An Accelerated Newton–Dinkelbach Method and its Application to Two Variables per Inequality Systems[*]

Daniel Dadush[1], Zhuan Khye Koh[2], Bento Natura[2], and László A. Végh[2]

[1]Centrum Wiskunde & Informatica, Netherlands.
[2]Department of Mathematics, London School of Economics, UK.

dadush@cwi.nl,{z.koh3,b.natura,l.vegh}@lse.ac.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).

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 [22] 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 [27] 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* [6]—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 [28] 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 3 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. [35] from 2006. We remark that Radzik's first analysis [27] yielded a bound of $O(m^4 \log^2 m)$ iterations, improved to $O(m^2 \log^2 m)$ in [28].

- Goemans et al. [10] 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 [31]. 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 [23] 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 [7, 13] (sketch in Appendix B.1).

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 [30] 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 of the Bellman-Ford algorithm—to decide whether the optimum ratio is smaller or larger than a fixed value δ. The optimum value can be found using binary search on the parameter.

Megiddo's strongly polynomial algorithm [23] replaced the binary search framework in Aspvall and Shiloach's algorithm by extending the parametric search technique in [22]. Subsequently, Cohen and Megiddo [3] devised faster strongly polynomial algorithms for the problem. The current fastest deterministic strongly polynomial algorithm is given by Hochbaum and Naor [14], an efficient Fourier–Motzkin elimination with running time of $O(mn^2 \log m)$. Recently, Karczmarz [18] gave a

randomized trade-off algorithm which runs in $\widetilde{O}(nmh + (n/h)^3)$ time and uses $\widetilde{O}(n^2/h + m)$ space for any parameter $h \in [1, n]$.

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 [27, 28] are not directly applicable, since the domain $\mathcal{D}$ in this setting is a polyhedron and not a discrete set.[1] Madani [21] 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 [32] on minimum-cost flows, and has been a central idea of strongly polynomial algorithms, see in particular [11, 29] for cycle cancelling minimum-cost flow algorithms, and [25, 34] 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 [14], 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.

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.

---

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

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 [30] 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 [20]. However, if the discount factors can be arbitrarily close to 1, then Feinberg and Huang [8] showed that value iteration cannot reveal the optimal policy in strongly polynomial time even for DMDPs. Post and Ye [26] proved that simplex with the most negative reduced cost pivoting rule is strongly polynomial for DMDPs; this was later improved by Hansen et al. [12]. 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 [21] 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 in Section 2. In Section 3, we present an accelerated Newton's method for univariate concave functions, and apply it to linear fractional combinatorial optimization and linear fractional programming. Section 4 contains our main application of the method to the 2VPI problem. Our results on parametric submodular function minimization are in Section 5.

---

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

2 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 $f : \mathbb{R} \rightarrow \bar{\mathbb{R}}$, 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.

3 An Accelerated Newton–Dinkelbach Method

Let $f : \mathbb{R} \rightarrow \bar{\mathbb{R}}$ 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 (see Figure 1 for examples). Therefore, the Newton–Dinkelbach method returns δ^* if f has a root, and certifies that $f(\delta^*) < 0$ otherwise.
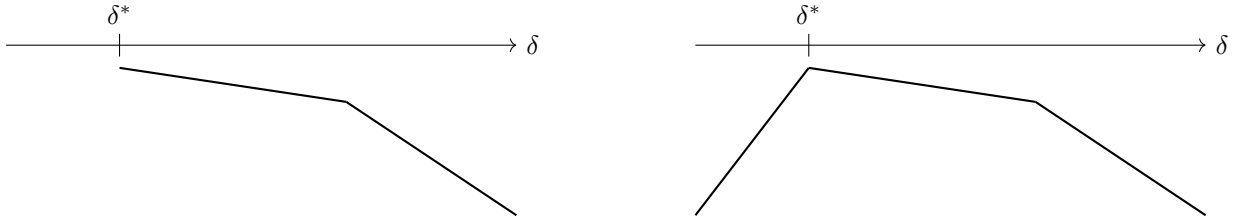

Figure 1: Two examples of f with no root. In the left picture, $f(\delta) = -\infty$ for all $\delta < \delta^*$.

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 (see Figure 2 for an example).

According to this update rule, observe that $g^{(i)} < 0$ except possibly in the final iteration when $f(\delta^{(i)}) = 0$. This proves the correctness of the algorithm. Indeed, $\delta^{(i)} = \delta^*$ if $f(\delta^{(i)}) = 0$. On the other hand, if either of the aforementioned scenarios apply, then combining it with $f(\delta^{(i)}) < 0$ and $g^{(i)} < 0$ certifies that $f(\delta^*) < 0$.

The following lemma shows 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. The lemma also illustrates the useful property that $|f(\delta^{(i)})|$ or $|g^{(i)}|$ decreases geometrically. These are well-known facts and similar statements can be found in e.g. Radzik [28, Lemmas 3.1 & 3.2]. The proof is given in Appendix A.
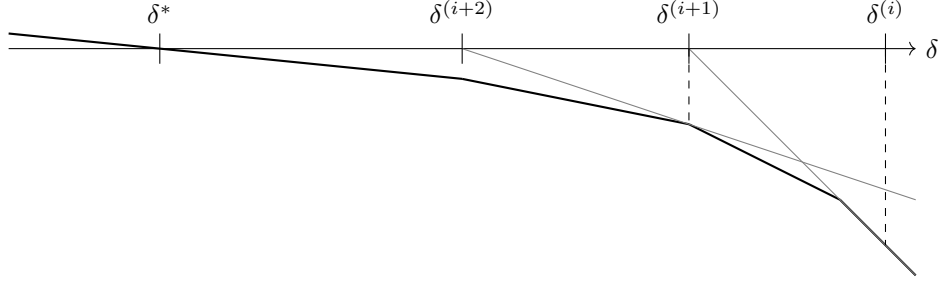
Figure 2: An example run of the Newton–Dinkelbach method when f has a root.

Lemma 3.1. 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 3.2. Given a proper concave function $f : \mathbb{R} \to \bar{\mathbb{R}}$, the *Bregman divergence associated with f* is defined as

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

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

Since f is concave, the Bregman divergence is nonnegative. See Figure 3 for an example. The next lemma shows that $D_f(\delta^*, \delta^{(i)})$ is monotonically decreasing except in the final iteration where it may remain unchanged. The proof is given in Appendix A.
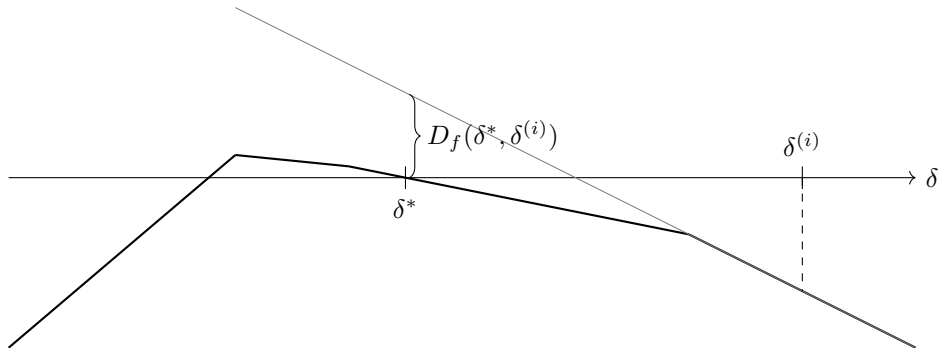

Figure 3: The Bregman divergence $D_f(\delta^*, \delta^{(i)})$ of an example function f.

Lemma 3.3. 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 $f(\delta^{(i)}) = 0$.

7

To accelerate this classical method, we perform an aggressive guess $\delta' := 2\delta - \delta^{(i)}$ on the next point at the end of every iteration i. Note that this is twice the usual Newton step, i.e., $\delta' = \delta^{(i)} + 2(\delta - \delta^{(i)}) < \delta$. 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 $g^{(i+1)} := g$ (see Figure 4 for an example). It is easy to verify that Lemmas 3.1 and 3.3 also hold for Algorithm 1.
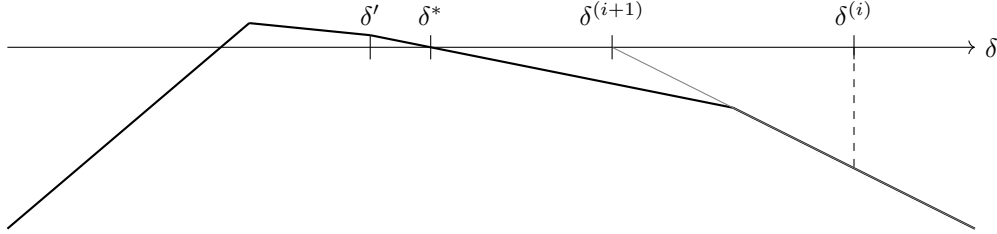

Figure 4: An example run of Algorithm 1 where look-ahead failed in iteration i.

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.

1 $i \leftarrow 1$
2 **while** $f(\delta^{(i)}) < 0$ **do**
3 $\delta \leftarrow \delta^{(i)} - f(\delta^{(i)})/g^{(i)}$
4 $g \in \partial f(\delta)$ /* Empty if $f(\delta) = -\infty$ */
5   **if** $f(\delta) = -\infty$ **or** $(f(\delta) < 0$ **and** $g \geq 0)$ **then**
6 $\lfloor$ **return** NO ROOT
7 $\delta' \leftarrow 2\delta - \delta^{(i)}$ /* Look-ahead guess */
8 $g' \in \partial f(\delta')$ /* Empty if $f(\delta') = -\infty$ */
9   **if** $-\infty < f(\delta') < 0$ **and** $g' < 0$ **then**   /* Is the guess successful? */
10 $\lfloor$ $\delta \leftarrow \delta', g \leftarrow g'$
11 $\delta^{(i+1)} \leftarrow \delta, g^{(i+1)} \leftarrow g$
12 $i \leftarrow i + 1$
13 **return** $\delta^{(i)}$

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

Proof. Fix an iteration $i > 2$ of Algorithm 1. Let $g_+^{(i)} = \min_{g \in \partial f(\delta^{(i)})} g$ denote the right derivative of f at $\delta^{(i)}$. From Lemma 3.1, we know that $\delta^* \leq \delta^{(i)} < \delta^{(i-1)} < \delta^{(i-2)}$, $0 \geq f(\delta^*) \geq f(\delta^{(i)}) >$

$f(\delta^{(i-1)}) > f(\delta^{(i-2)})$ and $0 > g_+^{(i)} \geq g^{(i-1)} > g^{(i-2)}$. Since $\delta^* \leq \delta^{(i)}$, we see that $D_f(\delta^*, \delta^{(i)}) = f(\delta^{(i)}) + g_+^{(i)}(\delta^* - \delta^{(i)}) - f(\delta^*)$.

Assume first that the look-ahead step in iteration $i-1$ was successful. We now claim that $0 < -2g_+^{(i)} \leq -g^{(i-1)}$. To see this, we have that

$$
\begin{aligned}
f(\delta^{(i-1)}) &\leq f(\delta^{(i)}) + g_+^{(i)}(\delta^{(i-1)} - \delta^{(i)}) \quad \text{(by concavity of } f \text{)} \\
&\leq g_+^{(i)}(\delta^{(i-1)} - \delta^{(i)}) \quad \left(\text{since } f(\delta^{(i)}) \leq 0 \right) \\
&= 2g_+^{(i)} \frac{f(\delta^{(i-1)})}{g^{(i-1)}} \quad \text{(by definition of the accelerated step)}.
\end{aligned}
$$

The desired inequality follows by multiplying through by $-\frac{g^{(i-1)}}{f(\delta^{(i-1)})} < 0$.

