.t. } x \in \mathcal{D}\,. \tag{1}$$

The domain $\mathcal{D}$ could be either a convex set or a discrete set $\mathcal{D} \subseteq \{0,1\}^m$. Classical examples include finding minimum cost-to-time ratio cycles and minimum ratio spanning trees. One can equivalently formulate (1) as a parametric search problem. Let

$$f(\delta) = \inf\{(c - \delta d)^\top x : x \in \mathcal{D}\}, \tag{2}$$

be a concave and decreasing function. Assuming (1) has a finite optimum δ, it corresponds to the unique root $f(\delta) = 0$.

A natural question is to investigate how the computational complexity of solving the minimum ratio problem (1) may depend on the complexity of the corresponding linear optimization problem $\min c^\top x$ s.t. $x \in \mathcal{D}$. Using the reformulation (2), one can reduce the fractional problem to the linear problem via binary search; however, the number of iterations needed to find an exact solution may depend on the bit complexity of the input. A particularly interesting question is: assuming there exists a strongly polynomial algorithm for linear optimization over a domain $\mathcal{D}$, can we find a strongly polynomial algorithm for linear fractional optimization over the same domain?

A seminal paper by Megiddo [14] introduced the *parametric search* technique to solve linear fractional combinatorial optimization problems. He showed that if the linear optimization algorithm only uses $p(m)$ comparisons and $q(m)$ additions, then there exists an $O(p(m)(p(m)+q(m))$ algorithm for the linear fractional optimization problem. This in particular yielded the first strongly polynomial algorithm for the minimum cost-to-time ratio cycle problem. On a very high level, parametric search works by simulating the linear optimization algorithm for the parametric problem (2), with the parameter $\delta \in \mathbb{R}$ being indeterminate.

A natural alternative approach is to solve (2) using a standard root finding algorithm. Radzik [18] showed that for a discrete domain $\mathcal{D} \subseteq \{0,1\}^m$, the *discrete Newton* method – in this context, also known as *Dinkelbach's method* [4] – terminates in a strongly polynomial number of iterations. In contrast to parametric search, there are no restrictions on the possible operations in the linear optimization algorithm. In certain settings, such as the maximum ratio cut problem, the discrete Newton method outperforms parametric search; we refer to the comprehensive survey by Radzik [19] for details and comparison of the two methods.

1.1 Our Contributions

We introduce a new, *accelerated variant of Newton's method for univariate functions*. Let $f : \mathbb{R} \to \mathbb{R} \cup \{-\infty\}$ be a concave function. Under some mild assumptions on f, our goal is to either find the largest root, or show that no root exists. Let δ^* denote the largest root, or in case $f < 0$, let δ^* denote the largest maximizer of f. For simplicity, we now describe the method for differentiable functions. This will not hold in general: functions of the form (2) will be piecewise linear if $\mathcal{D}$ is finite or polyhedral. The algorithm description in Section 2 uses a form with supergradients (that can be choosen arbitrarily between the left and right derivatives).

The standard Newton method, also used by Radzik, proceeds through iterates $\delta^{(1)} > \delta^{(2)} > \ldots > \delta^{(t)}$ such that $f(\delta^{(i)}) \leq 0$, and updates $\delta^{(i+1)} = \delta^{(i)} - f(\delta^{(i)})/f'(\delta^{(i)})$.

Our new variant uses a more aggressive *"look-ahead"* technique. At each iteration, we compute $\delta = \delta^{(i)} - f(\delta^{(i)})/f'(\delta^{(i)})$, and jump ahead to $\delta' = 2\delta - \delta^{(i)}$. In case $f(\delta') \leq 0$ and $f'(\delta') < 0$, we update $\delta^{(i+1)} = \delta'$; otherwise, we continue with the standard iterate δ.

This modification leads to an improved and at the same time simplified analysis based on the *Bregman divergence* $D_f(\delta^*, \delta^{(i)}) = f(\delta^{(i)}) + f'(\delta^{(i)})(\delta^* - \delta^{(i)}) - f(\delta^*)$. We show that this decreases by a factor of two between any two iterations.

A salient feature of the algorithm is that it handles both feasible and infeasible outcomes in a unified framework. In the context of linear fractional optimization, this means that the assumption $d^\top x > 0$ for all $x \in \mathcal{D}$ in (1) can be waived. Instead, $d^\top x > 0$ is now added as a feasibility constraint to (1). This generalization is important when we use the algorithm to solve two variables per inequality systems.

This general result leads to improvements and simplifications of a number of algorithms using the discrete Newton method.

- For *linear fractional combinatorial optimization*, namely the setting (1) with $\mathcal{D} \subseteq \{0,1\}^m$, we obtain an $O(m \log m)$ bound on the number of iterations, a factor m improvement over the previous best bound $O(m^2 \log m)$ by Wang et al. [25] from 2006. We remark that Radzik's first analysis [18] yielded a bound of $O(m^4 \log^2 m)$ iterations, improved to $O(m^2 \log^2 m)$ in [19].
- Goemans et al. [7] used the discrete Newton method to obtain a strongly polynomial algorithm for parametric submodular function minimization. We give a simple new variant of this result with the same asymptotic running time, using the accelerated algorithm.
- For *two variable per inequality (2VPI)* systems, we obtain a *strongly polynomial label-correcting algorithm*. This will be discussed in more detail next.

1.2 Two Variables Per Inequality Systems

A major open question in the theory of linear programming (LP) is whether there exists a strongly polynomial algorithm for LP. This problem is one of Smale's eighteen mathematical challenges for the twenty-first century [22]. An LP algorithm is *strongly polynomial* if it only uses elementary arithmetic operations $(+, -, \times, /)$ and comparisons, and the number of such operations is polynomially bounded in the number of variables and constraints. Furthermore, the algorithm needs to be in PSPACE, i.e. the numbers occurring in the computations must remain polynomially bounded in the input size.

The notion of a strongly polynomial algorithm was formally introduced by Megiddo [15] in 1983 (using the term *"genuinely polynomial"*), where he gave the first such algorithm for *two variables per inequality (2VPI)* systems. These are feasibility LPs where every inequality contains at most two variables. More formally, let $\mathcal{M}_2(n, m)$ be the set of $n \times m$ matrices with at most two nonzero entries per column. A 2VPI system is of the form $A^\top y \leq c$ for $A \in \mathcal{M}_2(n, m)$ and $c \in \mathbb{R}^m$.

If we further require that every inequality has at most one positive and at most one negative coefficient, it is called a *monotone two variables per inequality* (M2VPI) system. A simple and efficient reduction is known from 2VPI systems with n variables and m inequalities to M2VPI systems with $2n$ variables and $\leq 2m$ inequalities [5, 10].

Connection between 2VPI and parametric optimization. An M2VPI system has a natural graphical interpretation: after normalization, we can assume every constraint is of the form $y_u - \gamma_e y_v \leq c_e$. Such a constraint naturally maps to an arc $e = (u, v)$ with *gain factor* $\gamma_e > 0$ and cost c_e. Based on Shostak's work [21] that characterized feasibility in terms of this graph, Aspvall and Shiloach [2] gave the first weakly polynomial algorithm for M2VPI systems.

