# A Strongly Polynomial Label-Correcting Algorithm for Linear Systems with Two Variables per Inequality[*]

Zhuan Khye Koh Bento Natura László A. Végh

Department of Mathematics
London School of Economics and Political Science
{z.koh3,b.natura,l.vegh}@lse.ac.uk

**Abstract**

We present a strongly polynomial label-correcting algorithm for solving the feasibility of linear systems with two variables per inequality (2VPI). The algorithm is based on the Newton–Dinkelbach method for fractional combinatorial optimization. We extend and strengthen previous work of Madani (2002) that showed a weakly polynomial bound for a variant of the Newton–Dinkelbach method for solving deterministic Markov decision processes (DMDPs), a special class of 2VPI linear programs. For a 2VPI system with $n$ variables and $m$ constraints, our algorithm runs in $O(mn)$ iterations. Every iteration takes $O(m + n \log n)$ time for DMDPs, and $O(mn)$ time for general 2VPI systems.

The key technical idea is a new analysis of the Newton–Dinkelbach method exploiting gauge symmetries of the algorithm. This also leads to an acceleration of the Newton–Dinkelbach method for general fractional combinatorial optimization problems. For the special case of linear fractional combinatorial optimization, our method converges in $O(m \log m)$ iterations, improving upon the previous best bound of $O(m^2 \log m)$ by Wang et al. (2006).

## 1 Introduction

Linear programming (LP) is one of the most important tools in computer science and operations research. Currently known methods for solving LP, such as the ellipsoid method and interior point methods, are *weakly polynomial* as their running times depend on the magnitude of numerical entries in the input. A major open question in the theory of linear programming 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 [28]. An LP algorithm is *strongly polynomial* if it only uses elementary arithmetic operations (additions, subtractions, multiplications, divisons, 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 occuring in the computations must remain polynomially bounded in the input size.

In order to make progress on this question, significant work has been done in designing strongly polynomial algorithms for special classes of LP. These classes are often obtained by imposing certain restrictions on the constraint matrix $A \in \mathbb{R}^{n\times m}$. In this paper, we consider the classes of LP which arise from restricting the number of nonzero entries in $A$. In particular, let $\mathcal{M}_2(n, m)$ be the set of $n \times m$ matrices with at most two nonzero entries per column. It is easy to see that every LP can

[*]Supported by the ERC Starting Grant ScaleOpt–757481.

be transformed into an equivalent LP whose contraint matrix has at most three nonzero entries per column.

There is a long history of algorithmic developments on the dual feasibility problem, that is, finding a feasible solution to $A^\top y \leq c$ for $A \in \mathcal{M}_2(n,m)$ and $c \in \mathbb{R}^m$. The system $A^\top y \leq c$ is called a *two variable per inequality* system, abbreviated as 2VPI. If we further require that every inequality has at most one positive and at most one negative entry, it is called a *monotone two variable per inequality* system (M2VPI). An efficient reduction is known from 2VPI systems to M2VPI systems [8, 16]. A fundamental property of M2VPI systems is that, whenever bounded, a unique pointwise maximal solution exists.

The first algorithm for the 2VPI feasibility problem was given by Nelson [21]. It is based on the Fourier–Motzkin elimination method, and has a quasi-polynomial running time. Shostak [27] characterized feasibility in terms of cycles and paths in an associated graph. This characterization also led to an algorithm, but it has an exponential worst case behaviour. Nevertheless, Shostak's characterization laid the foundation for all future algorithms.

The first weakly polynomial algorithm for 2VPI systems was given by Aspvall and Shiloach [2]. The main idea is to compute the tightest upper and lower bounds for each variable using a carefully designed binary search. Upon obtaining these bounds, we can simply fix a variable to any value that lies between its upper and lower bounds. Note that this yields a new 2VPI instance with one less variable, so repeating the procedure either yields a feasible solution to the original system, or detects infeasibility by finding contradicting lower and upper bounds. Megiddo [20] gave the first strongly polynomial algorithm, by replacing the binary search framework in Aspvall and Shiloach's algorithm with the parametric search technique [19]. In fact, the notion of a strongly polynomial algorithm was first defined in the same paper (called 'genuinely polynomial'). Subsequently, Cohen and Megiddo [3] devised faster strongly polynomial algorithms for the problem. The current fastest strongly polynomial algorithm is given by Hochbaum and Naor [17], an efficient Fourier–Motzikin elimination with running time of $O(mn^2 \log m)$.