Using the above inequality, we compare Bregman divergences as follows:

$$
\begin{aligned}
D_f(\delta^*, \delta^{(i-1)}) &\geq f(\delta^{(i-1)}) + g^{(i-1)}(\delta^* - \delta^{(i-1)}) - f(\delta^*) \quad \text{(since } D_f \text{ is a maximum over supergradients)} \\
&> g^{(i-1)}(\delta^* - \delta^{(i)}) - f(\delta^*) \quad \left(f(\delta^{(i-1)}) + g^{(i-1)}(\delta^{(i)} - \delta^{(i-1)}) = -f(\delta^{(i-1)}) > 0 \right) \\
&\geq g^{(i-1)}(\delta^* - \delta^{(i)}) \quad (-f(\delta^*) \geq 0) \\
&\geq 2g_+^{(i)}(\delta^* - \delta^{(i)}) \quad \left(-g^{(i-1)} \geq -2g_+^{(i)} \text{ and } \delta^{(i)} > \delta^* \right) \\
&\geq 2(f(\delta^{(i)}) + g_+^{(i)}(\delta^* - \delta^{(i)}) - f(\delta^*)) \quad \left(\text{since } f(\delta^*) \geq f(\delta^{(i)}) \right) \\
&= 2D_f(\delta^*, \delta^{(i)}) \quad \left(\text{by our choice of } g_+^{(i)} \right).
\end{aligned}
$$

The desired inequality nows follows from $D_f(\delta^*, \delta^{(i-2)}) > D_f(\delta^*, \delta^{(i-1)})$ by Lemma 3.3.

Now assume that the look-ahead step at iteration $i-1$ was unsuccessful. This implies that $2\delta^{(i)} - \delta^{(i-1)} \leq \delta^* \Leftrightarrow 2(\delta^{(i)} - \delta^*) \leq \delta^{(i-1)} - \delta^*$, i.e. that the look-ahead step "went past or exactly to" δ^*. We compare Bregman-divergences as follows:

$$
\begin{aligned}
D_f(\delta^*, \delta^{(i-2)}) &\geq f(\delta^{(i-2)}) + g^{(i-2)}(\delta^* - \delta^{(i-2)}) - f(\delta^*) \quad \text{(since } D_f \text{ is a maximum over supergradients)} \\
&\geq g^{(i-2)}(\delta^* - \delta^{(i-1)}) - f(\delta^*) \quad \left(f(\delta^{(i-2)}) + g^{(i-2)}(\delta^{(i-1)} - \delta^{(i-2)}) \geq 0 \right) \\
&\geq g^{(i-2)}(\delta^* - \delta^{(i-1)}) \quad (-f(\delta^*) \geq 0) \\
&> g_+^{(i)}(\delta^* - \delta^{(i-1)}) \quad \left(0 > g_+^{(i)} > g^{(i-2)} \text{ and } \delta^{(i-1)} > \delta^* \right) \\
&\geq 2g_+^{(i)}(\delta^* - \delta^{(i)}) \quad \left(0 > g_+^{(i)} \text{ and } \delta^{(i-1)} - \delta^* \geq 2(\delta^{(i)} - \delta^*) \right) \\
&\geq 2(f(\delta^{(i)}) + g_+^{(i)}(\delta^* - \delta^{(i)}) - f(\delta^*)) \quad \left(\text{since } f(\delta^*) \geq f(\delta^{(i)}) \right) \\
&= 2D_f(\delta^*, \delta^{(i)}) \quad \left(\text{by our choice of } g_+^{(i)} \right).
\end{aligned}
$$

This concludes the proof. $\qquad\square$

Remark 3.5. Instead of taking twice the usual Newton step during look-ahead, one could consider $\delta' := \delta^{(i)} + \alpha(\delta - \delta^{(i)})$ for any $\alpha > 1$. By redoing the proof of Lemma 3.4 with this choice of δ', one gets $D_f(\delta^*, \delta^{(i-2)}) \geq \alpha D_f(\delta^*, \delta^{(i)})$ if look-ahead was successful in iteration $i-1$, and $D_f(\delta^*, \delta^{(i-2)}) \geq \frac{\alpha}{\alpha-1} D_f(\delta^*, \delta^{(i)})$ if look-ahead failed in iteration $i-1$. So, choosing $\alpha = 2$ balances the decay in Bregman divergence for both cases.

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 Section 5.

3.1 Linear Fractional Combinatorial Optimization

The problem (1) with $\mathcal{D} \subseteq \{0,1\}^m$ is known as *linear fractional combinatorial optimization*. Radzik [27] 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. [35] on the standard Newton–Dinkelbach method. We will need the following lemma, given by Radzik and credited to Goemans in [28]. It gives a strongly polynomial bound on the length of a geometrically decreasing sequence of sums.

Lemma 3.6 ([28]). *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 3.7. *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 3.4, $(c - \delta^* d)^\top x^{(i)} = D_f(\delta^*, \delta^{(i)}) < \frac{1}{2} D_f(\delta^*, \delta^{(i-2)}) = \frac{1}{2}(c - \delta^* d)^\top x^{(i-2)}$ for all $3 \leq i \leq k$. By Lemma 3.3, we also know that $D_f(\delta^*, \delta^{(i)}) > 0$ for all $1 \leq i \leq k - 2$. Thus, applying Lemma 3.6 yields $k = O(m \log m)$. $\qquad\square$

3.2 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 4. 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 $\mathcal{D} = \text{conv}(\mathcal{X})$ because $c^\top \bar{x} / d^\top \bar{x} \geq \min_{x \in \mathcal{X}} c^\top x / d^\top x$ for all $\bar{x} \in \mathcal{D}$ by the median inequality. The next theorem characterizes when (F) is unbounded.

Theorem 3.8. *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 $r \in \mathbb{R}^m$ such that $c^\top r < 0$, $d^\top r = 0$ and $x + \lambda r \in \mathcal{D}$ for all $x \in \mathcal{D}, \lambda \geq 0$.*

Proof. By the Minkowski-Weyl Theorem, the polyhedron $\bar{\mathcal{D}} := \mathcal{D} \cap \{x : d^\top x \geq 0\}$ can be written as

$$\bar{\mathcal{D}} = \left\{ \sum_{i=1}^k \lambda_i g_i + \sum_{j=1}^\ell \nu_j h_j : \lambda \geq 0, \nu \geq 0, \|\lambda\|_1 = 1 \right\}$$

for some vectors $g_1, \ldots, g_k$ and $h_1, \ldots, h_\ell$. Note that $d^\top g_i \geq 0$ for all $i \in [k]$ and $d^\top h_j \geq 0$ for all $j \in [\ell]$. Let $x^\circ \in \mathcal{D} \cap \{x : d^\top x > 0\}$. If there exists $i \in [k]$ such that $c^\top g_i < 0$ and $d^\top g_i = 0$ or $j \in [\ell]$ such that $c^\top h_j < 0$ and $d^\top h_j = 0$, then,

$$\lim_{\lambda \nearrow 1} \frac{c^\top (\lambda g_i + (1-\lambda)x^\circ)}{d^\top (\lambda g_i + (1-\lambda)x^\circ)} = -\infty \quad \text{or} \quad \lim_{\lambda \to \infty} \frac{c^\top (x^\circ + \lambda h_j)}{d^\top (x^\circ + \lambda h_j)} = -\infty$$

as in Condition 1 or Condition 2.

Otherwise, the fractional value of any element in $\mathcal{D} \cap \{x : d^\top x > 0\}$ can be lower bounded by

$$\frac{c^\top (\sum_{i=1}^k \lambda_i g_i + \sum_{j=1}^\ell \nu_j h_j)}{d^\top (\sum_{i=1}^k \lambda_i g_i + \sum_{j=1}^\ell \nu_j h_j)} \geq \frac{\sum_{i \in [k], d^\top g_i > 0} \lambda_i c^\top g_i + \sum_{j \in [\ell], d^\top h_j > 0} \nu_j c^\top h_j}{\sum_{i \in [k], d^\top g_i > 0} \lambda_i d^\top g_i + \sum_{j \in [\ell], d^\top h_j > 0} \nu_j d^\top h_j}$$

$$\geq \min \left\{ \min_{i \in [k], d^\top g_i > 0} \frac{c^\top g_i}{d^\top g_i}, \min_{j \in [\ell], d^\top h_j > 0} \frac{c^\top h_j}{d^\top h_j} \right\},$$

where the last expression is finite by the assumption that $\mathcal{D} \cap \{x : d^\top x > 0\}$ is non-empty. □

Example 3.9. 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 $\delta \in \text{dom}(f)$. Hence, f is concave and piecewise linear. Observe that $f(\delta) > -\infty$ if and only if every ray r in the recession cone of $\mathcal{D}$ satisfies $(c - \delta d)^\top r \geq 0$. For f to be proper, we need to assume that Condition 2 in Theorem 3.8 does not hold. Moreover, we require the existence of a point $\delta' \in \text{dom}(f)$ 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 $\text{dom}(f)$ is closed. We are ready to characterize the optimal value of (F) using f.

Lemma 3.10. *Assume that there exists $\delta' \in \text{dom}(f)$ 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 $-\infty$.*

Proof. Recall the definition of $\delta^* = \max(\{\delta : f(\delta) = 0\} \cup \arg\max f(\delta))$. By our assumption on f, there exists $x^* \in \mathcal{D}$ such that $f(\delta^*) = (c - \delta^* d)^\top x^*$ and $d^\top x^* > 0$. If f has a root, then $f(\delta^*) = 0$. This implies that $c^\top x / d^\top x \geq \delta^* = c^\top x^* / d^\top x^*$ for all $x \in \mathcal{D}$ with $d^\top x > 0$ as desired. Next, assume that f does not have a root. Then $f(\delta^*) < 0$ and $0 \in \partial f(\delta^*)$. By convexity, there exists $\bar{x} \in \mathcal{D}$ such that $(c - \delta^* d)^\top \bar{x} = f(\delta^*) < 0$ and $d^\top \bar{x} = 0$. Then $c^\top \bar{x} < 0$, so $\bar{x}$ is a point as in Condition 1 of Theorem 3.8. $\qquad \square$

4 Monotone Two Variables per Inequality Systems

Recall that an M2VPI system can be represented as a directed multigraph $G = (V, E)$ with arc 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 Appendix B.2.

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 4.1. 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 $y_u \leq c(C)/(1 - \gamma(C))$, while a flow-generating cycle C induces a lower bound $y_u \geq -c(C)(\gamma(C) - 1)$. 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 4.2. 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 4.3 ([30])**.** *An M2VPI system (G, c, γ) is infeasible if and only if G contains a negative unit-gain cycle or a negative bicycle.*

12

4.1 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 Appendix B.3 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 $\nabla x_v := \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} \gamma_e x_e$ denotes the net flow at a node v.

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

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

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

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

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

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 4.4. *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 $1 - \sum_{e \in \delta^-(u)} \gamma_e \bar{x}_e \leq 0$, then the optimal value of (F_u) is $-\infty$.*

Proof. First, assume that $f(\bar{\delta}) > -\infty$. Let $\lambda := (1 - \sum_{e \in \delta^-(u)} \gamma_e x_e) / \sum_{e \in \delta^-(u)} \gamma_e (\bar{x}_e - x_e)$. Note that $\lambda \in (0, 1]$. Consider the convex combination $\hat{x} := \lambda \bar{x} + (1 - \lambda) x \in \mathcal{D}$. Then, $c^\top \hat{x} < 0$ and $1 - \sum_{e \in \delta^-(u)} \gamma_e \hat{x}_e = 0$. Hence, the optimal value of (F_u) is unbounded by Condition 1 of Theorem 3.8. Next, assume that $f(\bar{\delta}) = -\infty$. There exists a ray r in the recession cone of $\mathcal{D}$ such that $c^\top r - \bar{\delta} \sum_{e \in \delta^-(u)} \gamma_e r_e < 0$. Note that $r \geq 0$. If $r(\delta^-(u)) = 0$, then r satisfies Condition 2 of Theorem 3.8. So, the optimal value is unbounded. Otherwise, for a sufficiently large $\alpha > 0$, we have $c^\top (x + \alpha r) + \bar{\delta}(1 - \sum_{e \in \delta^-(u)} \gamma_e (x_e + \alpha r_e)) < 0$ and $1 - \sum_{e \in \delta^-(u)} \gamma_e (x_e + \alpha r_e) < 0$. Then, taking an appropriate convex combination of $x + \alpha r$ and x like before produces a point in $\mathcal{D}$ which satisfies Condition 1 of Theorem 3.8. $\qquad\square$

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

Definition 4.5. 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 4.6. *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 $G \setminus \delta^+(u)$.*

Proof. The primal LP for $f(\delta)$ is unbounded if and only if $\mathcal{D} \neq \emptyset$ and there exists an extreme ray r in the recession cone of $\mathcal{D}$ such that $c^\top r + \delta \sum_{e \in \delta^-(u)} \gamma_e r_e < 0$. Note that the recession cone of $\mathcal{D}$ is $\{x \in \mathbb{R}^m_+ : x(\delta^+(u)) = 0, \nabla x_v = 0 \; \forall v \neq u\}$. By the generalized flow decomposition theorem, r belongs to one of the following three fundamental flows in $G \setminus \delta^+(u)$: (1) a unit-gain cycle, (2) a bicycle, (3) a flow-generating cycle C and a path P from C to u. In the first two cases, $r_e = 0$ for all $e \in \delta^-(u)$. Thus, the unit-gain cycle or bicycle is negative. In the last case, we have $c(C) + (\gamma(C) - 1)(c(P) + \gamma(P)\delta) = c^\top r + \delta \sum_{e \in \delta^-(u)} \gamma_e r_e$. $\qquad\square$

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.