We say that a directed cycle C is *flow absorbing* if $\prod_{e \in C} \gamma_e < 1$ and *flow generating* if $\prod_{e \in C} \gamma_e > 1$. Every flow absorbing cycle C implies an upper bound for every variable y_u incident to C; similarly, flow generating cycles imply lower bounds. The crux of Aspvall and Shiloach's algorithm is to find the tightest upper and lower bounds for each variable y_u.

Finding these bounds corresponds to solving fractional optimization problems of the form (1), where $\mathcal{D} \subseteq \mathbb{R}^m$ describes "generalized flows" around cycles. The paper [2] introduced the *Grapevine* algorithm – a natural modification the Bellman-Ford algorithm – to decide whether the optimum ratio is smaller or larger than a fixed value δ. The optimum value can found using binary search on the parameter.

Megiddo's strongly polynomial algorithm [15] replaced the binary search framework in Aspvall and Shiloach's algorithm by extending the parametric search technique in [14]. Subsequently, Cohen and Megiddo [3] devised faster strongly polynomial algorithms for the problem. The current fastest strongly polynomial algorithm is given by Hochbaum and Naor [11], an efficient Fourier–Motzkin elimination with running time of $O(mn^2 \log m)$.

2VPI via Newton's method. Since Newton's method proved to be an efficient and viable alternative to parametric search, a natural question is to see whether it can solve the parametric problems occuring in 2VPI systems. Radzik's fractional combinatorial optimization results [18, 19] are not directly applicable, since the domain $\mathcal{D}$ in this setting is a polyhedron and not a discrete set.[1] Madani [13] used a variant of the Newton–Dinkelbach method as a tool to analyze the convergence of policy iteration on *deterministic Markov Decision Processes (DMDPs)*, a special class of M2VPI systems (discussed later in more detail). He obtained a weakly polynomial convergence bound; it remained open whether such an algorithm could be strongly polynomial.

Our 2VPI algorithm. We introduce a new type of strongly polynomial 2VPI algorithm by combining the accelerated Newton–Dinkelbach method with a *"variable fixing"* analysis. Variable fixing was first introduced in the seminal work of Tardos [23] on minimum-cost flows, and has been a central idea of strongly polynomial algorithms, see in particular [8, 20] for cycle cancelling minimum-cost flow algorithms, and [16, 24] for maximum generalized flows, a dual to the 2VPI problem.

We show that for every iterate $\delta^{(i)}$, there is a constraint that has been "actively used" at $\delta^{(i)}$ but will not be used ever again after a strongly polynomial number of iterations. The analysis combines the decay in *Bregman divergence* shown in the general accelerated Newton–Dinkelbach analysis with a combinatorial *"subpath monotonicity"* property.

Our overall algorithm can be seen as an extension of Madani's DMDP algorithm. In particular, we adapt his "unfreezing" idea: the variables y_u are admitted to the system one-by-one, and the accelerated Newton–Dinkelbach method is used to find the best "cycle bound" attainable at the newly admitted y_u in the graph induced by the current variable set. This returns a feasible solution or reports infeasibility within $O(m)$ iterations. As every iteration takes $O(mn)$ time, our overall algorithm terminates in $O(m^2n^2)$ time. For the special setting of deterministic MDPs, the runtime per iteration improves to $O(m + n \log n)$, giving a total runtime of $O(mn(m + n \log n))$.

Even though our running time bound is worse than the state-of-the-art 2VPI algorithm [11], it is of a very different nature from all previous 2VPI algorithms. In fact, our algorithm is a *label-correcting algorithm*, naturally fitting to the family of algorithms used in other combinatorial optimization problems with constraint matrices from $\mathcal{M}_2(n, m)$ such as maximum flow, shortest paths, minimum-cost flow, and generalized flow problems. We next elaborate on this connection.

[1] The problem could be alternatively formulated with $\mathcal{D} \subseteq \{0, 1\}^m$ but with nonlinear functions instead of $c^\top x$ and $d^\top x$.

Label-correcting algorithms. An important special case of M2VPI systems corresponds to the shortest paths problem: given a directed graph $G = (V, E)$ with target node $t \in V$ and arc costs $c \in \mathbb{R}^E$, we associate constraints $y_u - y_v \leq c_e$ for every arc $e = (u, v) \in E$ and $y_t = 0$. If the system is feasible and bounded, the pointwise maximal solution corresponds to the shortest path labels to t; an infeasible system contains a negative cost cycle. A generic label-correcting algorithm maintains distance labels y that are upper bounds on the shortest path distances to t. The labels are decreased according to violated constraints. Namely, if $y_u - y_v > c_e$, then decreasing y_u to $c_e + y_v$ gives a smaller valid distance label at u. We terminate with the shortest path labels once all constraints are satisfied. The Bellman–Ford algorithm for the shortest paths problem is a particular implementation of the generic label-correcting algorithm; we refer the reader to [1, Chapter 5] for more details.

It is a natural question if label-correcting algorithms can be extended to general M2VPI systems, where constraints are of the form $y_u - \gamma_e y_v \leq c_e$ for a "gain/loss factor" $\gamma_e > 0$ associated with each arc. A fundamental property of M2VPI systems is that, whenever bounded, a unique pointwise maximal solution exists, i.e. a feasible solution y^* such that $y \leq y^*$ for every feasible solution y. A label-correcting algorithm for such a setting can be naturally defined as follows. Let us assume that the problem is bounded. The algorithm should proceed via a decreasing sequence $y^{(0)} \geq y^{(1)} \geq \ldots \geq y^{(t)}$ of labels that are all valid upper bounds on any feasible solution y to the system. The algorithm either terminates with the unique pointwise maximal solution $y^{(t)} = y^*$, or finds an infeasibility certificate.

The basic label-correcting operation is the "arc update", decreasing y_u to $\min\{y_u, c_e + \gamma_e y_v\}$ for some arc $e = (u, v) \in E$. Such updates suffice in the shortest path setting. However, in the general setting arc operations only may not lead to finite termination. Consider a system with only two variables, y_u and y_v, and two constraints, $y_u - y_v \leq 0$, and $y_v - \frac{1}{2} y_u \leq -1$. The alternating sequence of arc updates converges to $(y_u^*, y_v^*) = (-2, -2)$, but does not finitely terminate. In this example, we can "detect" the cycle formed by the two arcs, that implies the bound $y_u - \frac{1}{2} y_u \leq -1$.

Shostak's [21] result demonstrates that arc updates, together with such "cycle updates" should be sufficient for finite termination. Our M2VPI algorithm amounts to the first strongly polynomial label-correcting algorithm for general M2VPI systems, using arc updates and cycle updates.

Deterministic Markov decision processes. A well-studied special case of M2VPI systems in which $\gamma \leq \mathbb{1}$ is known as *deterministic Markov decision process* (DMDP). A *policy* corresponds to selecting an outgoing arc from every node, and the objective is to find a policy that minimizes the total discounted cost over an infinite time horizon. The pointwise maximal solution of this system corresponds to the optimal values of a policy.