All strongly polynomial algorithms for 2VPI systems are based on the same principle by Aspvall and Shiloach. They maintain an interval of possible values for each variable, and progressively shrink the intervals until a feasible value for a variable is found. A common denominator of these algorithms is the reliance on binary search or parametric search to narrow down the search space. These search techniques are remarkably different from those used in other combinatorial optimization problems with constraint matrices from $\mathcal{M}_2(n,m)$. They include the maximum flow, shortest paths, minimum-cost flow, and generalized flow problems. Most strongly polynomial algorithms for these problems operate by maintaining a feasible solution while moving towards optimality, or by maintaining a superoptimal solution while moving towards feasibility, such as label-correcting algorithms.

**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 $y_u - y_v \leq c_{uv}$ 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 $y_u - y_v > c_{uv}$, then decreasing $y_u$ to $c_{uv} + y_v$ gives a smaller valid distance label at $u$. We terminate with the shortest path labels once all constraints are satisified. The Bellman–Ford algorithm for the shortest paths problem is a particular implementation of the generic label-correcting algorithm; we refer the reader to [1, Chapter 5] for more details.

It is a natural question if label-correcting algorithms can be extended to general M2VPI systems, where constraints are of the form $y_u - \gamma_{uv} y_v \le c_{uv}$ for 'gain/loss factors' $\gamma_{uv} \in \mathbb{R}_{>0}$ associated with each arc. 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 $y^{(0)} \ge y^{(1)} \ge \ldots \ge y^{(k)}$ of labels that are all valid upper bounds on any feasible solution $y$ to the problem. The algorithm either terminates with the unique pointwise maximal solution $y^{(k)} = y^*$, or finds an infeasibility certificate.

The basic label-correcting operation is the 'arc update', decreasing $y_u$ to $\min\{y_u, c_{uv} + \gamma_{uv} y_v\}$ for some arc $(u, v) \in E$. 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, $y_u - y_v \le 0$, and $y_v - \frac{1}{2} y_u \le -1$. 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 $y_u - \frac{1}{2} y_u \le -1$.

Shostak's [27] result demonstrates that arc updates, together with such 'cycle updates' should be sufficient for finite termination. In this paper, we present 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 $\gamma \le \mathbb{1}$ is known as *deterministic Markov decision process* (DMDP). In this problem, 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. The

$$\begin{aligned} \min\ & c^\top x && && \text{(P)} \\ \text{s. t.}\ & \sum_{u:uv\in E} x_{uv} - \sum_{u:vu\in E} \gamma_{vu} x_{vu} = 1 && \forall v \in V \\ & x \ge 0 \end{aligned}$$

$$\begin{aligned} \max\ & \mathbb{1}^\top y && && \text{(D)} \\ \text{s. t.}\ & y_u - \gamma_{uv} y_v \le c_{uv} && \forall uv \in E \end{aligned}$$

standard policy iteration, value iteration, and simplex algorithms can be all interpreted as variants of the label-correcting framework.[1] 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 $\gamma_{uv}$ is at most $\gamma'$ 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. However, if the discount factors can be arbitrarily close to 1, then Feinberg and Huang [9] showed that value iteration cannot reveal the optimal policy in strongly polynomial time even for DMDPs. Post and Ye [23] proved that simplex with the highest gain pivoting rule is strongly polynomial for DMDPs. Their analysis heavily relies on the assumption $\gamma \le \mathbb{1}$, and does not seem to extend to general M2VPI systems.

**Parametric search and Newton's method.** Before detailing our approach, it is instructive to discuss fractional combinatorial optimization problems. A key ingredient of solving M2VPI problems is finding the best cycle bounds on a variable $y_u$; this problem can be formulated as minimizing the ratio $f(x)/g(x)$, where $x \in \{0, 1\}^E$ is the characteristic vector of a closed walk through node $u$, and $f, g : \{0, 1\}^E \to \mathbb{R}$ are certain functions, $g$ strictly positive. Such problems

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

are well-understood for linear functions $f$ and $g$; a key difficulty in M2VPI systems is that the functions are nonlinear.

Ratio minimization can be reduced to the parametric problem $\min_x f(x)-\delta g(x)$, with a parameter $\delta \in \mathbb{R}$. The optimal ratio is given by the largest value of $\delta$ where the minimum is nonnegative. If we can solve the minimization problem for a fixed value of $\delta$, then we can use binary search to find the optimal value. However, this only yields a weakly polynomial algorithm.

Consider now the case where $f$ and $g$ are both linear functions. Classical examples include finding a minimum cost-to-time ratio cycle and computing a minimum ratio spanning tree. A seminal paper by Megiddo [19] introduced the *parametric search* technique to solve linear fractional combinatorial optimization problems. On a very high level, parametric search works by simulating the algorithm for the nonfractional problem $\min_x f(x) - \delta g(x)$, with the parameter $\delta \in \mathbb{R}$ being indeterminate.

A natural alternative approach is to adapt a standard root finding algorithm such as Newton's method for the optimal value of $\delta$. The *discrete Newton* method, also known as *Dinkelbach's method* [7] has been well-known in the fractional programming literature. Radzik [24] analyzed the convergence of the Newton–Dinkelbach method for linear fractional combinatorial optimization problems, and gave a strongly polynomial bound. We refer to a comprehensive survey by Radzik [25] for details.