1 **for** $i = 1$ **to** n **do**
2   **foreach** $v \in V$ **do**
3 $y'_v \leftarrow \min(y_v, \min_{vw \in \delta^+(v)} c_{vw} + \gamma_{vw} y_w)$
4   **if** $y'_v < y_v$ **then**
5 $\mathrm{pred}(v, i) \leftarrow \arg\min_{vw \in \delta^+(v)} c_{vw} + \gamma_{vw} y_w$ /* Break ties */
6   **else**
7 $\mathrm{pred}(v, i) \leftarrow \emptyset$
8 $y \leftarrow y'$
9 Let P be the walk obtained by tracing from $\mathrm{pred}(u, n)$
10 **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 4.7. *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.*

Proof. Since $f(\delta) = y_u - \delta$ is finite, we have $\mathcal{D} \neq \emptyset$. First, assume that $\bar{z}_V$ is not feasible to the dual LP for $f(\delta')$. Then, there exists a violated arc in G_u with respect to $\bar{z}$. Let w be the head of this arc and let R be the walk obtained by tracing $\mathrm{pred}(w, n)$ in reverse chronological order. Then, R ends at u' because y is dual feasible to $f(\delta)$. Since R has n edges, decompose it into $R = QCP'$ where Q is a w-v walk, C is a nontrivial cycle at v, and P' is a v-u' path for some node v. Then, we have $c(CP') + \gamma(CP')\delta' < c(P') + \gamma(P')\delta' \leq \bar{y}_v$. Due to Lemma 4.6, it suffices to show that $\gamma(C) > 1$, as this would imply that C is a (δ', u)-negative flow-generating cycle in G. Suppose otherwise for a contradiction. Since y is dual feasible to $f(\delta)$ and $u' \notin V(C)$, we have $\bar{y}_v \leq c(C) + \gamma(C)\bar{y}_v$. If $\gamma(C) = 1$, then we obtain $0 \leq c(C) < 0$ from the previous two inequalities. Otherwise, we get the following contradiction

$$\bar{y}_v \leq \frac{c(C)}{1 - \gamma(C)} < c(P') + \gamma(P')\delta' \leq \bar{y}_v.$$

Next, assume that $\bar{z}_V$ is a dual feasible solution to $f(\delta')$. Then, P is a u-t path for some node t. This is because if P is not simple, repeating the argument from the previous paragraph proves that the dual LP for $f(\delta')$ is infeasible. Note that $\bar{y}_t = \bar{z}_t$. Moreover, $\bar{z}_v \leq c_{vw} + \gamma_{vw}\bar{z}_w$ for all $vw \in E_u$, with equality on $E(P)$. Let $c^{\bar{z}} \in \mathbb{R}_+^m$ be the reduced cost defined by $c_{vw}^{\bar{z}} := c_{vw} + \gamma_{vw}\bar{z}_w - \bar{z}_v$ for all $vw \in E_u$. Since for every u-t walk P' we have

$$c(P) + \gamma(P)\bar{z}_t - \bar{z}_u = c^{\bar{z}}(P) = 0 \leq c^{\bar{z}}(P') = c(P') + \gamma(P')\bar{z}_t - \bar{z}_u,$$

it follows that P is a shortest u-t path with respect to $\bar{y}$.

It is left to show that $\bar{z}_V$ is a dual optimal solution to $f(\delta')$. Let z^* be an optimal dual solution to $f(\delta')$. Note that $z_u^* \leq y_u$ because $\delta' < \delta$. For the purpose of contradiction, suppose that $\bar{z}_u < z_u^*$. Since $\bar{z}_u < \bar{y}_u$, the path P ends at u' because y is dual feasible to $f(\delta)$. Thus, $\bar{z}_u = c(P) + \gamma(P)\delta'$. However, P also implies the valid inequality $z_u^* \leq c(P) + \gamma(P)\delta'$, which is a contradiction. $\qquad\square$

15

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, $-1 \in \partial f(\delta')$.

4.2 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 $G^{(0)} := (V, \emptyset)$.

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.

1 Initialize graph $G^{(0)} \leftarrow (V, \emptyset)$ and counter $k \leftarrow 0$
2 Initialize node labels $y \in \bar{\mathbb{R}}^n$ as $y_v \leftarrow \infty \; \forall v \in V$
3 foreach $u \in V$ **do**
4 $k \leftarrow k + 1$
5 $G^{(k)} \leftarrow G^{(k-1)} \cup \delta^+(u)$
6 $y_u \leftarrow \min_{uv \in \delta^+(u)} c_{uv} + \gamma_{uv} y_v$
7   **if** $y_u = \infty$ **and** $\mathcal{C}_u^{abs}(G^{(k)}) \neq \emptyset$ **then**
8 $y_u \leftarrow c(C)/(1 - \gamma(C))$ for any $C \in \mathcal{C}_u^{abs}(G^{(k)})$
9   **if** $y_u < \infty$ **then**
10   Define node labels $\bar{y} \in \bar{\mathbb{R}}^{n+1}$ as $\bar{y}_{u'} \leftarrow y_u$ and $\bar{y}_v \leftarrow y_v \; \forall v \in V$
11 $(\bar{y}, P) \leftarrow \text{GRAPEVINE}(G_u^{(k)}, \bar{y}, u)$
12   **if** $\exists$ a violated arc w.r.t. $\bar{y}$ in $G_u^{(k)}$ **or** $(|E(P)| > 0$ **and** $\gamma(P) \geq 1)$ **then**
13   **return** INFEASIBLE
14 $\bar{y}_{u'} \leftarrow \text{LOOK-AHEADNEWTON}(\text{GRAPEVINE}(G_u^{(k)}, \cdot, u), \bar{y}_{u'}, \gamma(P) - 1)$
15   **if** $\bar{y}_{u'} = \text{NO ROOT}$ **then**
16   **return** INFEASIBLE
17 $y \leftarrow \bar{y}_V$

18 return y

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

16

$(G^{(k)}, c, \gamma)$. 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 Appendix B.3 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 $(G^{(k)}, c, \gamma)$.

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 4.8. *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.*

Proof. It suffices to prove the theorem for the subsystem $(G^{(k)}, c, \gamma)$ encountered in each phase k. We proceed by induction on k. For the base case $k = 0$, the system $(G^{(0)}, c, \gamma)$ is trivially feasible as it does not have any constraints. Hence, $y^{\max} = (\infty, \infty, \ldots, \infty) = y$, where the second equality is due to our initialization. For the inductive step, assume that the theorem is true for some $0 \leq k < n$ and consider the system $(G^{(k+1)}, c, \gamma)$. If Algorithm 3 terminated in phase k, then $(G^{(k+1)}, c, \gamma)$ is infeasible by the inductive hypothesis. So, let $y \in \bar{\mathbb{R}}^n$ be the node labels maintained by the algorithm during Line 9 of phase $k + 1$. We have $y_u = \infty$ if and only if $\mathcal{C}_u^{abs}(G^{(k+1)}) = \emptyset$ and $y_v = \infty$ for all $v \in N^+(u)$. For each $v \neq u$, we also have $y_v = \infty$ if and only if v cannot reach a flow-absorbing cycle in $G^{(k)}$. So, if $y_u = \infty$, then u cannot reach a flow-absorbing cycle in $G^{(k+1)}$. By the inductive hypothesis, $y = y^{\max}$ if the system $(G^{(k+1)}, c, \gamma)$ is feasible.

Next, assume that $y_u < \infty$. Without loss of generality, we may assume that every node v with $y_v = \infty$ can reach u in $G^{(k+1)}$. Let $W := \{v \in V : y_v = \infty\}$. Note that the cut W does not have any outgoing edges in $G^{(k+1)}$. If there exists a negative unit-gain cycle in $G^{(k+1)}[W]$, then it contains a violated arc with respect to any finite labels. In this case, the algorithm correctly detects infeasibility. Otherwise, by Lemma 4.6, $f(\delta') > -\infty$ for a sufficiently high $\delta' \in \mathbb{R}$ because there are no flow-absorbing cycles in $G^{(k+1)}[W]$. Pick $\delta' > y_u$ big enough such that an optimal dual solution $y' \in \mathbb{R}^n$ to $f(\delta')$ satisfies $y'_v = y_v$ for all $v \in V \setminus W$. Among all such optimal dual solutions, choose y' as the pointwise maximal one. Then, every vertex $v \in W$ has a tight path to u in $G^{(k+1)}$. Now, let $\bar{y}' \in \mathbb{R}^{n+1}$ be node labels defined by $\bar{y}'_{u'} := y_u$ and $\bar{y}'_v := y'_v$ for all $v \in V$. It is easy to see that running GRAPEVINE on $G_u^{(k+1)}$ with input node labels $\bar{y}$ and $\bar{y}'$ yields the same behaviour. Let $(\bar{z}, P)$ be the node labels and walk returned by GRAPEVINE.

Let $x \in \mathbb{R}_+^{E(G^{(k+1)})}$ be a feasible solution to (F_u) such that $y_u = c^\top x / (1 - \sum_{e \in \delta^-(u)} \gamma_e x_e)$. Clearly, such an x exists if $y_u = c(C)/1 - \gamma(C)$ for some flow-absorbing cycle $C \in \mathcal{C}_u^{abs}(G^{(k+1)})$. Otherwise, if $y_u = c_{uv} + \gamma_{uv} y_v$ for some $uv \in \delta^+(u)$, then $y_v = c(Q) + \gamma(Q)(c(C)/1 - \gamma(C))$ where Q is a path leading to a flow-absorbing cycle C in $G^{(k)}[V \setminus W]$. This is because $y_{V \setminus W}$ is the pointwise maximal solution to the feasible subsystem $(G^{(k)}[V \setminus W], c, \gamma)$ by the inductive hypothesis. Hence, x can be chosen as the fundamental flow from u to the cycle C via the path $Q + uv$.

Now, according to Lemma 4.7, if $\bar{z}_V$ is not feasible to the dual LP for $f(y_u)$, then $f(y_u) = -\infty$. By Lemma 4.4, the optimal value of (F_u) is $-\infty$. On the other hand, if $\bar{z}_V$ is a feasible solution to the dual LP for $f(y_u)$, then it is also optimal. Moreover, P is a shortest path from u with respect to $\bar{y}'$ in $G_u^{(k+1)}$. If $|E(P)| > 0$ and $\gamma(P) \geq 1$, then the path ends at u' because $\bar{y}'$ is dual feasible to $f(\delta')$. Let $\bar{x}$ be the fundamental u-u' flow on P. By complementary slackness, $\bar{x}$ is an optimal

primal solution to $f(y_u) < 0$ and $1 - \sum_{e \in \delta^-(u)} \gamma_e \bar{x}_e = 1 - \gamma(P) \leq 0$. Applying Lemma 4.4 again yields unboundedness of (F_u). In both cases, as (P_u) is feasible, $(G^{(k+1)}, c, \gamma)$ is infeasible.