The standard policy iteration, value iteration, and simplex algorithms can be all interpreted as variants of the label-correcting framework.[2]

Value iteration can be seen as a generalization of the Bellman–Ford algorithm to the DMDP setting. As our previous example shows, value iteration may not be finite. One could still consider as the termination criterion the point where value iteration "reveals" the optimal policy, i.e. updates are only performed using constraints that are tight in the optimal solution. If each discount factor γ_e is at most γ' for some $\gamma' > 0$, then it is well-known that value iteration converges at the rate $1/(1 - \gamma')$. This is in fact true more generally, for nondeterministic MDPs [12]. However, if the discount factors can be arbitrarily close to 1,

[2] The value sequence may violate monotonicity in certain cases of value iteration.

then Feinberg and Huang [6] showed that value iteration cannot reveal the optimal policy in strongly polynomial time even for DMDPs. Post and Ye [17] proved that simplex with the most negative reduced cost pivoting rule is strongly polynomial for DMDPs; this was later improved by Hansen et al. [9]. These papers heavily rely on the assumption $\gamma \leq \mathbb{1}$, and does not seem to extend to general M2VPI systems.

Madani's previously mentioned work [13] used a variant of the Newton–Dinkelbach method as a tool to analyze the convergence of policy iteration on deterministic MDPs, and derived a weakly polynomial runtime bound.

Paper organization We start by giving preliminaries and introducing notation below. In Section 2, we present an accelerated Newton's method for univariate concave functions, and apply it to linear fractional combinatorial optimization and linear fractional programming. Section 3 contains our main application of the method to the 2VPI problem. Our results on parametric submodular function minimization and deterministic MDPs can be found in the full version of the paper. Missing proofs also appear in the full version.

Preliminaries Let $\mathbb{R}_+$ and $\mathbb{R}_{++}$ be the nonnegative and positive reals respectively, and denote $\bar{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$. Given a proper concave function $f : \mathbb{R} \to \bar{\mathbb{R}}$, let $\mathrm{dom}(f) := \{x : -\infty < f(x) < \infty\}$ be the effective domain of f. For a point $x_0 \in \mathrm{dom}(f)$, denote the set of supergradients of f at x_0 as $\partial f(x_0) := \{g : f(x) \leq f(x_0) + g(x - x_0)\ \forall x \in \mathbb{R}\}$. If x_0 is in the interior of $\mathrm{dom}(f)$, then $\partial f(x_0) = [f'_-(x_0), f'_+(x_0)]$, where $f'_-(x_0)$ and $f'_+(x_0)$ are the left and right derivatives. Throughout, we use $\log(x) = \log_2(x)$ to indicate base 2 logarithm. For $x, y \in \mathbb{R}^m$, denote $x \circ y \in \mathbb{R}^m$ as the element-wise product of the two vectors.

2 An Accelerated Newton–Dinkelbach Method

Let $f : \mathbb{R} \to \bar{\mathbb{R}}$ be a proper concave function such that $f(\delta) \leq 0$ and $\partial f(\delta) \cap \mathbb{R}_{<0} \neq \emptyset$ for some $\delta \in \mathrm{dom}(f)$. Given a suitable starting point, as well as value and supergradient oracles of f, the Newton–Dinkelbach method either computes the largest root of f or declares that it does not have a root. In this paper, we make the mild assumption that f has a root or attains its maximum. Consequently, the point $\delta^* := \max(\{\delta : f(\delta) = 0\} \cup \arg\max f(\delta))$ is well-defined. It is the largest root of f if f has a root. Otherwise, it is the largest maximizer of f. Therefore, the Newton–Dinkelbach method returns δ^* if f has a root, and certifies that $f(\delta^*) < 0$ otherwise.

The algorithm takes as input an initial point $\delta^{(1)} \in \mathrm{dom}(f)$ and a supergradient $g^{(1)} \in \partial f(\delta^{(1)})$ such that $f(\delta^{(1)}) \leq 0$ and $g^{(1)} < 0$. At the start of every iteration $i \geq 1$, it maintains a point $\delta^{(i)} \in \mathrm{dom}(f)$ and a supergradient $g^{(i)} \in \partial f(\delta^{(i)})$ where $f(\delta^{(i)}) \leq 0$. If $f(\delta^{(i)}) = 0$, then it returns $\delta^{(i)}$ as the largest root of f. Otherwise, a new point $\delta := \delta^{(i)} - f(\delta^{(i)})/g^{(i)}$ is generated. Now, there are two scenarios in which the algorithm terminates and reports that f does not have a root: (1) $f(\delta) = -\infty$; (2) $f(\delta) < 0$ and $g \geq 0$ where $g \in \partial f(\delta)$ is the supergradient given by the oracle. If both scenarios do not apply, the next point and supergradient is set to $\delta^{(i+1)} := \delta$ and $g^{(i+1)} := g$ respectively. Then, a new iteration begins.

To accelerate this classical method, we perform an aggressive guess $\delta' := 2\delta - \delta^{(i)} < \delta$ on the next point at the end of every iteration i. We call this procedure *look-ahead*, which is implemented on Lines 7–10 of Algorithm 1. Let $g' \in \partial f(\delta')$ be the supergradient returned by the oracle. If $-\infty < f(\delta') < 0$ and $g' < 0$, then the next point and supergradient are set to $\delta^{(i+1)} := \delta'$ and $g^{(i+1)} := g'$ respectively as $\delta' \geq \delta^*$. In this case, we say that look-ahead is *successful* in iteration i. Otherwise, we proceed as usual by taking $\delta^{(i+1)} := \delta$ and $g^{(i+1)} := g$.

Algorithm 1 LOOK-AHEADNEWTON.

input : Value and supergradient oracles for a proper concave function f, an initial point $\delta^{(1)} \in \mathrm{dom}(f)$ and supergradient $g^{(1)} \in \partial f(\delta^{(1)})$ where $f(\delta^{(1)}) \leq 0$ and $g^{(1)} < 0$.

output : The largest root of f if it exists; report `NO ROOT` otherwise.

$i \leftarrow 1$
while $f(\delta^{(i)}) < 0$ **do**
  $\delta \leftarrow \delta^{(i)} - f(\delta^{(i)})/g^{(i)}$
  $g \in \partial f(\delta)$ `/* Empty if` $f(\delta) = -\infty$ `*/`
  if $f(\delta) = -\infty$ **or** $(f(\delta) < 0$ **and** $g \geq 0)$ **then**
    return `NO ROOT`
  $\delta' \leftarrow 2\delta - \delta^{(i)}$ `/* Look-ahead guess */`
  $g' \in \partial f(\delta')$ `/* Empty if` $f(\delta') = -\infty$ `*/`
  if $-\infty < f(\delta') < 0$ **and** $g' < 0$ **then** `/* Is the guess successful? */`
    $\delta \leftarrow \delta'$, $g \leftarrow g'$
  $\delta^{(i+1)} \leftarrow \delta$, $g^{(i+1)} \leftarrow g$
  $i \leftarrow i + 1$
return $\delta^{(i)}$

It is easy to see that $\delta^{(i)}$ is monotonically decreasing while $f(\delta^{(i)})$ is monotonically increasing. Furthermore, $g^{(i)}$ is monotonically increasing except in the final iteration where it may remain unchanged (Lemma 2.1). Similarly, we have $g^{(i)} < 0$ except possibly in the final iteration when $f(\delta^{(i)}) = 0$.