Let us now turn to the nonlinear problem $\min f(x)/g(x)$ arising in 2VPI systems. For the corresponding parametric problem $\min_x f(x)-\delta g(x)$, Apsvall and Shiloach [2] introduced the *'Grapevine'* algorithm. This is a natural modification of the Bellman-Ford algorithm; it can be used to determine the sign of the minimum value for any fixed $\delta \in \mathbb{R}$. Aspvall and Shiloach used this subroutine in a binary search framework to obtain a weakly polynomial algorithm.

Megiddo [20] extended the parametric search technique to the 2VPI setting, thus obtaining the first strongly polynomial algorithm. Hochbaum and Naor [17] introduced an elegant variant of Fourier–Motzkin elimination. The usual drawback of Fourier–Motzkin elimination is the exponential increase in the number of constraints. They show that this can be avoided by reducing the number of parallel arcs, using the Grapevine algorithm in a binary search framework on the breakpoints.

The Newton–Dinkelbach algorithm is also naturally applicable to the 2VPI setting. However, Radzik's [24] analysis does not carry over, due to the nonlinearity of $f$ and $g$. Madani [18] used a variant of the Newton–Dinkelbach method as a tool to analyze the convergence of policy iteration on deterministic MDPs. A weakly polynomial runtime of $O(mn^2 \log m \log W)$ was derived for a variant of policy iteration, where $W$ denotes the magnitude of the largest number in the problem description. It was also shown that this algorithm can be adapted to solve 2VPI systems in $O(mn^3 \log(nW))$ time. It was left open whether the algorithm is strongly polynomial already for DMDPs.

**Our results and techniques.** We extend and strengthen Madani's work, by proving that the Newton–Dinkelbach method converges in a strongly polynomial number of iterations. Using this result, we develop a strongly polynomial label-correcting algorithm for solving the feasibility of M2VPI systems. It returns a feasible solution or reports infeasibility within $O(mn)$ iterations of the Newton–Dinkelbach method. As every iteration takes $O(mn)$ time, our algorithm terminates in $O(m^2n^2)$ time. When it is used in the setting of deterministic MDPs, every iteration takes $O(m + n \log n)$ time. Thus, our algorithm returns an optimal solution to (D) or reports unboundedness of (P) in $O(mn(m + n \log n))$ time.

Our strongly polynomial analysis is based on the idea of 'variable fixing'. It was first introduced in the seminal work of Tardos [29] on minimum-cost flows, and has been a central idea of strongly polynomial algorithms. At a very high-level, our analysis is similar to that of the Goldberg–Tarjan minimum-mean cycle cancellation algorithm for the minimum-cost flow problem [12, 26]. Their algorithm proceeds through 'cancelling' a sequence of cycles, and they show that every cycle in the sequence uses an arc that will not be used anymore after a strongly polynomial number of iterations.

In our algorithm, nodes are admitted to the graph one-by-one, using the 'unfreezing' idea of Madani. Let $G_k$ be the current subgraph where $u$ was the most recently admitted node; we use the Newton–Dinkelbach method to find the best 'cycle bound' attainable at $u$ in $G_k$. We use a similar variable fixing argument: a cycle used in the current iteration includes an arc that will not be used again after a certain number of iterations. The details of the argument are significantly more complicated than in [12, 26].

A key idea is to exploit a certain 'gauge symmetry' of the Newton–Dinkelbach algorithm for 2VPI systems. We show that the algorithm is naturally invariant under shifting the variables by an arbitrary potential vector. This generalizes the idea of changing a cost function in the minimum-cost flow problem to a reduced cost; a 'gauge symmetry' of the Goldberg–Tarjan algorithm. Thus, we can find the most convenient potential shifting and argue about running the algorithm for the shifted instance.

Using this perspective, we obtain an enhanced variant of the Newton–Dinkelbach method not only for M2VPI systems, but for general fractional combinatorial optimization problems. Recall that the algorithm aims to minimize $f(x)/g(x)$ where $x \in \mathcal{X}$ belongs to a discrete set, such as the set of cycles in a graph. The algorithm proceeds through iterates $x^{(i)}$, with function values $f^{(i)} = f(x^{(i)})$, $g^{(i)} = g(x^{(i)})$, $\delta^{(i+1)} = f^{(i)}/g^{(i)}$, and $h^{(i)} = f^{(i)} - \delta^{(i)} g^{(i)}$. Radzik's analysis [25] shows that all sequences $\delta^{(i)}$, $g^{(i)}$, and $h^{(i)}$ are strictly monotone decreasing, and in every iteration, either $g^{(i)}$ or $h^{(i)}$ must decrease by a constant factor (see Lemma 3.2). An iteration is 'good', if $g^{(i)}$ decreases by a constant factor, otherwise, the iteration is 'bad'. The complexity bottleneck arises from possibly long sequences of consecutive bad iterations.