If the above cases do not apply, then $\bar{z}_u$ and $\gamma(P) - 1$ constitute a suitable initial point and supergradient for Algorithm 1 respectively. Note that the node labels $\bar{y}$ are updated to $\bar{z} \in \mathbb{R}^{n+1}$. Throughout the execution of Algorithm 1, it is easy to see that $\bar{y}_V$ remains an upper bound on every feasible solution to the system $(G^{(k+1)}, c, \gamma)$. If phase $k + 1$ terminates with node labels $y := \bar{y}_V$, then y_u is the largest root of f. By Lemma 3.10, y_u is the optimal value of (F_u). Since y is an optimal solution to (D_u), we obtain $y = y^{\max}$ as desired. On the other hand, if phase $k + 1$ terminates with INFEASIBLE, then f does not have a root. By Lemma 3.10, the optimal value of (F_u) is $-\infty$. As (P_u) is feasible, this implies that $(G^{(k+1)}, c, \gamma)$ is infeasible. $\square$

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 (see Appendix B.3 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 4.9. 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 4.10. *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 $\bar{y}_{u'}^{(i)} := \delta^{(i)}$ and $\bar{y}_v^{(i)} := y_v^{(i)}$ for all $v \neq u'$. By complementary slackness, every edge 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 $P_{uv}^{(i)}$ and $P_{uv}^{(j)}$ are also shortest u-v paths in G_u with respect to $\bar{y}^{(i)}$ and $\bar{y}^{(j)}$ respectively. Observe that $\bar{y}_v^{(i)} > \bar{y}_v^{(j)}$ because $\bar{y}_{u'}^{(i)} = \delta^{(i)} > \delta^{(j)} = \bar{y}_{u'}^{(j)}$. Define the function $\psi : [\bar{y}_v^{(j)}, \bar{y}_v^{(i)}] \to \bar{\mathbb{R}}$ as

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

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

Theorem 4.11. *In each phase k of Algorithm 3, Algorithm 1 terminates in $O(|E(G^{(k)})|)$ iterations.*

Proof. Fix a phase $k \in [n]$ and denote $m_k := |E(G^{(k)})|$. Let $\bar{\mathcal{Y}} = (\bar{y}^{(1)}, \bar{y}^{(2)}, \ldots, \bar{y}^{(\ell)})$ be the sequence of node labels at the start of every iteration of Algorithm 1 in phase k. Note that $\bar{y}^{(i)} \geq \bar{y}^{(i+1)}$ and $\bar{y}_{u'}^{(i)} > \bar{y}_{u'}^{(i+1)}$ for all $i < \ell$. Let $f : \mathbb{R} \to \bar{\mathbb{R}}$ be the parametric function associated with the linear fractional program (F_u) for the subsystem $(G^{(k)}, c, \gamma)$. We may assume that $\ell \geq 1$, which in turn implies that $f(y_{u'}^{(1)})$ is finite by Lemma 4.4. By Lemma 4.6, there are no negative unit-gain cycles or bicycles in $G^{(k)} \setminus \delta^+(u)$. It follows that all negative unit-gain cycles and negative bicycles in $G^{(k)}$ contain u. Hence, there exists a smallest $\varepsilon \geq 0$ such that the subsystem $(G^{(k)}, \hat{c}, \gamma)$ is feasible, where $\hat{c} \in \mathbb{R}^{m_k}$ are modified arc costs defined by $\hat{c}_e := c_e + \varepsilon$ if $e \in \delta^+(u)$ and $\hat{c}_e := c_e$ otherwise.

18

For each $i > 1$, every basic optimal primal solution to $f(\bar{y}_{u'}^{(i)})$ is a path flow from u to u' in $G_u^{(k)}$. This is because u cannot reach a flow-absorbing cycle in the subgraph of $G_u^{(k)}$ induced by tight arcs with respect to $\bar{y}_u^{(i)}$. Indeed, such a cycle would impose an upper bound of $\bar{y}_u^{(i)}$ on the variable y_u. As $\bar{y}_u^{(i-1)} > \bar{y}_u^{(i)}$, this contradicts the feasibility of $\bar{y}_V^{(i-1)}$ to the dual LP for $f(\bar{y}_{u'}^{(i-1)})$. For each $i > 1$, let $P^{(i)}$ be a u-u' path with the smallest gain factor in the subgraph of $G_u^{(k)}$ induced by tight arcs with respect to $\bar{y}^{(i)}$. Note that $P^{(i)}$ is well-defined due to the same reason as above. Then, $\gamma(P^{(i)}) - 1 = \min \partial f(\bar{y}_{u'}^{(i)})$. Denote this sequence of u-u' paths as $\mathcal{P} := (P^{(2)}, P^{(3)}, \ldots, P^{(\ell)})$.

Without loss of generality, we may assume that $\bar{y}^{(i)}$ is finite for all $i \geq 1$. Since every vertex can reach a flow-absorbing cycle in $G^{(k)}$, there exists a pointwise maximal solution $y^* \in \mathbb{R}^n$ to the modified system $(G^{(k)}, \hat{c}, \gamma)$. Define the reduced cost $c^* \in \mathbb{R}_+^{m_k}$ as $c_{vw}^* := \hat{c}_{vw} + \gamma_{vw} y_w^* - y_v^*$ for all $vw \in E(G^{(k)})$. Since $f(y_u^*) = -\varepsilon$, we obtain

$$
\begin{aligned}
c^*(P^{(i)}) &= c(P^{(i)}) - (1 - \gamma(P^{(i)})) y_u^* + \varepsilon \\
&= f(\bar{y}_{u'}^{(i)}) - (1 - \gamma(P^{(i)}))(y_u^* - \bar{y}_u^{(i)}) - f(y_u^*) \\
&= D_f(y_u^*, \bar{y}_{u'}^{(i)}) \leq \frac{1}{2} D_f(y_u^*, \bar{y}_{u'}^{(i-2)}) = \frac{1}{2} c^*(P^{(i-2)})
\end{aligned}
$$

for all $i > 3$, where the inequality is due to Lemma 3.4.

Consider the vector $x \in \mathbb{R}_+^{m_k}$ defined by

$$
x_{vw} := \begin{cases} \max_{i \in [\ell]} \left\{ \gamma(P_{uv}^{(i)}) : vw \in E(P^{(i)}) \right\} & \text{if } vw \in \cup_{i=1}^\ell E(P^{(i)}), \\ 0 & \text{otherwise.} \end{cases}
$$

By Lemma 4.10, the sequence $\mathcal{P}$ satisfies subpath monotonicity at u. Hence, x_{vw} is equal to the gain factor of the u-v subpath of the last path in $\mathcal{P}$ that contains vw. Let $0 \leq c_1^* x_1 \leq c_2^* x_2 \leq \cdots \leq c_{m_k}^* x_{m_k}$ be the elements of $c^* \circ x$ in nondecreasing order. Let $e_1, e_2, \ldots, e_{m_k}$ denote the arcs in $G^{(k)}$ according to this order, and define $d_i := \sum_{j=1}^i c_j^* x_j$ for every $i \in [m_k]$. Then, $c^*(P^{(i)}) \in [d_1, d_{m_k}]$ for all $i \in [\ell]$ because $c^*(P^{(\ell)}) \geq d_1$ and $c^*(P^{(1)}) \leq d_{m_k}$. To prove that $\ell = O(m_k)$, it suffices to show that every interval $(d_i, d_{i+1}]$ contains the cost of at most two paths from $\mathcal{P}$.

Pick $j < m_k$. Among all the paths in $\mathcal{P}$ whose costs lie in $(d_j, d_{j+1}]$, let $P^{(i)}$ be the most expensive one. If $d_j \geq d_{j+1}/2$, then

$$
c^*(P^{(i+2)}) \leq \frac{1}{2} c^*(P^{(i)}) \leq \frac{1}{2} d_{j+1} \leq d_j.
$$

On the other hand, if $d_j < d_{j+1}/2$, then

$$
c^*(P^{(i+2)}) \leq \frac{1}{2} c^*(P^{(i)}) \leq \frac{1}{2} d_{j+1} = d_{j+1} - \frac{1}{2} d_{j+1} = c_{j+1}^* x_{j+1} + d_j - \frac{1}{2} d_{j+1} < c_{j+1}^* x_{j+1}.
$$

By subpath monotonicity, the paths from $P^{(i+2)}$ onwards do not contain an arc from the set $\{e_{j+1}, e_{j+2}, \ldots, e_{m_k}\}$. Therefore, their costs are at most d_j each. $\qquad\square$

The runtime of every iteration of Algorithm 1 is dominated by GRAPEVINE. Thus, following the discussion in Appendix B.3, we obtain the following result.

Corollary 4.12. *Algorithm 3 solves the feasibility of M2VPI linear systems in $O(m^2 n^2)$ time.*

One might wonder if Algorithm 3 is still strongly polynomial if we replace the look-ahead Newton–Dinkelbach method on Line 14 with the standard version. In Appendix C, we show that this is indeed the case, though with a slower convergence.

4.3 Deterministic Markov Decision Processes

In this subsection, we replace GRAPEVINE with a variant of Dijkstra's algorithm (Algorithm 4) in order to speed up Algorithm 3 for solving a special class of 2VPI linear programs, known as deterministic Markov decision processes (DMDPs). This idea was briefly mentioned by Madani in [21]; we will supply the details. Recall that an instance of DMDP is described by a directed multigraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^m$ and *discount* factors $\gamma \in (0, 1]^m$. The goal is to select an outgoing arc from every node so as to minimize the total discounted cost over an infinite time horizon. It can be formulated as the following pair of primal and dual LPs.

$$\begin{array}{ll} \min \ c^\top x & \text{(P)} \\ \text{s.t.} \ \ \nabla x_v = 1 \quad \forall v \in V \\ \qquad x \geq 0 \end{array} \qquad\qquad \begin{array}{ll} \max \ \mathbb{1}^\top y & \text{(D)} \\ \text{s.t.} \ \ y_v - \gamma_e y_w \leq c_e \quad \forall e = (v, w) \in E \end{array}$$

Since the discount factor of every cycle is at most 1, there are no bicycles in G. Consequently, by Theorem 4.3, the linear program (D) is infeasible if and only if there is a negative unit-gain cycle in G. This condition can be easily checked by running a negative cycle detection algorithm on the subgraph induced by arcs with discount factor 1.

Algorithm 4 is slightly modified from the standard Dijkstra's algorithm [5] to handle our notion of shortest paths that depends on node labels. As part of the input, it requires a target node t with out-degree zero, node labels $y \in \mathbb{R}^n$ which induce nonnegative reduced costs, and a parameter $\alpha < y_t$. As output, it returns a shortest path tree T to t when y_t is decreased to α. It also returns node labels $z \in \mathbb{R}^n$ which certify the optimality of T, i.e. z induces nonnegative reduced costs with zero reduced costs on T, and $z_t = \alpha$.

An *iteration* of Algorithm 4 refers to a repetition of the while loop. In the pseudocode, observe that $\bar{c}_e \geq 0$ for all $e \in E \setminus \delta^-(u)$.