▶ **Lemma 2.1.** *For every iteration $i \geq 2$, we have $\delta^* \leq \delta^{(i)} < \delta^{(i-1)}$, $f(\delta^*) \geq f(\delta^{(i)}) > f(\delta^{(i-1)})$ and $g^{(i)} \geq g^{(i-1)}$, where the last inequality holds at equality if and only if $g^{(i)} = \inf_{g \in \partial f(\delta^{(i)})} g$, $g^{(i-1)} = \sup_{g \in \partial f(\delta^{(i-1)})} g$ and $f(\delta^{(i)}) = 0$. Moreover,*

$$\frac{f(\delta^{(i)})}{f(\delta^{(i-1)})} + \frac{g^{(i)}}{g^{(i-1)}} \leq 1\,.$$

Our analysis of the Newton–Dinkelbach method utilizes the Bregman divergence associated with f as a potential. Even though the original definition requires f to be differentiable and strictly concave, it can be naturally extended to our setting in the following way.

▶ **Definition 2.2.** *Given a proper concave function $f : \mathbb{R} \to \bar{\mathbb{R}}$, the* Bregman divergence associated with f *is defined as*

$$D_f(\delta', \delta) := \begin{cases} f(\delta) + \sup\limits_{g \in \partial f(\delta)} g(\delta' - \delta) - f(\delta') & \text{if } \delta \neq \delta', \\ 0 & \text{otherwise.} \end{cases}$$

for all $\delta, \delta' \in \mathrm{dom}(f)$ such that $\partial f(\delta) \neq \emptyset$.

Since f is concave, the Bregman divergence is nonnegative. It is again easy to see that $D_f(\delta^*, \delta^{(i)})$ is monotonically decreasing except in the final iteration where it may remain unchanged (Lemma 2.3).

▶ **Lemma 2.3.** *For every iteration $i \geq 2$, we have $D_f(\delta^*, \delta^{(i)}) \leq D_f(\delta^*, \delta^{(i-1)})$ which holds at equality if and only if $g^{(i-1)} = \inf_{g \in \partial f(\delta^{(i-1)})} g$ and $f(\delta^{(i)}) = 0$.*

If look-ahead is successful, then we have made significant progress. Otherwise, by our choice of δ', we learn that we are not too far away from δ^*. The next lemma demonstrates the advantage of using the look-ahead Newton–Dinkelbach method. It exploits the proximity to δ^* to produce a geometric decay in the Bregman divergence of $\delta^{(i)}$ and δ^*.

▶ **Lemma 2.4.** *For every iteration $i > 2$ in Algorithm 1, we have $D_f(\delta^*, \delta^{(i)}) < \frac{1}{2} D_f(\delta^*, \delta^{(i-2)})$.*

In the remaining of this section, we apply the accelerated Newton–Dinkelbach method to linear fractional combinatorial optimization and linear fractional programming. The application to parametric submodular function minimization is in the full version.

2.1 Linear Fractional Combinatorial Optimization

The problem (1) with $\mathcal{D} \subseteq \{0,1\}^m$ is known as *linear fractional combinatorial optimization*. Radzik [18] showed that the Newton–Dinkelbach method applied to the function $f(\delta)$ as in (2) terminates in a strongly polynomial number of iterations. Recall that $f(\delta) = \min_{x \in \mathcal{D}} (c - \delta d)^\top x$. By the assumption $d^\top x > 0$ for all $x \in \mathcal{D}$, this function is concave, strictly decreasing, finite and piecewise-linear. Hence, it has a unique root. Moreover, $f(\delta) < 0$ and $\partial f(\delta) \cap \mathbb{R}_{<0} \neq \emptyset$ for sufficiently large δ. To implement the value and supergradient oracles, we assume that a linear optimization oracle over $\mathcal{D}$ is available, i.e. it returns an element in $\arg\min_{x \in \mathcal{D}} (c - \delta d)^\top x$ for any $\delta \in \mathbb{R}$.

Our result for the accelerated variant improves the state-of-the-art bound $O(m^2 \log m)$ by Wang et al. [25] on the standard Newton–Dinkelbach method. We will need the following lemma, given by Radzik and credited to Goemans in [19]. It gives a strongly polynomial bound on the length of a geometrically decreasing sequence of sums.

▶ **Lemma 2.5** ([19])**.** *Let $c \in \mathbb{R}^m_+$ and $x^{(1)}, x^{(2)}, \ldots, x^{(k)} \in \{-1, 0, 1\}^m$. If $0 < c^\top x^{(i+1)} \leq \frac{1}{2} c^\top x^{(i)}$ for all $i < k$, then $k = O(m \log m)$.*

▶ **Theorem 2.6.** *Algorithm 1 converges in $O(m \log m)$ iterations for linear fractional combinatorial optimization problems.*

Proof. Observe that Algorithm 1 terminates in a finite number of iterations because f is piecewise linear. Let $\delta^{(1)} > \delta^{(2)} > \cdots > \delta^{(k)} = \delta^*$ denote the sequence of iterates at the start of Algorithm 1. Since f is concave, we have $D_f(\delta^*, \delta^{(i)}) \geq 0$ for all $i \in [k]$. For each $i \in [k]$, pick $x^{(i)} \in \arg\min_{x \in \mathcal{D}} (c - \delta^{(i)} d)^\top x$ which maximizes $d^\top x$. This is well-defined because f is finite. Note that $-d^\top x^{(i)} = \min \partial f(\delta^{(i)})$. As $f(\delta^*) = 0$, the Bregman divergence of $\delta^{(i)}$ and δ^* can be written as

$$D_f(\delta^*, \delta^{(i)}) = f(\delta^{(i)}) + \max_{g \in \partial f(\delta^{(i)})} g(\delta^* - \delta^{(i)}) = (c - \delta^{(i)} d)^\top x^{(i)} - d^\top x^{(i)} (\delta^* - \delta^{(i)}) = (c - \delta^* d)^\top x^{(i)}.$$

According to Lemma 2.4, $(c - \delta^* d)^\top x^{(i)} = D_f(\delta^*, \delta^{(i)}) < \frac{1}{2} D_f(\delta^*, \delta^{(i-2)}) = \frac{1}{2} (c - \delta^* d)^\top x^{(i-2)}$ for all $3 \leq i \leq k$. By Lemma 2.3, we also know that $D_f(\delta^*, \delta^{(i)}) > 0$ for all $1 \leq i \leq k - 2$. Thus, applying Lemma 2.5 yields $k = O(m \log m)$. ◀

2.2 Linear Fractional Programming

We next consider *linear fractional programming*, an extension of (1) with the assumption that the domain $\mathcal{D} \subseteq \mathbb{R}^m$ is a polyhedron, but removing the condition $d^\top x > 0$ for $x \in \mathcal{D}$. For $c, d \in \mathbb{R}^m$, the problem is

$$\inf c^\top x / d^\top x \quad \text{s.t. } d^\top x > 0,\ x \in \mathcal{D}. \tag{F}$$

For the problem to be meaningful, we assume that $\mathcal{D} \cap \{x : d^\top x > 0\} \neq \emptyset$. The common form in the literature assumes $d^\top x > 0$ for all $x \in \mathcal{D}$ as in (1); we consider the more general setup for the purpose of solving M2VPI systems in Section 3. It is easy to see that any linear fractional combinatorial optimization problem on a domain $\mathcal{X} \subseteq \{0,1\}^m$ can be cast as a linear fractional program with the polytope $\mathcal{D} = \mathrm{conv}(\mathcal{X})$ because $c^\top \bar{x}/d^\top \bar{x} \geq \min_{x \in \mathcal{X}} c^\top x/d^\top x$ for all $\bar{x} \in \mathcal{D}$. The next theorem characterizes when (F) is unbounded.