We propose a 'look-ahead' version of the Newton–Dinkelbach method. The intuition is that, at every bad iteration, we try to jump over a bad iteration sequence at once, by guessing the $\delta$ value that would be reached after such a sequence. This guess may fail, i.e. fall below the optimal value. In this case, we continue with the original iterate.

We exploit an invariance of the Newton–Dinkelbach method which implies that it suffices to bound the number of iterations under the assumption that the optimal value of the ratio is 0. Under this assumption, in our look-ahead algorithm, the value of $f^{(i)}$ decreases by a constant factor in every two iterations.

Radzik's first analysis [24] of the Newton–Dinkelbach method for linear fractional combinatorial optimization yielded a bound of $O(m^4 \log^2 m)$ iterations, improved to $O(m^2 \log^2 m)$ in [25], and later to $O(m^2 \log m)$ by Wang et al. [33]. Using the geometric decrease in the $f^{(i)}$ values, we can prove that our look-ahead version terminates in $O(m \log m)$ iterations.

**Related work.** For DMDPs, Post and Ye [23] proved that simplex with the highest gain pivoting rule takes $O(m^2 n^3 \log^2 n)$ iterations if every arc has the same discount factor, and $O(m^3 n^5 \log^2 n)$ iterations if every arc has a possibly distinct discount factor. The bound for uniform discount was later improved by Hansen et al. [13] to $O(m^2 n^2 \log^2 n)$.

Strongly polynomial bounds are also known for discounted *nondeterministic* Markov Decision Processes. Ye [35] gave an interior-point algorithm that is strongly polynomial for fixed discount factors. Later, Ye [36] showed that for fixed discount factors, simplex is also strongly polynomial;

this was improved by Hansen et al. [14].

The primal feasibility problem, that is, finding a feasible solution to $Ax = b, x \geq 0$ for $A \in \mathcal{M}_2(n, m)$ and $b \in \mathbb{R}^n$, can be reduced to generalized flow maximization [32]. Consequently, it can be solved via recently developed strongly polynomial algorithms for the latter problem by Végh [32], and by Olver and Végh [22].

Other restrictions on the constraint matrix were considered. Under the assumption that $A$ is integral, Tardos [30] devised an algorithm with running time $\mathrm{poly}(n, m, \Delta)$, where $\Delta$ is an upper bound on the largest subdeterminant of $A$. Hence, if every entry of $A$ has size $\mathrm{poly}(n, m)$, then this algorithm is strongly polynomial. On the other hand, Vavasis and Ye [31] gave an interior point method which does not rely on the integrality assumption. This is achieved by replacing $\Delta$ with a more general condition number $\bar{\chi}_A$ of the matrix. This was recently improved to a dependence of the optimized variant $\bar{\chi}^*_A$ by Dadush et al. [5].

Another interesting application of the Newton–Dinkelbach method in combinatorial optimization is parametric submodular function minimization, where Goemans et al. [11] showed a strongly polynomial bound.

**Paper organization.** In Section 2, we give some preliminaries and introduce notation. In Section 3, we introduce the classical Newton–Dinkelbach method for fractional combinatorial optimization, and present an accelerated version. Then, we demonstrate a simple analysis for linear functions in Section 4, before delving into the 2VPI problem in Section 5. Finally, Section 6 contains our results on deterministic Markov decision processes. Lemmas marked with (*) are proven in the appendix.

## 2 Preliminaries

Let $\mathbb{R}_+$ and $\mathbb{R}_{++}$ denote the nonnegative and positive reals respectively. Let $\bar{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. Every 2VPI system with $n$ variables and $m$ constraints can be transformed into an equivalent M2VPI system with $2n$ variables and $2m$ constraints in $O(m)$ time [8, 16]. Hence, we may restrict our attention to M2VPI systems. Without loss of generality, we may assume that every inequality is of the form $y_u - \gamma_e y_v \leq c_e$, where $\gamma_e \in \mathbb{R}_{++}$ and $c_e \in \mathbb{R}$. Note that inequalities involving only one variable can also be expressed in this form, e.g.

$$y_u \leq c_e \iff y_u - 0.5y_u \leq 0.5c_e \qquad\qquad y_u \geq c_e \iff y_u - 2y_u \leq -c_e.$$

Thus, the system can be naturally represented as a directed multigraph $G = (V, E)$, with arc costs $c \in \mathbb{R}^E$ and gain factors $\gamma \in \mathbb{R}^E_{++}$. In particular, we add a node $v$ for every variable $y_v$, and add an arc $e$ from $u$ to $v$ with cost $c_e$ and gain factor $\gamma_e$ for every inequality $y_u - \gamma_e y_v \leq c_e$. Notice that $G$ may contain self-loops, and they correspond to inequalities with only one variable. Let $n := |V|$ and $m := |E|$. For an arc set $F \subseteq E$, let $\overleftarrow{F} := \{vu : uv \in F\}$ denote the set of reversed arcs.