Lemma 4.13. *Algorithm 4 is correct.*

Proof. We proceed by induction on the number of elapsed iterations k. Let z be the node labels at the end of iteration k. For each $i \leq k$, let v_i be the node added to S in iteration i. Note that z_S remains unchanged in future iterations. We first show that $z_{v_2} \leq z_{v_3} \leq \cdots \leq z_{v_k} < z_{v_1} = 0$. The base case $k = 1$ is true due to our initialization, while the base case $k = 2$ is true because $v_2 \in R$. For the inductive step, suppose that the claim is true for some $k \geq 2$. Let $v_{k+1} = \arg\min_{v \in R} \{z_v\}$ and $v_j = \text{pred}(v_{k+1})$ for some $j \leq k$. We know that $z_{v_{k+1}} < 0$ because $v_{k+1} \in R$. If $j < k$, then $z_{v_{k+1}} \geq z_{v_k}$, as otherwise v_k would not have been chosen to enter S in iteration k. If $j = k$, using the fact that $\gamma_{v_{k+1}v_k} \leq 1$ and $\bar{c}_{v_{k+1}v_k} \geq 0$, we obtain

$$z_{v_{k+1}} = \bar{c}_{v_{k+1}v_k} + \gamma_{v_{k+1}v_k} z_{v_k} \geq z_{v_k}.$$

It is left to show that $\bar{c}_{vw} + \gamma_{vw} z_w - z_v \geq 0$ for all $vw \in E(G[S])$. The base case $k = 1$ is trivially true. For the inductive step, suppose that the statement is true for some $k \geq 1$. We know that $z_{v_{k+1}} \leq \bar{c}_{v_{k+1}v} + \gamma_{v_{k+1}v} z_v$ for every outgoing arc $v_{k+1}v \in E(G[S])$. For every incoming arc $vv_{k+1} \in E(G[S])$, using the fact that $\gamma_{vv_{k+1}} \leq 1$ and $\bar{c}_{vv_{k+1}} \geq 0$, we get

$$z_v \leq \bar{c}_{vv_{k+1}} + \gamma_{vv_{k+1}} z_v \leq \bar{c}_{vv_{k+1}} + \gamma_{vv_{k+1}} z_{v_{k+1}},$$

where the second inequality follows from $z_v \leq z_{v_{k+1}}$. $\qquad\square$

Algorithm 4: Recompute shortest paths to t

> **input** : A directed multigraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^E$ and discount factors
> $\gamma \in (0, 1]^E$, a target node $t \in V$ where $\delta^+(t) = \emptyset$, node labels $y \in \mathbb{R}^V$ such that
> $c_{vw} + \gamma_{vw} y_w - y_v \geq 0$ for every $vw \in E$, and a parameter $\alpha < y_t$
> **output**: An in-tree T rooted at t and node labels $z \in \mathbb{R}^V$ such that $z \leq y$, $z_u = \alpha$ and
> $c_{vw} + \gamma_{vw} z_w - z_v \geq 0$ for every $vw \in E$, with equality on every arc of T.

1 $y_u \leftarrow \alpha$
2 Define reduced cost $\bar{c} \in \mathbb{R}^E$ by $\bar{c}_{vw} \leftarrow c_{vw} + \gamma_{vw} y_w - y_v$ for all $vw \in E$
3 Initialize node labels $z \in \mathbb{R}^V$ by $z_v \leftarrow 0$ for all $v \in V$
4 Initialize sets $R \leftarrow \{t\}$ and $S \leftarrow \emptyset$
5 **while** $R \neq \emptyset$ **do**
6 $w \leftarrow \arg\min_{v \in R} \{z_v\}$
7 $R \leftarrow R \setminus \{w\}$
8 $S \leftarrow S \cup \{w\}$
9   **foreach** $vw \in E$ where $v \notin S$ **do**
10   **if** $z_v > \bar{c}_{vw} + \gamma_{vw} z_w$ **then**
11 $z_v \leftarrow \bar{c}_{vw} + \gamma_{vw} z_w$
12 $\text{pred}(v) \leftarrow vw$
13 $R \leftarrow R \cup \{v\}$

14 Let T be the in-tree defined by $\text{pred}()$
15 $z \leftarrow y + z$
16 **return** (z, T)

In every phase k of Algorithm 3, Algorithm 4 now replaces GRAPEVINE as the new value and supergradient oracle of f. Given an optimal dual solution y to $f(\alpha)$ for some $\alpha \in \mathbb{R}$, Algorithm 4 is used to compute an optimal dual solution to $f(\alpha')$ for any $\alpha' < \alpha$. In particular, we run it on the modified graph $G_u^{(k)}$ with input node labels $\bar{y}$ defined by $\bar{y}_{u'} := \alpha$ and $\bar{y}_v := y_v$ for all $v \neq u'$, target node $t = u'$, and parameter $\alpha' < \alpha$. Note that u' has out-degree zero in $G_u^{(k)}$ by construction. Let $(\bar{z}, T)$ be the node labels and tree returned by Algorithm 4, where $\bar{z}_V$ is an optimal dual solution to $f(\alpha')$. A supergradient at $f(\alpha')$ can be inferred from the output via complementary slackness. Specifically, if $u \in V(T)$, then $\gamma(P) - 1 \in \partial f(\alpha')$ where P is the unique u-u' path in T. Otherwise, u can reach a flow-absorbing cycle in the tight subgraph with respect to $\bar{z}$, so $-1 \in \partial f(\alpha')$.

An efficient implementation of Dijkstra's algorithm using Fibonacci heaps was given by Fredman and Tarjan [9]. It can also be applied to our setting, with the same running time of $O(m + n \log n)$. Consequently, we obtain a faster running time of Algorithm 3 for DMDPs.

Corollary 4.14. *Algorithm 3 solves deterministic MDPs in $O(mn(m + n \log n))$ time.*

5 Parametric Submodular Function Minimization

Let V be a set with n elements and define $2^V := \{S : S \subseteq V\}$ to be the set of all subsets of V. A function $h : 2^V \rightarrow \mathbb{R}$ is submodular if

$$h(S) + h(T) \geq h(S \cap T) + h(S \cup T) \quad \forall S, T \subseteq V.$$

Given a non-negative submodular function $h : 2^V \to \mathbb{R}_+$ and a vector $a \in \mathbb{R}^V$ satisfying $\max_{i \in V} a_i > 0$, we examine the problem of computing

$$\delta^* := \max\{\delta : \min_{S \subseteq V} h(S) - \delta a(S) \geq 0\}, \tag{3}$$

where $a(S) := \sum_{i \in S} a_i$. As the input model, we assume access to an evaluation oracle for h, which allows us to query $h(S)$ for any set $S \subseteq V$. The above problem models the line-search problem inside a submodular polyhedron and has been studied in [10, 24, 33].

To connect to the root finding problem studied in previous sections, for $\delta \in \mathbb{R}$, we define

$$f(\delta) := \min_{S \subseteq V} h_\delta(S) := \min_{S \subseteq V} h(S) - \delta a(S).$$

Since f is the minimum of 2^n affine functions, f is a piecewise linear concave function. Noting that f is continuous, problem (3) can be equivalently restated as that of computing the largest root of f, i.e., the largest $\delta^* \in \mathbb{R}$ such that $f(\delta^*) = 0$. The assumption that h is non-negative ensures that $f(0) \geq 0$, and the assumption that $\max_{i \in V} a_i > 0$ ensures that δ^* exists and $\delta^* \geq 0$ (see the initialization section below). Given the root finding representation, we may apply the Newton–Dinkelbach method on f to compute δ^*. This approach was taken by Goemans, Gupta and Jaillet [10], who were motivated to give a more efficient alternative to the parametric search based algorithm of Nagano [24]. Their main result is as follows:

Theorem 5.1 ([10]). *The Newton-Dinkelbach method requires at most $n^2 + O(n \log^2 n)$ iterations to solve* (3).

The goal of this section is to give a simplified potential function based proof of the above theorem using the *accelerated* Newton–Dinkelbach method (Algorithm 1), where we will give a slightly weaker $2n^2 + 2n + 4$ bound on the iteration count. Our analysis uses the same combinatorial ring family analysis as in [10], but the Bregman divergence enables considerable simplifications.

5.1 Implementing the Accelerated Newton–Dinkelbach

We explain how to implement and initialize the accelerated Newton–Dinkelbach method in the present context. To begin, Algorithm 1 requires access to the supergradients of f. For $\delta \in \mathbb{R}$, it is easy to verify that

$$S \in \operatorname{argmin}\{h_\delta(T) : T \subseteq V\} \Rightarrow -a(S) \in \partial f(\delta).$$

Therefore, computing supergradients of f can be reduced to computing minimizers of the submodular functions $h_\delta(S) := h(S) - \delta a(S)$, $\delta \in \mathbb{R}$. Submodular function minimization (SFM) is a classic problem in combinatorial optimization and has been extensively studied from the viewpoint of strongly polynomial algorithms [4, 15, 16, 19, 17]. The fastest strongly polynomial running time is due to Jiang [17] who gave an algorithm for SFM using $O(n^3)$ calls to the evaluation oracle.

In what follows, we assume access to an SFM oracle, that we will call on the submodular functions h_δ, for $\delta \in \mathbb{R}$. Each iteration of Algorithm 1 requires two calls to a supergradient oracle, one for the standard step and one for the look-ahead step, and hence can be implemented using two calls to the SFM oracle. Gupta, Goemans and Jaillet [10] were directly concerned with the number of calls to an SFM oracle, which is exactly equal to the number of iterations of standard Newton–Dinkelbach (it requires only one SFM call per iteration instead of two). As mentioned above, we will prove a $2n^2 + 2n + 4$ bound on the iteration count for accelerated Newton–Dinkelbach, which will recover the bound on the number of SFM calls of [10] up to a factor 4. Since accelerated

Newton–Dinkelbach is always as fast as the standard method (it goes at least as far in each iteration), the iteration bound in Theorem 5.1 in fact applies to the accelerated method as well.

We now explain how to initialize the method. For this purpose, Algorithm 1 requires $\delta^{(1)} \in \mathbb{R}$ and $g^{(1)} \in \partial f(\delta^{(1)})$ such that $f(\delta^{(1)}) \leq 0$ and $g^{(1)} < 0$. We proceed as in [10] and let $\delta^{(1)} := \min\{h(\{i\})/a_i : i \in V, a_i > 0\} \geq 0$, which is well-defined by assumption on a. We compute $f(\delta^{(1)})$ by the SFM oracle. Note that

$$f(\delta^{(1)}) = \min_{S \subseteq V} h_\delta(S) \leq \min_{i \in V, a_i > 0} h(\{i\}) - \delta^{(1)} a_i = 0.$$

If $f(\delta^{(1)}) = 0$, we return $\delta^{(1)}$, as we are already done. Otherwise if $f(\delta^{(1)}) < 0$, set $g^{(1)} = -a(S^{(1)})$, where $S^{(1)} \in \operatorname{argmin}_{S \subseteq V} h_{\delta^{(1)}}(S)$ as returned by the oracle. From here, note that

$$0 > f(\delta^{(1)}) = h_{\delta^{(1)}}(S^{(1)}) = h(S^{(1)}) - \delta^{(1)} a(S^{(1)}) = h(S^{(1)}) + g^{(1)} \delta^{(1)} \geq g^{(1)} \delta^{(1)},$$

where the last inequality follows by non-negativity of h. Since $\delta^{(1)} \geq 0$, the above implies that $\delta^{(1)} > 0$ and $g^{(1)} < 0$. We may therefore initialize Algorithm 1 with $\delta^{(1)}$ and $g^{(1)}$.

Assuming $f(\delta^{(1)}) < 0$, the largest root δ^* of f is guaranteed to exist in the interval $[0, \delta^{(1)})$. This follows since f is continuous, $f(0) = \min_{S \subseteq V} h(S) \geq 0$ (by non-negativity of h) and $f(\delta^{(1)}) < 0$. Note that there does not exist a root larger than $\delta^{(1)}$ due to the concavity of f. So, Algorithm 1 on input $f, \delta^{(1)}, g^{(1)}$ is guaranteed to output the desired largest root δ^* in a finite number of iterations (recalling that f is piecewise affine with 2^n pieces). In the next subsection, we prove a $2n^2 + 2n + 4$ bound on the number of iterations.

5.2 Proof of the $2n^2 + 2n + 4$ Iteration Bound

Let $\delta^{(1)} > \cdots > \delta^{(\ell)} = \delta^*$ denote the iterates of Algorithm 1 on input f and $\delta^{(1)} > 0, g^{(1)} < 0$ as above. For each $i \in [\ell]$, let $S^{(i)}$ be any set satisfying

$$S^{(i)} \in \operatorname{argmax}\{a(S) : S \in \operatorname{argmin}_{T \subseteq V} h_{\delta^{(i)}}(T)\}.$$

It is not hard to verify that $S^{(i)}$, $i \in [\ell]$, is a minimizer of $h_{\delta^{(i)}}$ inducing the right derivative of f at $\delta^{(i)}$. Precisely, $-a(S^{(i)}) = \inf_{g \in \partial f(\delta^{(i)})} g$, $\forall i \in [\ell]$. We note that the sets $S^{(i)}$, $i \in [\ell]$, need not be the sets outputted by the SFM oracle, and are only required for the analysis of the algorithm.