▶ **Theorem 2.7.** *If $\mathcal{D} \cap \{x : d^\top x > 0\} \neq \emptyset$, then the optimal value of* (F) *is $-\infty$ if and only if at least one of the following two conditions hold:*

1. *There exists $x \in \mathcal{D}$ such that $c^\top x < 0$ and $d^\top x = 0$;*
2. *There exists $y \in \mathbb{R}^m$ such that $c^\top y < 0$, $d^\top y = 0$ and $x + \lambda y \in \mathcal{D}$ for all $x \in \mathcal{D}, \lambda \geq 0$.*

▶ **Example 2.8.** Unlike in linear programming, the optimal value may not be attained even if it is finite. Consider the instance given by $\inf(-x_1 + x_2)/(x_1 + x_2)$ subject to $x_1 + x_2 > 0$ and $-x_1 + x_2 = 1$. The numerator is equal to 1 for any feasible solution, while the denominator can be made arbitrarily large. Hence, the optimal value of this program is 0, which is not attained in the feasible region.

We use the Newton–Dinkelbach method for f as in (2), that is, $f(\delta) = \inf_{x \in \mathcal{D}} (c - \delta d)^\top x$. Since $\mathcal{D} \neq \emptyset$, $f(\delta) < \infty$ for all $\delta \in \mathbb{R}$. By the Minkowski–Weyl theorem, there exist finitely many points $P \subseteq \mathcal{D}$ such that $f(\delta) = \min_{x \in P} (c - \delta d)^\top x$ for all $\delta \in \mathrm{dom}(f)$. Hence, f is concave and piecewise linear. Observe that $f(\delta) > -\infty$ if and only if every ray y in the recession cone of $\mathcal{D}$ satisfies $(c - \delta d)^\top y \geq 0$. For f to be proper, we need to assume that Condition 2 in Theorem 2.7 does not hold. Moreover, we require the existence of a point $\delta' \in \mathrm{dom}(f)$ such that $f(\delta') = (c - \delta' d)^\top x' \leq 0$ for some $x' \in \mathcal{D}$ with $d^\top x' > 0$. It follows that f has a root or attains its maximum because $\mathrm{dom}(f)$ is closed. We are ready to characterize the optimal value of (F) using f.

▶ **Lemma 2.9.** *Assume that there exists $\delta' \in \mathrm{dom}(f)$ such that $f(\delta') = (c - \delta' d)^\top x' \leq 0$ for some $x' \in \mathcal{D}$ with $d^\top x' > 0$. If f has a root, then the optimal value of* (F) *is equal to the largest root and is attained. Otherwise, the optimal value is $-\infty$.*

3 Monotone Two Variables per Inequality Systems

Recall that an M2VPI system can be represented as a directed multigraph $G = (V, E)$ with arcs costs $c \in \mathbb{R}^m$ and gain factors $\gamma \in \mathbb{R}^m_{++}$. For a u-v walk P in G with $E(P) = (e_1, e_2, \ldots, e_k)$, its *cost* and *gain factor* are defined as $c(P) := \sum_{i=1}^k \left(\prod_{j=1}^{i-1} \gamma_{e_j} \right) c_{e_i}$ and $\gamma(P) := \prod_{i=1}^k \gamma_{e_i}$ respectively. If P is a single vertex, then $c(P) := 0$ and $\gamma(P) := 1$. The walk P induces the valid inequality $y_u \leq c(P) + \gamma(P) y_v$, implied by the sequence of arcs/inequalities in $E(P)$. It is also worth considering the dual interpretation. Dual variables on arcs correspond to generalized flows: if 1 unit of flow enter the arc $e = (u, v)$ at u, then γ_e units reach v, at a shipping cost of c_e. Thus, if 1 unit of flow enter a path P, then $\gamma(P)$ units reach the end of the path, incurring a cost of $c(P)$.

Given node labels $y \in \bar{\mathbb{R}}^n$, the *y-cost* of a u-v walk P is defined as $c(P) + \gamma(P) y_v$. Note that the y-cost of a walk only depends on the label at the sink. A u-v path is called a *shortest u-v path with respect to y* if it has the smallest y-cost among all u-v walks. A *shortest path from u with respect to y* is a shortest u-v path with respect to y for some node v. Such a path does not always exist, as demonstrated in the full version.

If P is a u-u walk such that its intermediate nodes are distinct, then it is called a *cycle at u*. Given a u-v walk P and a v-w walk Q, we denote PQ as the u-w walk obtained by concatenating P and Q.

▶ **Definition 3.1.** *A cycle C is called* flow-generating *if* $\gamma(C) > 1$, unit-gain *if* $\gamma(C) = 1$, *and* flow-absorbing *if* $\gamma(C) < 1$. *We say that a unit-gain cycle C is* negative *if* $c(C) < 0$.

Note that $c(C)$ depends on the starting point u of a cycle C. This ambiguity is resolved by using the term *cycle at* u. For a unit-gain cycle C, it is not hard to see that the starting point does not affect the sign of $c(C)$. Hence, the definition of a negative unit-gain cycle is sound. Observe that a flow-absorbing cycle C induces an upper bound $y_u \leq \frac{c(C)}{1-\gamma(C)}$, while a flow-generating cycle C induces a lower bound $y_u \geq \frac{-c(C)}{\gamma(C)-1}$. Let $\mathcal{C}_u^{abs}(G)$ and $\mathcal{C}_u^{gen}(G)$ denote the set of flow-absorbing cycles and flow-generating cycles at u in G respectively.

▶ **Definition 3.2.** *Given a flow-generating cycle C at u, a flow-absorbing cycle D at v, and a u-v path P, the walk CPD is called a* bicycle. *We say that the bicycle is* negative *if* $c(P) + \gamma(P)\frac{c(D)}{1-\gamma(D)} < \frac{-c(C)}{\gamma(C)-1}$.

Using these two structures, Shostak characterized the feasibility of M2VPI systems.

▶ **Theorem 3.3** ([21]). *An M2VPI system (G, c, γ) is infeasible if and only if G contains a negative unit-gain cycle or a negative bicycle.*

3.1 A Linear Fractional Programming Formulation

Our goal is to compute the pointwise maximal solution $y^{\max} \in \bar{\mathbb{R}}^n$ to an M2VPI system if it is feasible, where $y_u^{\max} := \infty$ if and only if the variable y_u is unbounded from above. It is well known how to convert $y^{\max}$ into a finite feasible solution – we refer to the full version for details. In order to apply Algorithm 1, we first need to reformulate the problem as a linear fractional program. Now, every coordinate $y_u^{\max}$ can be expressed as the following primal-dual pair of linear programs, where $\nabla x_v := \sum_{e\in\delta^+(u)} x_e - \sum_{e\in\delta^-(u)} \gamma_e x_e$ denotes the net flow at a node v.

$$\begin{aligned} &\min\ c^\top x && && (\mathrm{P}_u) \qquad &\max\ y_u && && (\mathrm{D}_u)\\ &\text{s.t. } \nabla x_u = 1 && && &\text{s.t. } y_v - \gamma_e y_w \leq c_e && \forall e=(v,w)\in E\\ &\quad \nabla x_v = 0 && \forall v \in V\setminus u && \\ &\quad x \geq 0 && && \end{aligned}$$