For a $u$-$v$ walk $P$ in $G$ with $E(P) = (e_1, e_2, \ldots, e_k)$, we define its *cost* and *gain factor* as

$$c(P) := \sum_{i=1}^{k} \left( \prod_{j=1}^{i-1} \gamma_{e_j} \right) c_{e_i} \qquad\qquad \gamma(P) := \prod_{i=1}^{k} \gamma_{e_i}$$

respectively. If $P$ is a singleton, i.e. $V(P) = \{u\}$ and $E(P) = \emptyset$, then $c(P) := 0$ and $\gamma(P) := 1$. The walk $P$ gives rise to the inequality $y_u \leq c(P) + \gamma(P) y_v$, which is implied by the sequence of arcs/inequalities in $E(P)$. If $P$ is a $u$-$u$ walk such that $u$ does not occur as an intermediate node, then it is called a *loop at $u$*. In addition, if its intermediate nodes are distinct, then it is called a *cycle at $u$*. Note that the singleton $(u)$ is considered a trivial 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 $Q$.

For 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 cycle at $u$ is called a *shortest cycle at $u$ with respect to $y$* if it has the smallest $y$-cost among all loops at $u$. Similarly, 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 cycle or path does not always exist.

**Definition 2.1.** A loop $C$ 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 loop $C$ is *negative* if $c(C) < 0$.

**Definition 2.2.** Given a flow-generating loop $C$ at $u$, a flow-absorbing loop $D$ at $v$, and a walk $P$ from $u$ to $v$, the graph $C \cup P \cup D$ is called a *biloop*. If $C, D$ are cycles and $P$ is a path, then the biloop is also known as a *bicycle*. We say that the biloop 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 2.3** ([27])**.** *An M2VPI system $(G, c, \gamma)$ is infeasible if and only if $G$ contains a negative unit-gain cycle or a negative bicycle.*

Theorem 2.3 also holds when we replace the cycle and bicycle with loop and biloop respectively. We will utilize this fact when applying the converse to certify infeasibility. The following lemma, given by Radzik and credited to Goemans in [25], will come in handy later when bounding the length of a geometrically decreasing sequence of sums.

**Lemma 2.4** ([25])**.** *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 $k = O(m \log m)$.*

We will strengthen it slightly for our purposes. The proof remains largely the same.

**Lemma 2.5** (*)**.** *Let $c \in \mathbb{R}^m_+$ and $x^{(1)}, x^{(2)}, \ldots, x^{(k)} \in \{-1, 0, 1\}^m$ such that $\left|\operatorname{supp}(x^{(i)})\right| \leq n$ 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 $k = O(m \log n)$.*

Finally, in the rest of the paper, we use log to indicate the binary logarithm.

## 3 The Newton–Dinkelbach Method

We consider fractional combinatorial optimization problems. An instance consists of a finite set $E$ of *elements*, a set $\mathcal{X} \subseteq \{0,1\}^E$ of *structures*, and two functions $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}_{++}$. Our goal is to solve the following optimization problem:

$$\min_{x \in \mathcal{X}} \frac{f(x)}{g(x)}. \tag{$\mathcal{F}$}$$

It will be more convenient to work with the following reformulation of ($\mathcal{F}$):

$$\begin{aligned} &\max\ \delta \\ &\text{s. t.}\ -f(x)+\delta g(x) \leq 0 \qquad \forall x \in \mathcal{X}. \end{aligned} \tag{$\mathcal{H}$}$$

A pair $(\delta^*, x^*)$ where $\delta^* \in \mathbb{R}$ and $x^* \in \mathcal{X}$ is an optimal solution to ($\mathcal{H}$) if and only if

$$-f(x) + \delta^* g(x) \leq -f(x^*) + \delta^* g(x^*) = 0 \qquad \forall x \in \mathcal{X}.$$

It is easy to see that $x^*$ is an optimal structure to ($\mathcal{F}$) with optimal value $\delta^*$. Now, for any fixed value of $\delta \in \mathbb{R}$, we can test its feasibility and optimality to ($\mathcal{H}$) by solving the following subproblem:

$$h_{f,g}(\delta) := \max_{x\in\mathcal{X}} \{-f(x) + \delta g(x)\}. \tag{$\mathcal{H}_\delta$}$$

When the functions $f$ and $g$ are clear from context, we will omit them from the subscripts of $h$. Observe that $\delta$ is suboptimal when $h(\delta) < 0$, optimal when $h(\delta) = 0$, and infeasible when $h(\delta) > 0$. Thus, solving ($\mathcal{H}$) amounts to finding a root of the function $h$. Since $h$ is the upper envelope of finitely many increasing linear functions, it is convex, increasing and piecewise linear. Consequently, it has a unique root.