Our goal is to prove that $\ell \leq 2n^2 + 2n + 4$. For this purpose, we rely on the key idea of [10], which is to extract an increasing sequence of *ring-families* from the sets $S^{(i)}$, $i \in [\ell]$.

A *ring family* $\mathcal{R} \subseteq 2^V$ is a subsystem of sets that is closed under unions and intersections, precisely $A, B \in \mathcal{R} \Rightarrow A \cap B, A \cup B \in \mathcal{R}$. Given $\mathcal{T} \subseteq 2^V$, we let $\mathcal{R}(\mathcal{T})$ denote the smallest ring-family containing $\mathcal{T}$. We will use the following lemma of [10] which bounds the length of an increasing sequence of ring-families:

Lemma 5.2 ([10, Theorem 2]). *Let $\emptyset \neq \mathcal{R}_1 \subsetneq \mathcal{R}_2 \subsetneq \cdots \subsetneq \mathcal{R}_k \subseteq 2^V$, where $|V| = n$. Then $k \leq \binom{n+1}{2} + 1$.*

The proof of the above lemma is based on the Birkhoff representation of a ring family. Precisely, for any ring-family $\mathcal{R} \subseteq 2^V$, with $\emptyset, V \in \mathcal{R}$, there exists a directed graph G on V, such that the sets $S \in \mathcal{R}$ are exactly the subsets of vertices of G having no out-neighbors. The main idea for the bound is that the digraph representation of $\mathcal{R}_i, i \in [k]$, must lose edges as i increases. The next statement is a slightly adapted version of [10, Theorem 5] that is sufficient for our purposes. It shows that a sequence of sets with geometrically increasing h values forms an increasing sequence of ring families. We include a proof for completeness.

Lemma 5.3. *Let $h : 2^V \to \mathbb{R}_+$ be a non-negative submodular function. Consider a sequence of distinct sets $T_1, T_2, \ldots, T_q \subseteq V$ such that $h(T_{i+1}) > 4h(T_i)$ for $i \in [q-1]$. Then $T_{i+1} \notin \mathcal{R}(\{T_1, \ldots, T_i\})$ for all $i \in [q-1]$.*

Proof. Let $\mathcal{R}_i := \mathcal{R}(\{T_1, \ldots, T_i\})$, $\forall i \in [q]$. We claim that $\max_{S \in \mathcal{R}_i} h(S) \leq 2h(T_i)$, $\forall i \in [q]$. This proves $T_{i+1} \notin \mathcal{R}_i$, for $i \in [q-1]$, since $h(T_{i+1}) > 4h(T_i) \geq 2h(T_i) \geq \max_{S \in \mathcal{R}_i} h(S)$, noting that the second inequality uses that h is non-negative.

We now prove the claim by induction on $i \in [q]$. The base case $i = 1$ is trivial since $\mathcal{R}_1 = \{T_1\}$. We now assume that $\max_{S \in \mathcal{R}_i} h(S) \leq 2h(T_i)$, for $1 \leq i \leq q-1$, and prove the corresponding bound for $\mathcal{R}_{i+1}$. Recalling that $\mathcal{R}_{i+1}$ is the ring-family generated by $\mathcal{R}_i$ and T_{i+1}, it is easy to verify that the set system

$$\mathcal{R}_i \cup \{T_{i+1}\} \cup \{S \cup T_{i+1} : S \in \mathcal{R}_i\} \cup \{S \cap T_{i+1} : S \in \mathcal{R}_i\} \cup \{S_1 \cup (S_2 \cap T_{i+1}) : S_1, S_2 \in \mathcal{R}_i\}$$

is a ring-family and hence is equal to $\mathcal{R}_{i+1}$. It therefore suffices to upper bound $h(X)$ for a set X of the above type. For $X \in \mathcal{R}_i$ or $X = T_{i+1}$, the bound is by assumption. For $X = S_1 \cup (S_2 \cap T_{i+1})$, $S_1, S_2 \in R_{i+1}$, we prove the bound as follows:

$$\begin{aligned}
h(S_1 \cup (S_2 \cap T_{i+1})) &\leq h(S_1) + h(S_2 \cap T_{i+1}) - h(S_1 \cap S_2 \cap T_{i+1}) \quad (\text{ by submodularity of } h) \\
&\leq h(S_1) + h(S_2) + h(T_{i+1}) - h(S_2 \cup T_{i+1}) - h(S_1 \cap S_2 \cap T_{i+1}) \\
&\leq h(S_1) + h(S_2) + h(T_{i+1}) \quad (\text{ by non-negativity of } h) \\
&\leq 4h(T_i) + h(T_{i+1}) \quad (\text{ by the induction hypothesis }) \\
&\leq 2h(T_{i+1}). \quad (\text{ since } 4h(T_i) < h(T_{i+1}))
\end{aligned}$$

For $X = S \cup T_{i+1}$ or $X = S \cap T_{i+1}$, $S \in \mathcal{R}_i$, similarly to the above, one has

$$h(X) \leq h(S) + h(T_{i+1}) \leq 2h(T_i) + h(T_{i+1}) \leq \frac{3}{2} h(T_{i+1}), \text{ as needed.}$$

$\square$

We now use the Bregman-divergence analysis to show that for the function h_{δ^*}, the sequence of sets $T_i = S^{(\ell - 4(i-1))}$, $1 \leq i \leq \lfloor \frac{\ell+3}{4} \rfloor$ satisifes the conditions of this lemma. Combined with Lemma 5.2, we get that the number of iterations satisfies

$$\lfloor (\ell + 3)/4 \rfloor \leq \binom{n+1}{2} + 1 \Rightarrow \ell \leq 2n^2 + 2n + 4, \text{ as needed.}$$

Lemma 5.4. *Let us define*

$$T_i := S^{(\ell - 4(i-1))}, \quad i \in [q] \quad \text{for } q := \left\lfloor \frac{\ell+3}{4} \right\rfloor.$$

Then, the function h_{δ^*} and the sequence of sets $T_1, T_2, \ldots, T_q$ satisfy the conditions in Lemma 5.3.

Proof. The function h_{δ^*} is clearly submodular, and its minimum is 0 since $0 = f(\delta^*) = \min_{S \subseteq V} h_{\delta^*}(S) = h_{\delta^*}(S^{(\ell)}) = h_{\delta^*}(T_1)$. In particular, h_{δ^*} is non-negative. It is left to show $h_{\delta^*}(T_{i+1}) > 4h_{\delta^*}(T_i)$ for $i \in [q-1]$. For each $\delta^{(i)}$, $i \in [\ell]$, we see that

$$\begin{aligned}
D_f(\delta^*, \delta^{(i)}) &= f(\delta^{(i)}) + \sup_{g \in \partial f(\delta^{(i)})} g(\delta^* - \delta^{(i)}) - f(\delta^*) \\
&= h_{\delta^{(i)}}(S^{(i)}) - a(S^{(i)})(\delta^* - \delta^{(i)}) \quad \left(\text{ by our choice of } S^{(i)} \text{ and } f(\delta^*) = 0 \right) \\
&= h(S^{(i)}) - \delta^{(i)} a(S^{(i)}) - a(S^{(i)})(\delta^* - \delta^{(i)}) = h_{\delta^*}(S^{(i)}).
\end{aligned}$$

24

By Lemma 3.4 and the above, we get for $3 \leq i \leq \ell$ that

$$D_f(\delta^*, \delta^{(i)}) < \frac{1}{2}D_f(\delta^*, \delta^{(i-2)}) \Leftrightarrow h_{\delta^*}(S^{(i)}) < \frac{1}{2}h_{\delta^*}(S^{(i-2)}). \tag{4}$$

Then, $h_{\delta^*}(T_{i+1}) > 4h_{\delta^*}(T_i)$ for $i \in [q-1]$ follows by the definition of the T_i sets. $\qquad\square$

6 Conclusion

We have presented an accelerated version of the Newton–Dinkelbach method for univariate concave functions, and illustrated its utility on three application domains. For linear fractional combinatorial optimization, we obtain an improved $O(m \log m)$ iteration bound. For 2VPI LP feasibility, we get a strongly polynomial label-correcting algorithm which runs in $O(mn)$ iterations. Finally, the method yields a simplified analysis of the parametric submodular function minimization result by Goemans et al. [10].

The key idea is to analyze the Newton–Dinkelbach method using the Bregman divergence; previous work [28, 35] analyzed the gradient and function value of the iterates. With the look-ahead step, we show that the Bregman divergence halves every two iterations. The Bregman divergence has a useful interpretation in terms of a 'modified' cost function, which allows us to derive convergence bounds for various problems. We expect that this accelerated method and its analysis find more applications in other fractional/parametric optimization problems.

For 2VPI LP feasibility, every iteration of the Newton–Dinkelbach method takes $O(mn)$ time, which results in a total running time of $O(m^2n^2)$ for the overall label-correcting algorithm. We do not know whether our analysis is tight; it may be possible to amortize over the $O(mn)$ iterations. It is interesting to see whether the running time can be lowered to match the $O(mn^2 \log m)$ bound of Hochbaum and Naor [14].

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, 1993. 4

[2] B. Aspvall and Y. 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, 14, 30

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

[4] D. Dadush, L. A. Végh, and G. Zambelli. Geometric rescaling algorithms for submodular function minimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 832–848, USA, 2018. Society for Industrial and Applied Mathematics. 22

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 20

[6] W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967. 2

[7] H. Edelsbrunner, G. Rote, and E. Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theor. Comput. Sci.*, 66(2):157–180, 1989. 3, 29

[8] E. A. Feinberg and J. Huang. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. *Oper. Res. Lett.*, 42(2):130–131, 2014. 5

[9] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. 21

[10] M. X. Goemans, S. Gupta, and P. 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. 3, 22, 23, 25

[11] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989. 4

[12] T. D. Hansen, H. Kaplan, and U. 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. 5

[13] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Math. Program.*, 62:69–83, 1993. 3, 29

[14] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, 1994. 3, 4, 25

[15] S. Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, 2008. 22

[16] S. Iwata and J. B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1230–1237. SIAM, 2009. 22

[17] H. Jiang. Minimizing convex functions with integral minimizers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 976–985. SIAM, 2021. 22

[18] A. Karczmarz. Improved strongly polynomial algorithms for deterministic MDPs, 2VPI feasibility, and discounted all-pairs shortest paths. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 154–172, 2022. 3

[19] Y. T. Lee, A. Sidford, and S. C.-w. Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065. IEEE, 2015. 22

[20] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 394–402, 1995. 5

[21] O. 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. 4, 5, 20

[22] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979. 2, 3

[23] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12(2):347–353, 1983. 3

[24] K. Nagano. A strongly polynomial algorithm for line search in submodular polyhedra. *Discrete Optimization*, 4(3-4):349–359, 2007. 22

[25] N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *J. ACM*, 67(2):10:1–10:26, 2020. 4

[26] I. Post and Y. Ye. The simplex method is strongly polynomial for deterministic Markov decision processes. *Math. Oper. Res.*, 40(4):859–868, 2015. 5

[27] T. Radzik. Newton's method for fractional combinatorial optimization. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992. 2, 3, 4, 10

[28] T. Radzik. Fractional combinatorial optimization. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization: Volume 1–3*, pages 429–478. Springer US, 1998. 2, 3, 4, 6, 10, 25

[29] T. Radzik and A. V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, 1994. 4

[30] R. E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981. 3, 5, 12

[31] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998. 3

[32] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985. 4

[33] D. M. Topkis. Minimizing a submodular function on a lattice. *Operations research*, 26(2):305–321, 1978. 22

[34] L. A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.*, 42(1):179–211, 2017. 4

[35] Q. Wang, X. Yang, and J. 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. 3, 10, 25

A Proofs for Section 3 (An Accelerated Newton–Dinkelbach Method)

Lemma 3.1. 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,

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

Proof. Since $f(\delta^{(i)}) \leq 0$ and $g^{(i)} < 0$, by concavity of f we have that $f(\delta) \leq f(\delta^{(i)}) + g^{(i)}(\delta - \delta^{(i)}) < f(\delta^{(i)}) \leq 0$, for all $\delta > \delta^{(i)}$. Given this, we must have $\delta^* \leq \delta^{(i)}$ since either $f(\delta^*) = 0 \geq f(\delta^{(i)})$ or $0 > f(\delta^*) = \max_{z \in \mathbb{R}} f(z) \geq f(\delta^{(i)})$. As $\delta^{(i)} = \delta^{(i-1)} - \frac{f(\delta^{(i-1)})}{g^{(i-1)}} < \delta^{(i-1)}$, since $f(\delta^{(i-1)}), g^{(i-1)} < 0$, we have $f(\delta^{(i-1)}) < f(\delta^{(i)})$. Furthermore, $g^{(i)} \geq g^{(i-1)}$ is immediate from the concavity of f.