The primal LP (P_u) is a minimum-cost generalized flow problem with a supply of 1 at node u. It asks for the cheapest way to destroy one unit of flow at u. Observe that it is feasible if and only if u can reach a flow-absorbing cycle in G. If it is feasible, then it is unbounded if and only if there exists a negative unit-gain cycle or a negative bicycle in G. It can be reformulated as the following linear fractional program

$$\inf\ \frac{c^\top x}{1-\sum_{e\in\delta^-(u)}\gamma_e x_e} \quad \text{s.t. } 1 - \sum_{e\in\delta^-(u)} \gamma_e x_e > 0,\ x \in \mathcal{D}\,. \tag{F_u}$$

with the polyhedron

$$\mathcal{D} := \left\{x \in \mathbb{R}_+^m : x(\delta^+(u)) = 1, \nabla x_v = 0\ \forall v \in V\setminus u\right\}.$$

Indeed, if x is a feasible solution to (P_u), then $x/x(\delta^+(u))$ is a feasible solution to (F_u) with the same objective value. This is because $1-\sum_{e\in\delta^-(u)}\gamma_e x_e/x(\delta^+(u)) = 1/x(\delta^+(u))$. Conversely, if x is a feasible solution to (F_u), then $x/(1-\sum_{e\in\delta^-(u)}\gamma_e x_e)$ is a feasible solution to (P_u) with the same objective value. Even though the denominator is an affine function of x, it can be made linear to conform with (F) by working with the polyhedron $\{(x,1) : x \in \mathcal{D}\}$.

Our goal is to solve (F_u) using Algorithm 1. For a fixed $\delta \in \mathbb{R}$, the value of the parametric function $f(\delta)$ can be written as the following pair of primal and dual LPs respectively

$$\begin{aligned} \min\ & c^\top x + \delta \sum_{e\in\delta^-(u)} \gamma_e x_e - \delta \\ \text{s.t.}\ & x \in \mathcal{D} \end{aligned} \qquad\qquad \begin{aligned} \max\ & y_u - \delta \\ \text{s.t.}\ & y_v - \gamma_e \delta \le c_e \qquad \forall e = (v,u) \in \delta^-(u) \\ & y_v - \gamma_e y_w \le c_e \qquad \forall e = (v,w) \notin \delta^-(u). \end{aligned}$$

We refer to them as the *primal (resp. dual) LP for* $f(\delta)$, and their corresponding feasible/optimal solution as a *feasible/optimal primal (resp. dual) solution to* $f(\delta)$.

Due to the specific structure of this linear fractional program, a suitable initial point for the Newton–Dinkelbach method can be obtained from any feasible solution to (F_u). This is a consequence of the unboundedness test given by the following lemma.

▶ **Lemma 3.4.** *Let x be a feasible solution to (F_u) and $\bar\delta := c^\top x/(1 - \sum_{e\in\delta^-(u)} \gamma_e x_e)$. If either $f(\bar\delta) = -\infty$ or $f(\bar\delta) = c^\top \bar x - \bar\delta(1 - \sum_{e\in\delta^-(u)} \gamma_e \bar x_e) < 0$ for some $\bar x \in \mathcal{D}$ with $1 - \sum_{e\in\delta^-(u)} \gamma_e \bar x_e \le 0$, then the optimal value of (F_u) is $-\infty$.*

In order to characterize the finiteness of $f(\delta)$, we introduce the following notion of a negative flow-generating cycle.

▶ **Definition 3.5.** *For a fixed $\delta \in \mathbb{R}$ and $u \in V$, a flow-generating cycle C is said to be* (δ, u)-negative *if there exists a path P from a node $v \in V(C)$ to node u such that $c(C) + (\gamma(C) - 1)(c(P) + \gamma(P)\delta) < 0$, where C is treated as a v-v walk in $c(C)$.*

▶ **Lemma 3.6.** *For any $\delta \in \mathbb{R}$, $f(\delta) = -\infty$ if and only if $\mathcal{D} \neq \emptyset$ and there exists a negative unit-gain cycle, a negative bicycle, or a (δ, u)-negative flow-generating cycle in $G \setminus \delta^+(u)$.*

It turns out that if we have an optimal dual solution y to $f(\delta)$ for some $\delta \in \mathbb{R}$, then we can compute an optimal dual solution to $f(\delta')$ for any $\delta' < \delta$. A suitable subroutine for this task is the so called Grapevine algorithm (Algorithm 2), developed by Aspvall and Shiloach [2].

Algorithm 2 Grapevine.

input : A directed multigraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^m$ and gain factors $\gamma \in \mathbb{R}^m_{++}$, node labels $y \in \bar{\mathbb{R}}^n$, and a node $u \in V$.
output : Node labels $y \in \bar{\mathbb{R}}^n$ and a walk P of length at most n starting from u.

for $i = 1$ **to** n **do**
  foreach $v \in V$ **do**
    $y'_v \leftarrow \min(y_v, \min_{vw\in\delta^+(v)} c_{vw} + \gamma_{vw} y_w)$
    if $y'_v < y_v$ **then**
      $\mathrm{pred}(v, i) \leftarrow \arg\min_{vw\in\delta^+(v)} c_{vw} + \gamma_{vw} y_w$ `/* Break ties */`
    else
      $\mathrm{pred}(v, i) \leftarrow \emptyset$
  $y \leftarrow y'$
Let P be the walk obtained by tracing from $\mathrm{pred}(u, n)$
return (y, P)

Given initial node labels $y \in \bar{\mathbb{R}}^n$ and a specified node u, Grapevine runs for n iterations. We say that an arc $e = (v, w)$ is *violated with respect to* y if $y_v > c_e + \gamma_e y_w$. In an iteration $i \in [n]$, the algorithm records the most violated arc with respect to y in $\delta^+(v)$ as $\mathrm{pred}(v, i)$, for each node $v \in V$ (ties are broken arbitrarily). Note that $\mathrm{pred}(v, i) = \emptyset$ if there are

no violated arcs in $\delta^+(v)$. Then, each y_v is decreased by the amount of violation in the corresponding recorded arc. After n iterations, the algorithm traces a walk P from u by following the recorded arcs in reverse chronological order. During the trace, if $\mathrm{pred}(v,i) = \emptyset$ for some $v \in V$ and $i > 1$, then $\mathrm{pred}(v, i-1)$ is read. Finally, the updated node labels y and the walk P are returned. Clearly, the running time of Grapevine is $O(mn)$.

Given an optimal dual solution $y \in \mathbb{R}^n$ to $f(\delta)$ and $\delta' < \delta$, the dual LP for $f(\delta')$ can be solved using Grapevine as follows. Define the directed graph $G_u := (V \cup \{u'\}, E_u)$ where $E_u := (E \setminus \delta^-(u)) \cup \{vu' : vu \in \delta^-(u)\}$. The graph G_u is obtained from G by splitting u into two nodes u, u' and reassigning the incoming arcs of u to u'. These arcs inherit the same costs and gain factors from their counterparts in G. Let $\bar{y} \in \mathbb{R}^{n+1}$ be node labels in G_u defined by $\bar{y}_{u'} := \delta'$ and $\bar{y}_v := y_v$ for all $v \neq u'$. Then, we run Grapevine on G_u with input node labels $\bar{y}$ and node u. Note that $\bar{y}_{u'}$ remains unchanged throughout the algorithm. The next lemma verifies the correctness of this method.