A common technique used to solve fractional combinatorial optimization problems is the Newton–Dinkelbach method. It is based on the aforementioned root-finding perspective. Given a subroutine which solves the subproblem ($\mathcal{H}_\delta$), the Newton–Dinkelbach method is an iterative process which generates a new and better upper estimate of the optimal value $\delta^*$ in each iteration. At the beginning of iteration $i \in \mathbb{N}$, we have an *iterate-structure* pair $(\delta^{(i)}, x^{(i)})$ such that $h(\delta^{(i)}) = -f(x^{(i)}) + \delta^{(i)} g(x^{(i)})$, where $\delta^{(i)} \geq \delta^*$ denotes the current estimate on the optimal value. If $h(\delta^{(i)}) = 0$, then we are done. Otherwise, we set $\delta^{(i+1)} := f(x^{(i)})/g(x^{(i)})$ and compute a structure $x^{(i+1)} \in \mathcal{X}$ such that $h(\delta^{(i+1)}) = -f(x^{(i+1)}) + \delta^{(i+1)} g(x^{(i+1)})$. Then, a new iteration begins.

To kick-start this process, we need to supply the Newton–Dinkelbach method with an initial iterate $\delta^{(1)}$ and compute a structure $x^{(1)} \in \mathcal{X}$ such that $h(\delta^{(1)}) = -f(x^{(1)}) + \delta^{(1)} g(x^{(1)})$. We can pick $\delta^{(1)}$ arbitrarily. In fact, we may assume that $\delta^{(1)} \geq \delta^*$ without loss of generality. Indeed, if $\delta^{(1)} < \delta^*$, then $\delta^{(i)} \geq \delta^*$ for all $i \geq 2$ because $h$ is convex and increasing. For the sake of brevity, we will use the following notation:

$$f^{(i)} := f(x^{(i)}) \qquad g^{(i)} := g(x^{(i)}) \qquad h^{(i)} := h(\delta^{(i)}) = -f^{(i)} + \delta^{(i)} g^{(i)}.$$

Note that $-f^{(i)}$ and $g^{(i)}$ represents the vertical intercept and gradient of the linear segment of $h$ at $\delta^{(i)}$ respectively. With this notation, we also have $\delta^{(i+1)} = f^{(i)}/g^{(i)}$. The following lemma shows that $\delta^{(i)}$, $g^{(i)}$ and $h^{(i)}$ are monotonically decreasing.

**Lemma 3.1** ([25, Lemma 3.1])**.** *The Newton–Dinkelbach method terminates in a finite number of iterations and*

*(a)* $\delta^{(1)} > \delta^{(2)} > \cdots > \delta^{(k-1)} > \delta^{(k)} = \delta^*$

*(b)* $g^{(1)} > g^{(2)} > \cdots > g^{(k-1)} \geq g^{(k)} > 0$

*(c)* $h^{(1)} > h^{(2)} > \cdots > h^{(k-1)} > h^{(k)} = 0$

The next lemma illustrates the advantage of the Newton–Dinkelbach method. Since $g^{(i)}$ and $h^{(i)}$ are nonnegative, it implies that $g^{(i)}$ or $h^{(i)}$ decreases geometrically in every iteration.

**Lemma 3.2** ([25, Lemma 3.2])**.** *For every iteration $i \geq 1$, we have*

$$\frac{h^{(i+1)}}{h^{(i)}} + \frac{g^{(i+1)}}{g^{(i)}} \leq 1.$$

## 3.1 A Useful Invariance

In this subsection, we introduce a new technique for analyzing the Newton–Dinkelbach method. The key idea is to work with a new instance which is "equivalent" to the original instance. First, we introduce the notion of an *eligible sequence.*

**Definition 3.3.** Given an instance $(\mathcal{X}, f, g)$, let $X := (x^{(1)}, x^{(2)}, \ldots, x^{(k)})$ be a sequence of structures and $\delta^{(1)} \in \mathbb{R}$. We say that $X$ is an *eligible sequence for* $(\mathcal{X}, f, g, \delta^{(1)})$ if

$$h_{f,g}(\delta^{(i)}) = -f(x^{(i)}) + \delta^{(i)} g(x^{(i)})$$

for all $i \in [k]$, where $\delta^{(i+1)} = f(x^{(i)})/g(x^{(i)})$ for all $i < k$.