To understand when $g^{(i)} = g^{(i-1)}$, we see by concavity that

$$g^{(i)} \geq \inf_{g \in \partial f(\delta^{(i)})} g \geq \frac{f(\delta^{(i-1)}) - f(\delta^{(i)})}{\delta^{(i-1)} - \delta^{(i)}} \geq \sup_{g \in \partial f(\delta^{(i-1)})} g \geq g^{(i-1)}.$$

To have equality throughout, we must therefore have that $g^{(i)}$ and $g^{(i-1)}$ are equal to the respective infimum and supremum. We must also have $f(\delta^{(i)}) = 0$ since

$$\frac{f(\delta^{(i-1)}) - f(\delta^{(i)})}{\delta^{(i-1)} - \delta^{(i)}} = \frac{f(\delta^{(i-1)}) - f(\delta^{(i)})}{\frac{f(\delta^{(i-1)})}{g^{(i-1)}}} = g^{(i-1)} \left(1 - \frac{f(\delta^{(i)})}{f(\delta^{(i-1)})} \right)$$

To have equality throughout, we must therefore have that $g^{(i)}$ and $g^{(i-1)}$ are equal to the respective infimum and supremum and that $f(\delta^{(i)}) = 0$.

Lastly, since f is concave

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

The moreover now follows by dividing both sides by $f(\delta^{(i-1)}) < 0$. $\qquad \square$

Lemma 3.3. 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 $f(\delta^{(i)}) = 0$.

Proof. By Lemma 3.1, we know that $\delta^* \leq \delta^{(i)} < \delta^{(i-1)}$ and $0 \geq f(\delta^{(i)}) > f(\delta^{(i-1)})$. Hence,

$$\begin{aligned}
D_f(\delta^*, \delta^{(i-1)}) &= f(\delta^{(i-1)}) + \sup_{g \in \partial f(\delta^{(i-1)})} g(\delta^* - \delta^{(i-1)}) - f(\delta^*) \\
&\geq f(\delta^{(i-1)}) + g^{(i-1)}(\delta^{(i)} - \delta^{(i-1)}) + g^{(i-1)}(\delta^* - \delta^{(i)}) - f(\delta^*) \\
&\geq f(\delta^{(i)}) + g^{(i-1)}(\delta^* - \delta^{(i)}) - f(\delta^*) \quad (\text{ by concavity of } f) \\
&\geq f(\delta^{(i)}) + \sup_{g \in \partial f(\delta^{(i)})} g(\delta^* - \delta^{(i)}) - f(\delta^*) \\
&= D_f(\delta^*, \delta^{(i)}).
\end{aligned}$$

For the equality condition, note that the first inequality holds at equality if and only if $g^{(i-1)} = \inf_{g \in \partial f(\delta^{(i-1)})} g$. The second inequality holds at equality if and only if $f(\delta^{(i)}) = 0$ because $f(\delta^{(i-1)}) + g^{(i-1)}(\delta^{(i)} - \delta^{(i-1)}) = 0$ from the definition of $\delta^{(i)}$. If $f(\delta^{(i)}) = 0$, then $\delta^{(i)} = \delta^*$, and hence the third inequality holds at equality as well. $\qquad \square$

B Further Explanations

B.1 Reducing 2VPI to M2VPI

Following [7, 13], the idea is to replace each variable y_u with $(y_u^+ - y_u^-)/2$, where y_u^+ and y_u^- are newly introduced variables. Then, an inequality $ay_u + by_v \leq c$ becomes

$$a\left(\frac{y_u^+ - y_u^-}{2}\right) + b\left(\frac{y_v^+ - y_v^-}{2}\right) \leq c,$$

which contains four variable, but will be adjusted based on the signs of a and b: If a or b is zero, then the resulting inequality is already monotone and contains two variables. Next, if $\mathrm{sgn}(a) = \mathrm{sgn}(b)$, then we replace the inequality with $ay_u^+ - by_v^- \leq c$ and $-ay_u^- + by_v^+ \leq c$. Otherwise, we replace it with $ay_u^+ + by_v^- \leq c$ and $-ay_u^- - by_v^- \leq c$. Observe that every inequality in the new system is monotone and supported on exactly two variables. If $\hat{y}$ is a feasible solution to the original system, then setting $y^+ = \hat{y}$ and $y^- = -\hat{y}$ yields a feasible solution to the new system. Conversely, if $(\hat{y}^+, \hat{y}^-)$ is a feasible solution to the new system, then setting $y = (\hat{y}^+ - \hat{y}^-)/2$ yields a feasible solution to the original system. It follows that the two systems are equivalent.

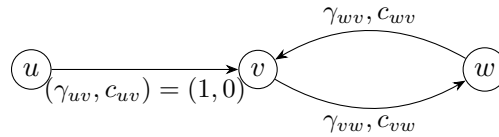## B.2 Non-existence of shortest paths

Figure 5: A shortest path from u with respect to node labels y may not exist.

Consider Figure 5. We will sketch three different scenarios in which a shortest path from u with respect to node labels $y \in \mathbb{R}^3$ does not exist. Throughout, let C be the unique directed cycle and C^k be the v-v walk that traverses C exactly $k \in \mathbb{N}$ times.

Negative unit gain cycle Let $\gamma_{wv} = \gamma_{vw} = 1$ and $c_{wv} = c_{vw} = -1$. Then the cycle C fulfils $\gamma(C) = 1$ and $c(C) = -2 < 0$. The concatenation of (u, v) and C^k leads to arbitrarily short walks from u. In particular, there exists no shortest path from u. This observation is independent of the node labels y. Recall as well, that the existence of such a cycle renders the M2VPI instance infeasible (Theorem 4.3).

Flow-absorbing cycle for large node labels Let $\gamma_{vw} = 1$ and $\gamma_{wv} = 1/2$. Then $\gamma(C) = \gamma_{vw}\gamma_{wv} = 1/2$, so C is flow-absorbing. Let further $c_{wv} = c_{vw} = 0$ and $y_w = y_v = 1$. Label-correcting for the cycle C then updates y_v and y_w in two strictly decreasing sequences, which both converge towards 0. Again, the concatenation of (u, v) and C^k leads to a sequence of u-v walks that have no smallest element.

Flow-generating cycle for small node labels Let $\gamma_{vw} = 1$ and $\gamma_{wv} = 2$. Then $\gamma(C) = \gamma_{vw}\gamma_{wv} = 2$, so C is flow-generating. Let further $c_{wv} = -1, c_{vw} = 0$ and $y_w = y_v = 0$. Label-correcting for the cycle C then updates y_v and y_w in two strictly decreasing and unbounded sequences. Again, the concatenation of (u, v) and C^k leads to a sequence of u-v walks that have no smallest element.

B.3 From $y^{\max}$ to a finite feasible solution

In this section, we show how to convert the node labels $y \in \bar{\mathbb{R}}^n$ obtained from Algorithm 3 into a finite feasible solution or an infeasibility certificate of the M2VPI system (G, c, γ) in question. We summarize the classical arguments already used by Aspvall and Shiloach [2]. If y is finite, then we are done because there are no violated arcs in G with respect to y. In fact, y is the pointwise maximal solution by Theorem 4.8. So, we may assume that $y_u = \infty$ for some $u \in V$.

Define $y^{\min} \in \bar{\mathbb{R}}^n$ as the pointwise minimal solution to (G, c, γ) if the system is feasible, where $y_v^{\min} := -\infty$ if and only if the variable y_v is unbounded from below. Consider the reversed graph $\overleftarrow{G} = (V, \overleftarrow{E})$, where $\overleftarrow{E} := \{vu : uv \in E\}$ denotes the set of reversed arcs. The cost and gain factor of each arc $vu \in \overleftarrow{E}$ are given by $\overleftarrow{c}_{vu} := c_{uv}/\gamma_{uv}$ and $\overleftarrow{\gamma}_{vu} := 1/\gamma_{uv}$ respectively. The M2VPI system defined by $(\overleftarrow{G}, \overleftarrow{c}, \overleftarrow{\gamma})$ is equivalent to the original system (G, c, γ), which can be verified by performing the change of variables $z = -y$. Let us run Algorithm 3 on $(\overleftarrow{G}, \overleftarrow{c}, \overleftarrow{\gamma})$. By Theorem 4.8, if it returns node labels $z \in \bar{\mathbb{R}}^n$, then $z = -y^{\min}$ if the system is feasible. Otherwise, the system is infeasible. If z is finite, then we are again done because there are no violated arcs in $\overleftarrow{G}$ with respect to z. So, we may assume that $z_v = \infty$ for some $v \in V$.

If $y_w = z_w = \infty$ for some $w \in V$, then we know that w cannot reach a flow-absorbing cycle in G and $\overleftarrow{G}$. The inability to reach a flow-absorbing cycle in $\overleftarrow{G}$ is equivalent to the inability to be reached by a flow-generating cycle in G. Denote $W := \{w \in V : y_w = z_w = \infty\}$. Observe that every node $w \in W$ is not strongly connected to any $v \notin W$ in G. Thus, checking the feasibility of the system amounts to checking whether there exists a negative unit-gain cycle in $G[W]$. This can be done by running GRAPEVINE on $G[W]$. Let $C_1, C_2, \ldots, C_k$ be the sink components in the strongly connected component decomposition of $G[W]$, and pick any $v_i \in V(C_i)$ for all $i \in [k]$. Then, the input node labels $y' \in \bar{\mathbb{R}}^W$ to GRAPEVINE are set as $y'_{v_i} \in \mathbb{R}$ for all $i \in [k]$ and $y'_v := \infty$ for all other nodes. Let $z' \in \mathbb{R}^W$ be the returned node labels. It is easy to see that there exists a negative unit-gain cycle in $G[W]$ if and only if there exists a violated arc in $G[W]$ with respect to z'.

If the check above reveals that the system is feasible, then we have $y = y^{\max}$ and $-z = y^{\min}$ by Theorem 4.8. Then, we can apply a result of Aspvall and Shiloach which states that the interval $[y_u^{\min}, y_u^{\max}]$ is the projection of the feasible region onto the coordinate y_u for every $u \in V$. To obtain a feasible solution, we simply fix a coordinate $y_u \in [y_u^{\min}, y_u^{\max}]$, update $y^{\min}$ and $y^{\max}$ using a generic label-correcting algorithm like GRAPEVINE, and repeat.

C 2VPI Analysis without Acceleration

In this section, we analyze the convergence of Algorithm 3 when the look-ahead Newton–Dinkelbach method is replaced with the standard version. Interestingly, we also obtain a strongly polynomial runtime in this case, albeit slower than the accelerated version by a factor of $O(\log n)$. To achieve the desired runtime, we slightly strengthen Lemma 3.6, whose proof remains largely the same.

Lemma C.1. *Let $c \in \mathbb{R}_+^m$ and $x^{(1)}, x^{(2)}, \ldots, x^{(k)} \in \mathbb{Z}^m$ such that $\left\|x^{(i)}\right\|_1 \leq n$ for all $i \in [k]$. If*

$$0 < c^\top x^{(i+1)} \leq \frac{1}{2} c^\top x^{(i)}$$

for all $i < k$, then $k = O(m \log n)$.