▶ **Lemma 3.7.** *Given an optimal dual solution $y \in \mathbb{R}^n$ to $f(\delta)$ and $\delta' < \delta$, define $\bar{y} \in \mathbb{R}^{n+1}$ as $\bar{y}_{u'} := \delta'$ and $\bar{y}_v := y_v$ for all $v \in V$. Let $(\bar{z}, P)$ be the node labels and walk returned by* Grapevine$(G_u, \bar{y}, u)$. *If $\bar{z}_V$ is not feasible to the dual LP for $f(\delta')$, then $f(\delta') = -\infty$. Otherwise, $\bar{z}_V$ is a dual optimal solution to $f(\delta')$ and P is a shortest path from u with respect to $\bar{y}$ in G_u.*

If $\bar{z}_V$ is an optimal dual solution to $f(\delta')$, a supergradient in $\partial f(\delta')$ can be inferred from the returned path P. We say that an arc $e = (v, w)$ is *tight with respect to* $\bar{z}$ if $\bar{z}_v = c_e + \gamma_e \bar{z}_w$. By complementary slackness, every optimal primal solution to $f(\delta')$ is supported on the subgraph of G_u induced by tight arcs with respect to $\bar{z}$. In particular, any u-u' path or any path from u to a flow-absorbing cycle in this subgraph constitutes a basic optimal primal solution to $f(\delta')$. As P is also a path in this subgraph, we have $\gamma(P) - 1 \in \partial f(\delta')$ if P ends at u'. Otherwise, u can reach a flow-absorbing cycle in this subgraph because $\delta' < \delta$. In this case, $-1 \in \partial f(\delta')$.

3.2 A Strongly Polynomial Label-Correcting Algorithm

Using Algorithm 1, we develop a strongly polynomial label-correcting algorithm for solving an M2VPI system (G, c, γ). The main idea is to start with a subsystem for which (D_u) is trivial, and progressively solve (D_u) for larger and larger subsystems. Throughout the algorithm, we maintain node labels $y \in \bar{\mathbb{R}}^n$ which form valid upper bounds on each variable. They are initialized to ∞ at every node. We also maintain a subgraph of G, which initially is $G^{(0)} := (V, \emptyset)$.

The algorithm (Algorithm 3) is divided into n *phases*. At the start of phase $k \in [n]$, a new node $u \in V$ is selected and all of its outgoing arcs in G are added to $G^{(k-1)}$, resulting in a larger subgraph $G^{(k)}$. Since $y_u = \infty$ at this point, we update it to the smallest upper bound implied by its outgoing arcs and the labels of its outneighbours. If y_u is still infinity, then we know that $\delta^+(u) = \emptyset$ or $y_v = \infty$ for all $v \in N^+(u)$. In this case, we find a flow-absorbing cycle at u in $G^{(k)}$ using the multiplicative Bellman–Ford algorithm, by treating the gain factors as arc costs. If there is none, then we proceed to the next phase immediately as y_u is unbounded from above in the subsystem $(G^{(k)}, c, \gamma)$. This is because u cannot reach a flow-absorbing cycle in $G^{(k)}$ by induction. We would like to point out that this does not necessarily imply that the full system (G, c, γ) is feasible (see the full version for details). On the other hand, if Bellman–Ford returns a flow-absorbing cycle, then y_u is set to the upper bound implied by the cycle. Then, we apply Algorithm 1 to solve (D_u) for the subsystem $(G^{(k)}, c, \gamma)$.

Algorithm 3 Label-correcting algorithm for M2VPI systems.

input : An M2VPI system (G, c, γ).
output : The pointwise maximal solution $y^{\max}$ or the string `INFEASIBLE`.

Initialize graph $G^{(0)} \leftarrow (V, \emptyset)$ and counter $k \leftarrow 0$
Initialize node labels $y \in \bar{\mathbb{R}}^n$ as $y_v \leftarrow \infty \; \forall v \in V$
foreach $u \in V$ **do**
  $k \leftarrow k + 1$
  $G^{(k)} \leftarrow G^{(k-1)} \cup \delta^+(u)$
  $y_u \leftarrow \min_{uv \in \delta^+(u)} c_{uv} + \gamma_{uv} y_v$
  if $y_u = \infty$ **and** $\mathcal{C}_u^{abs}(G^{(k)}) \neq \emptyset$ **then**
    $y_u \leftarrow c(C)/(1 - \gamma(C))$ for any $C \in \mathcal{C}_u^{abs}(G^{(k)})$
  if $y_u < \infty$ **then**
    Define node labels $\bar{y} \in \bar{\mathbb{R}}^{n+1}$ as $\bar{y}_{u'} \leftarrow y_u$ and $\bar{y}_v \leftarrow y_v \; \forall v \in V$
    $(\bar{y}, P) \leftarrow$ Grapevine$(G_u^{(k)}, \bar{y}, u)$
    if $\exists$ a violated arc w.r.t. $\bar{y}$ in $G_u^{(k)}$ **or** $(|E(P)| > 0$ **and** $\gamma(P) \geq 1)$ **then**
      return `INFEASIBLE`
    $\bar{y}_{u'} \leftarrow$ Look-aheadNewton(Grapevine$(G_u^{(k)}, \cdot, u), \bar{y}_{u'}, \gamma(P) - 1)$
    if $\bar{y}_{u'} =$ `NO ROOT` **then**
      return `INFEASIBLE`
    $y \leftarrow \bar{y}_V$
return y

The value and supergradient oracle for the parametric function $f(\delta)$ is Grapevine. Let $G_u^{(k)}$ be the modified graph and $\bar{y} \in \bar{\mathbb{R}}^{n+1}$ be the node labels as defined in the previous subsection. In order to provide Algorithm 1 with a suitable initial point and supergradient, we run Grapevine on $G_u^{(k)}$ with input node labels $\bar{y}$. It updates $\bar{y}$ and returns a walk P from u. If $\bar{y}_V$ is not feasible to the dual LP for $f(\bar{y}_{u'})$ or P is a non-trivial walk with $\gamma(P) \geq 1$, then we declare infeasibility. Otherwise, we run Algorithm 1 with the initial point $\bar{y}_{u'}$ and supergradient $\gamma(P) - 1$. We remark that Grapevine continues to update $\bar{y}$ throughout the execution of Algorithm 1.

▶ **Theorem 3.8.** *If Algorithm 3 returns $y \in \bar{\mathbb{R}}^n$, then $y = y^{\max}$ if the M2VPI system is feasible. Otherwise, the system is infeasible.*

We would like to point out that Algorithm 3 may return node labels $y \in \bar{\mathbb{R}}^n$ even if the M2VPI system is infeasible. This happens when y contains ∞ entries. It is well-known how to ascertain the system's feasibility status in this case. We refer the reader to the full version for details.

To bound the running time of Algorithm 3, it suffices to bound the running time of Algorithm 1 in every phase. Our strategy is to analyze the sequence of paths whose gain factors determine the right derivative of f at each iterate of Algorithm 1. The next property is crucial in our arc elimination argument.