An eligible sequence for $(\mathcal{X}, f, g, \delta^{(1)})$ is a sequence of structures that could be produced by the Newton–Dinkelbach method when it is run on the instance $(\mathcal{X}, f, g)$ and initialized with $\delta^{(1)}$. Now, let $\lambda \in \mathbb{R}$ and consider the function $f^\lambda : \mathcal{X} \to \mathbb{R}$ defined by $f^\lambda(x) := f(x) - \lambda g(x)$. The next lemma illustrates an invariance of the Newton–Dinkelbach method when $f$ is replaced by $f^\lambda$.

**Lemma 3.4.** *Let $(\mathcal{X}, f, g)$ be an instance and $\delta^{(1)} \in \mathbb{R}$. For any $\lambda \in \mathbb{R}$, an eligible sequence for $(\mathcal{X}, f, g, \delta^{(1)})$ is also an eligible sequence for $(\mathcal{X}, f^\lambda, g, \delta^{(1)} - \lambda)$.*

*Proof.* Let $X = (x^{(1)}, x^{(2)}, \ldots, x^{(k)})$ be an eligible sequence for $(\mathcal{X}, f, g, \delta^{(1)})$, with a sequence of corresponding iterates $(\delta^{(1)}, \delta^{(2)}, \ldots, \delta^{(k)})$. Let $\lambda \in \mathbb{R}$, and consider the new instance $(\mathcal{X}, f^\lambda, g)$. Its subproblem can be described by the following function

$$h^\lambda(\delta) := \max_{x \in \mathcal{X}} \left\{ -f^\lambda(x) + \delta g(x) \right\}.$$

For any $\delta \in \mathbb{R}$, observe that

$$\begin{aligned} h^\lambda(\delta - \lambda) &= \max_{x \in \mathcal{X}} \left\{ -f^\lambda(x) + (\delta - \lambda) g(x) \right\} \\ &= \max_{x \in \mathcal{X}} \{ -f(x) + \lambda g(x) + (\delta - \lambda) g(x) \} \\ &= \max_{x \in \mathcal{X}} \{ -f(x) + \delta g(x) \} = h(\delta) \end{aligned}$$

Hence, $h^\lambda$ is a horizontal translation of the function $h$ by $-\lambda$. Consequently, we obtain

$$x^{(1)} \in \arg\max_{x \in \mathcal{X}} \left\{ -f(x) + \delta^{(1)} g(x) \right\} = \arg\max_{x \in \mathcal{X}} \left\{ -f^\lambda(x) + (\delta^{(1)} - \lambda) g(x) \right\}.$$

The next iterate for this instance is given by

$$\frac{f^\lambda(x^{(1)})}{g(x^{(1)})} = \frac{f(x^{(1)})}{g(x^{(1)})} - \lambda = \delta^{(2)} - \lambda.$$

Thus, repeating the same argument yields the desired conclusion. $\square$

Since the set of eligible sequences for $(\mathcal{X}, f, g, \delta^{(1)})$ and $(\mathcal{X}, f^\lambda, g, \delta^{(1)} - \lambda)$ coincide, the following consequence is immediate.

**Lemma 3.5.** *Let $(\mathcal{X}, f, g)$ be an instance and $\delta^{(1)} \in \mathbb{R}$. For any $\lambda \in \mathbb{R}$, the Newton–Dinkelbach method proceeds via an identical sequence of structures for the instances $(\mathcal{X}, f, g)$ and $(\mathcal{X}, f^\lambda, g)$, when initialized with $\delta^{(1)}$ and $\delta^{(1)} - \lambda$ respectively.*

Given an input instance $(\mathcal{X}, f, g)$, let $\delta^*$ denote its optimal value. If we pick $\lambda = \delta^*$, then the optimal value of the instance $(\mathcal{X}, f^{\delta^*}, g)$ is zero. From Lemma 3.5, the Newton–Dinkelbach method produces the same sequence of structures for the instances $(\mathcal{X}, f, g)$ and $(\mathcal{X}, f^{\delta^*}, g)$, when initialized with $\delta^{(1)}$ and $\delta^{(1)} - \delta^*$ respectively. Thus, we may assume that the input instance is $(\mathcal{X}, f^{\delta^*}, g)$ instead. In particular, we may assume that the optimal value of every input instance is zero without loss of generality. Note that this assumption does not require the Newton–Dinkelbach method to know the value of $\delta^*$, as it only shows up in the analysis. With this assumption, we can strengthen Lemma 3.1.