Proof of Lemma C.1. Consider the polyhedron $P \subseteq \mathbb{R}^m$ defined by the following constraints:

$$(x^{(i)} - 2x^{(i+1)})^\top z \geq 0 \qquad \forall i < k$$
$$(x^{(k)})^\top z = 1$$
$$z \geq 0.$$

Let $A \in \mathbb{R}^{(k+m)\times m}$ and $b \in \mathbb{R}^{k+m}$ denote the coefficient matrix and right-hand side vector of this system. The polyhedron P is nonempty because it contains the vector $c/(x^{(k)})^\top c$. Moreover, since P does not contain a line, it has an extreme point. So there exists a vector $c' \in P$ such that $A'c' = b'$ for some nonsingular submatrix $A' \in \mathbb{R}^{m\times m}$ of the matrix A and a subvector $b' \in \mathbb{R}^m$ of the vector b. Cramer's rule says that for each $i \in [m]$,

$$c_i' = \frac{\det A_i'}{\det A'}$$

where the matrix A_i' is obtained from matrix A' by replacing the i-th column with vector b'. The 1-norm of the rows of A_i' is bounded by $3n$ and so by Hadamard's inequality $|\det(A_i')| \leq (3n)^m$.

As the matrix A' is nonsingular, we also have $|\det A'| \geq 1$, which implies that $c_i' \leq (3n)^m$ for all $i \in [m]$. Finally, using the constraints which define the polyhedron P, we obtain

$$1 = (x^{(k)})^\top c' \leq \frac{(x^{(1)})^\top c'}{2^{k-1}} \leq \frac{n(3n)^m}{2^{k-1}}.$$

So, $k \leq \log(3^m n^{m+1}) + 1 = O(m \log n)$ as desired. $\qquad\square$

Fix a phase $k \in [n]$ and denote $m_k = |E(G^{(k)})|$. It is helpful to classify the iterations of the Newton–Dinkelbach method based on the magnitude by which the supergradient changes. Recall that the supergradient at the start of iteration $i > 1$ is given by $\gamma(P^{(i)}) - 1$, where $P^{(i)}$ is the u-u' path returned by GRAPEVINE in the previous iteration.

Definition C.2. For every $i > 1$, we say that iteration i is *good* if $1 - \gamma(P^{(i)}) \leq \frac{1}{2}(1 - \gamma(P^{(i-1)}))$. Otherwise, we say that it is *bad*.

The next lemma gives a strongly polynomial bound on the number of good iterations.

Lemma C.3. *In each phase $k \in [n]$, the number of good iterations is $O(m_k \log k)$.*

Proof. Let $\mathcal{P}$ be a sequence of u-u' paths in $G_u^{(k)}$ at the start of every iteration of the Newton–Dinkelbach method. Let $\mathcal{P}^* = (P^{(1)}, P^{(2)}, \ldots, P^{(t)})$ be the subsequence of $\mathcal{P}$ restricted to good iterations. We claim that $\gamma(P^{(i+1)}) \geq \sqrt{\gamma(P^{(i)})}$ for all $i < t$. We use the simple inequality that $(1-x)/2 \leq 1 - \sqrt{x}$ for all $x \in \mathbb{R}_+$; one can derive this by rearranging $(\sqrt{x} - 1)^2/2 \geq 0$. This gives

$$1 - \gamma(P^{(i+1)}) \leq \frac{1}{2}\left(1 - \gamma(P^{(i)})\right) \leq 1 - \sqrt{\gamma(P^{(i)})},$$

which proves the claim. Next, enumerate the arcs of each path by $P^{(i)} = (e_1^{(i)}, e_2^{(i)}, \ldots, e_{\ell_i}^{(i)})$. By taking logarithms, the claim can be equivalently stated as

$$\sum_{j=1}^{\ell_{i+1}} \log \gamma_{e_j^{(i+1)}} \geq \frac{1}{2} \sum_{j=1}^{\ell_i} \log \gamma_{e_j^{(i)}}.$$

Note that both sides of the expression above are negative because $\gamma(P^{(i)}) < 1$ for all $i \in [t]$. Let $c \in \mathbb{R}_+^{m_k}$ be the vector defined by $c_e = |\log \gamma_e|$ for all $e \in E(G^{(k)})$. In addition, for every $i \in [t]$, define the vector $x^{(i)} \in \mathbb{Z}^m$ as

$$x_e^{(i)} = -\operatorname{sgn}(\log \gamma_e) \left| \left\{ j \in [\ell_i] : e_j^{(i)} = e \right\} \right|.$$

Then, we obtain

$$0 < c^\top x^{(i+1)} = \sum_{j=1}^{\ell_{i+1}} -\log \gamma_{e_j^{(i+1)}} \le \frac{1}{2} \sum_{j=1}^{\ell_i} -\log \gamma_{e_j^{(i)}} = \frac{1}{2} c^\top x^{(i)}.$$

for all $i < t$. Since $\|x^{(i)}\|_1 \le k$ for all $i \in [t]$, we conclude that $t = O(m_k \log k)$ by Lemma C.1. $\quad\square$

It is left to bound the number of bad iterations. We approach this by arguing that in a strongly polynomial number of bad iterations, an arc will no longer appear in future paths produced by the Newton–Dinkelbach method.

Lemma C.4. *In each phase $k \in [n]$, the number of bad iterations is $O(m_k \log k)$.*

Proof. Let $\bar{\mathcal{Y}} = (\bar{y}^{(1)}, \bar{y}^{(2)}, \dots, \bar{y}^{(\ell)})$ and $\mathcal{P} = (P^{(1)}, P^{(2)}, \dots, P^{(\ell)})$ be a sequence of node labels and u-u' paths in $G_u^{(k)}$ respectively at the start of every iteration of the Newton–Dinkelbach method. Without loss of generality, we may assume that $\bar{y}^{(i)}$ is finite for all $i \in [\ell]$. For each $i \in [\ell]$, define $y^{(i)} \in \mathbb{R}^n$ as $y_u^{(i)} := \bar{y}_{u'}^{(i)}$ and $y_v^{(i)} := \bar{y}_v^{(i)}$ for all $v \notin \{u, u'\}$. Now, pick an iteration $j \in [\ell]$ such that more than $\log(2n)$ bad iterations have elapsed. Consider the reduced cost $c' \in \mathbb{R}^{m_k}$ given by $c'_{vw} := c_{vw} + \gamma_{vw} y_w^{(j)} - y_v^{(j)}$ for all $vw \in E(G^{(k)})$. Note that $c'_{vw} \ge 0$ for all $v \ne u$.

According to Lemma 4.7, each $P^{(i)}$ is a shortest u-u' path with respect to $\bar{y}^{(i)}$. By complementary slackness, the unit flow on $P^{(i)}$ is an optimal primal solution to $f(\bar{y}_{u'}^{(i)})$. Since $\bar{y}_{u'}^{(i)} > \bar{y}_{u'}^{(i+1)}$ for all $i < \ell$, the sequence $\mathcal{P}$ satisfies subpath monotonicity at u by Lemma 4.10. Define the vector $x \in \mathbb{R}_+^m$ as

$$x_{vw} := \begin{cases} \max_{i \in [\ell]} \left\{ \gamma(P_{uv}^{(i)}) : vw \in E(P^{(i)}) \right\} & \text{if } vw \in \cup_{i=1}^\ell E(P^{(i)}), \\ 0 & \text{otherwise.} \end{cases}$$

Observe that x_{vw} is the gain factor of the u-v subpath of the last path in $\mathcal{P}$ which contains vw, due to subpath monotonicity.

Claim C.5. *We have $-f(\bar{y}_{u'}^{(j)}) < \|c' \circ x\|_\infty$.*

Proof. For every $i \in [\ell]$, we have

$$f(\bar{y}_{u'}^{(i)}) = c(P^{(i)}) - \bar{y}_{u'}^{(i)}(1 - \gamma(P^{(i)})) = c'(P^{(i)}) - (\bar{y}_{u'}^{(i)} - \bar{y}_{u'}^{(j)})(1 - \gamma(P^{(i)})).$$

By applying the definition of $\bar{y}_{u'}^{(i)}$, we can upper bound its negation by

$$-f(\bar{y}_{u'}^{(i)}) = -c'(P^{(i)}) + \frac{1 - \gamma(P^{(i)})}{1 - \gamma(P^{(i-1)})} c'(P^{(i-1)}) \le \left| c'(P^{(i)}) \right| + \left| c'(P^{(i-1)}) \right| \le 2k \|c' \circ x\|_\infty.$$

Lemma 3.1 tells us that $-f(\bar{y}_{u'}^{(i)})$ is nonnegative and monotonically decreasing. Moreover, it decreases geometrically by a factor of $1/2$ during bad iterations. Hence, by our choice of j, we obtain

$$-f(\bar{y}_{u'}^{(j)}) < \left(\frac{1}{2} \right)^{\log(2n)} \cdot 2k \|c' \circ x\|_\infty = \|c' \circ x\|_\infty. \qquad \square$$

Let $d \in \mathbb{R}^{m_k}$ be the arc costs defined by

$$d_{vw} = \begin{cases} c'_{vw} & \text{if } v \neq u, \\ c'_{vw} - f(\bar{y}^{(j)}_{u'}) & \text{if } v = u. \end{cases}$$

Since $f(\bar{y}^{(j)}_{u'}) = \bar{y}^{(j)}_u - \bar{y}^{(j)}_{u'}$, observe that $d \geq 0$ because $\bar{y}^{(j)}_V$ is feasible to the dual LP for $f(\bar{y}^{(j)}_{u'})$.

Claim C.6. *We have $\|d \circ x\|_\infty \geq \|c' \circ x\|_\infty$.*

Proof. Let $e^* = \arg\max_{e \in E(G^{(k)})} |c'_e x_e|$. The claim is trivial unless $e^* \in \cup_{i=1}^{\ell} E(P^{(i)})$ and the tail of e^* is u. Since $f(\bar{y}^{(j)}_{u'}) \leq 0$ and $d_{e^*} = c'_{e^*} - f(\bar{y}^{(j)}_{u'})$, it suffices to show that $c'_{e^*} \geq 0$. For the purpose of contradiction, suppose that $c'_{e^*} < 0$. Since $d_{e^*} \geq 0$, this implies that $|c'_{e^*}| \leq -f(\bar{y}^{(j)}_{u'}) < \|c' \circ x\|_\infty$ using Claim C.5. By the definition of x, $x_{e^*} = 1$ because e^* is the first arc of any path in $\mathcal{P}$ which uses it. However, this implies that

$$|c'_{e^*}| = |c'_{e^*} x_{e^*}| = \|c' \circ x\|_\infty,$$

which is a contradiction. $\qquad\square$

Consider the arc $e^* := \arg\max_{e \in E} |d_e x_e|$. We claim that e^* does not appear in subsequent paths in $\mathcal{P}$ after iteration j. For the purpose of contradiction, suppose that there exists an iteration $i > j$ such that $e^* \in E(P^{(i)})$. Pick the iteration i such that $P^{(i)}$ is the last path in $\mathcal{P}$ which contains e^*. Since the iterates $\bar{y}^{(\cdot)}_{u'}$ are monotonically decreasing, we have

$$0 > \bar{y}^{(i+1)}_{u'} - \bar{y}^{(j)}_{u'} = \frac{c(P^{(i)})}{1 - \gamma(P^{(i)})} - \bar{y}^{(j)}_{u'} = \frac{c'(P^{(i)})}{1 - \gamma(P^{(i)})} = \frac{d(P^{(i)}) - f(\bar{y}^{(j)}_{u'})}{1 - \gamma(P^{(i)})}$$

This implies that $d(P^{(i)}) < f(\bar{y}^{(j)}_{u'}) < \|c' \circ x\|_\infty$. However, it contradicts

$$d(P^{(i)}) \geq d_{e^*} x_{e^*} = \|d \circ x\|_\infty \geq \|c' \circ x\|_\infty,$$

where the first inequality is due to our choice of i and the nonnegativity of d, while the second inequality is due to Claim C.6. Repeating the argument above for m times yields the desired bound on the number of bad iterations. $\qquad\square$

The runtime of every iteration of the Newton–Dinkelbach method is dominated by Grapevine. Thus, following the discussion in Appendix B.3, we obtain the following result.

Corollary C.7. *If we replace Algorithm 1 with the Newton–Dinkelbach method in Algorithm 3, then it solves the feasibility of M2VPI linear systems in $O(m^2 n^2 \log n)$ time.*