▶ **Definition 3.9.** *Let $\mathcal{P} = (P^{(1)}, P^{(2)}, \ldots, P^{(\ell)})$ be a sequence of paths from u. We say that $\mathcal{P}$ satisfies* subpath monotonicity at u *if for every pair $P^{(i)}, P^{(j)}$ where $i < j$ and for every shared node $v \neq u$, we have $\gamma(P_{uv}^{(i)}) \leq \gamma(P_{uv}^{(j)})$.*

▶ **Lemma 3.10.** *Let $\delta^{(1)} > \delta^{(2)} > \cdots > \delta^{(\ell)}$ be a decreasing sequence of iterates. For each $\delta^{(i)} \in \mathbb{R}$, let $P^{(i)}$ be a u-u' path in G_u on which a unit flow is an optimal primal solution to $f(\delta^{(i)})$. Then, the sequence $(P^{(1)}, P^{(2)}, \ldots, P^{(\ell)})$ satisfies subpath monotonicity at u.*

Proof. For each $i \in [\ell]$, let $y^{(i)} \in \mathbb{R}^n$ be an optimal dual solution to $f(\delta^{(i)})$. Let $\bar{y}^{(i)} \in \mathbb{R}^{n+1}$ be the node labels in G_u defined by $\bar{y}^{(i)}_{u'} := \delta^{(i)}$ and $\bar{y}_v := y_v$ for all $v \neq u'$. By complementary slackness, every arc in $P^{(i)}$ is tight with respect to $\bar{y}^{(i)}$. Hence, $P^{(i)}$ is a shortest u-u' path in G_u with respect to $\bar{y}^{(i)}$. Now, pick a pair of paths $P^{(i)}$ and $P^{(j)}$ such that $i < j$ and they share a node $v \neq u$. Then, the subpaths $P^{(i)}_{uv}$ and $P^{(j)}_{uv}$ are also shortest u-v paths in G_u with respect to $\bar{y}^{(i)}$ and $\bar{y}^{(j)}$ respectively. Observe that $\bar{y}^{(i)}_v > \bar{y}^{(j)}_v$ because $\bar{y}^{(i)}_{u'} = \delta^{(i)} > \delta^{(j)} = \bar{y}^{(j)}_{u'}$. Define the function $\psi : [\bar{y}^{(j)}_v, \bar{y}^{(i)}_v] \to \bar{\mathbb{R}}$ as

$$\psi(x) := \inf \{c(P) + \gamma(P)x : P \text{ is a } u\text{-}v \text{ walk in } G_u\}.$$

Clearly, it is increasing and concave. It is also finite because $\psi(\bar{y}^{(i)}_v) = c(P^{(i)}_{uv}) + \gamma(P^{(i)}_{uv})\bar{y}^{(i)}_v$ and $\psi(\bar{y}^{(j)}_v) = c(P^{(j)}_{uv}) + \gamma(P^{(j)}_{uv})\bar{y}^{(j)}_v$. Subpath monotonicity then follows from the concavity of ψ. ◀

▶ **Theorem 3.11.** *In each phase k of Algorithm 3, Algorithm 1 terminates in $O(|E(G^{(k)})|)$ iterations.*

The full proof is given in the full version; we highlight some key steps. Let $m_k := |E(G^{(k)})|$. Let $\bar{\mathcal{Y}} = (\bar{y}^{(1)}, \bar{y}^{(2)}, \ldots, \bar{y}^{(\ell)})$ be a sequence of node labels at the start of every iteration of Algorithm 1 in phase k. Let $\mathcal{P} = (P^{(2)}, P^{(3)}, \ldots, P^{(\ell)})$ be a sequence of u-u' paths in $G^{(k)}_u$ such that $P^{(i)}$ determines the right derivative $f'_+(\bar{y}^{(i)}_{u'})$ for all $i > 1$. Perturb $\hat{c} := c + \varepsilon\chi_{\delta^+(u)}$ by a suitably small $\varepsilon \geq 0$ such that the system $(G^{(k)}, \hat{c}, \gamma)$ is feasible. Let $y^* \in \bar{\mathbb{R}}^n$ be its pointwise maximal solution, and define the reduced cost $c^* \in \mathbb{R}^{m_k}_+$ as $c^*_{vw} := \hat{c}_{vw} + \gamma_{vw}y^*_w - y^*_v$ for all $vw \in E(G^{(k)})$. For every arc $vw \in \cup_{i=1}^{\ell} E(P^{(i)})$, let r_{vw} be the largest gain factor of the u-v subpath of the paths in $\mathcal{P}$ that contain vw. By Lemma 3.10, this is achieved by the last path in $\mathcal{P}$ which contains vw. Order the elements of $r \circ c^*$ as $0 \leq r_1c^*_1 \leq r_2c^*_2 \leq \cdots \leq r_{m_k}c^*_{m_k}$, and let $d_i := \sum_{j=1}^{i} r_jc^*_j$ for every $i \in [m_k]$. The final step is showing that every interval $(d_i, d_{i+1}]$ contains the cost of at most two paths from $\mathcal{P}$. This can be derived from the Bregman divergence analysis, that yields $c^*(P^{(i)}) \leq \frac{1}{2}c^*(P^{(i-2)})$ for all $3 \leq i \leq \ell$.

The runtime of every iteration of Algorithm 1 is dominated by Grapevine. Thus, we obtain the following result.

▶ **Corollary 3.12.** *Algorithm 3 solves the feasibility of M2VPI linear systems in $O(m^2n^2)$ time.*

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows - Theory, Algorithms and Applications.* Prentice Hall, 1993.

2 Bengt Aspvall and Yossi Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. *SIAM J. Comput.*, 9(4):827–845, 1980.

3 Edith Cohen and Nimrod Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23(6):1313–1347, 1994.

4 Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.

5 Herbert Edelsbrunner, Günter Rote, and Emo Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theor. Comput. Sci.*, 66(2):157–180, 1989.

6 Eugene A. Feinberg and Jefferson Huang. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. *Oper. Res. Lett.*, 42(2):130–131, 2014.

7 Michel X. Goemans, Swati Gupta, and Patrick Jaillet. Discrete Newton's algorithm for parametric submodular function minimization. In *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization*, pages 212–227, 2017.

8 Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.

9 Thomas Dueholm Hansen, Haim Kaplan, and Uri Zwick. Dantzig's pivoting rule for shortest paths, deterministic MDPs, and minimum cost to time ratio cycles. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 847–860, 2014.

10 Dorit S. Hochbaum, Nimrod Megiddo, Joseph Naor, and Arie Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Math. Program.*, 62:69–83, 1993.

11 Dorit S. Hochbaum and Joseph Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994.

12 Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence*, pages 394–402, 1995.

13 Omid Madani. On policy iteration as a Newton's method and polynomial policy iteration algorithms. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 273–278, 2002.

14 Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.

15 Nimrod Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12(2):347–353, 1983.

16 Neil Olver and László A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *J. ACM*, 67(2):10:1–10:26, 2020.

17 Ian Post and Yinyu Ye. The simplex method is strongly polynomial for deterministic Markov decision processes. *Math. Oper. Res.*, 40(4):859–868, 2015.

18 Tomasz Radzik. Newton's method for fractional combinatorial optimization. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992.

19 Tomasz Radzik. Fractional combinatorial optimization. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume 1–3*, pages 429–478. Springer US, 1998.

20 Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, 1994.

21 Robert E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981.

22 Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998.

23 Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.

24 László A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.*, 42(1):179–211, 2017.

25 Qin Wang, Xiaoguang Yang, and Jianzhong Zhang. A class of inverse dominant problems under weighted ℓ_∞ norm and an improved complexity bound for Radzik's algorithm. *J. Global Optimization*, 34(4):551–567, 2006.