**Lemma 3.6.** *If the optimal value of* ($\mathcal{F}$) *is zero, then the Newton–Dinkelbach method terminates in a finite number of iterations and*

*(a)* $\delta^{(1)} > \delta^{(2)} > \cdots > \delta^{(k-1)} > \delta^{(k)} = 0$

*(b)* $f^{(1)} > f^{(2)} > \cdots > f^{(k-1)} \geq f^{(k)} = 0$

*(c)* $g^{(1)} > g^{(2)} > \cdots > g^{(k-1)} \geq g^{(k)} > 0$

*(d)* $h^{(1)} > h^{(2)} > \cdots > h^{(k-1)} > h^{(k)} = 0$

## 3.2 Accelerating the Newton–Dinkelbach Method

In this subsection, we present an accelerated version of the Newton–Dinkelbach method. First, it is helpful to classify the iterations based on the magnitude by which $g^{(i)}$ decreases.

**Definition 3.7.** Let $\alpha \in (0.5, 1)$ be a constant. For every $i \in \mathbb{N}$, we say that iteration $i$ is *good* if $g^{(i+1)} \leq \alpha g^{(i)}$. Otherwise, we say that it is *bad*.

The key idea of the acceleration is to skip over a sequence of bad iterations whenever it is possible to do so. In particular, every time we encounter a bad iteration, we pessimistically assume that there will be a long sequence of consecutive bad iterations ahead of us. In order to avoid this, we aggressively guess the final iterate of this sequence. We call this action *look-ahead*.

**Algorithm 1:** Look-ahead Newton–Dinkelbach method

**input :** A fractional combinatorial optimization instance $(\mathcal{X}, f, g)$ and a subroutine ORACLE($\delta$) which returns an optimal solution to the subproblem ($\mathcal{H}_\delta$).
**output:** An optimal solution to ($\mathcal{F}$).

```
1  Pick $\delta^{(1)} \in \mathbb{R}$ arbitrarily
2  $x^{(1)} \leftarrow$ ORACLE($\delta$)
3  $i \leftarrow 1$
4  while $h(\delta^{(i)}) \neq 0$ do
5      $\delta \leftarrow f(x^{(i)})/g(x^{(i)})$
6      $x \leftarrow$ ORACLE($\delta$)
7      if $g(x) > \alpha g(x^{(i)})$ then                      /* Bad iteration */
8          $\delta' \leftarrow \delta - \frac{1-\alpha}{2\alpha-1}(\delta^{(i)} - \delta)$     /* Guess next iterate */
9          $x' \leftarrow$ ORACLE($\delta'$)
10         if $h(\delta') \geq 0$ then                          /* $\delta' \geq \delta^*$ */
11             $\delta \leftarrow \delta'$, $x \leftarrow x'$
12     $\delta^{(i+1)} \leftarrow \delta$, $x^{(i+1)} \leftarrow x$
13     $i \leftarrow i + 1$
14 return $x^{(i)}$
```

An iteration of Algorithm 1 refers to a repetition of the `while` loop. Lines 7–11 implement the look-ahead procedure, where $\delta'$ denotes our guess in every iteration. Let $\delta^*$ be the unique root of $h$. Since $h$ is an increasing function, we know that $h(\delta') < 0$ if and only if $\delta' < \delta^*$. As long as $\delta' \geq \delta^*$, the algorithm picks $\delta'$ as the next iterate. In an iteration, we say that look-ahead is *successful* if $\delta$ is replaced by $\delta'$. It is easy to prove that Lemma 3.4 also holds for Algorithm 1. Therefore, the invariance given by Lemma 3.5 apply here as well.

**Lemma 3.8.** *If look-ahead is successful, then the current iteration is good.*

*Proof.* Fix an iteration $i \in \mathbb{N}$, and assume that look ahead is successful in iteration $i$. Let us run the standard Newton–Dinkelbach method on the same instance, initialized with the iterate-structure pair $(\delta^{(i)}, x^{(i)})$. From Lemma 3.1, we know that it terminates in $\ell$ iterations for some $\ell \in \mathbb{N}$. For every $j \in [\ell]$, let $(\bar{\delta}^{(j)}, \bar{x}^{(j)})$ denote the corresponding iterate-structure pair at the start of iteration $j$. Note that $\bar{\delta}^{(1)} = \delta^{(i)}$ and $\bar{x}^{(1)} = x^{(i)}$. Also, recall that $\bar{\delta}^{(j+1)} = f(\bar{x}^{(j)})/g(\bar{x}^{(j)})$ for all $j \in [\ell-1]$.

Since Algorithm 1 made a guess in iteration $i$, we know that $g(\bar{x}^{(2)}) > \alpha g(\bar{x}^{(1)})$. We claim that there exists an iteration $2 \leq k \leq \ell - 2$ such that $g(\bar{x}^{(k+1)}) \leq \alpha g(\bar{x}^{(k)})$. For the purpose of contradiction, suppose that $g(\bar{x}^{(j+1)}) > \alpha g(\bar{x}^{(j)})$ for all $j \in [\ell-2]$. By Lemma 3.2, this implies that $h(\bar{\delta}^{(j+1)}) < (1-\alpha) h(\bar{\delta}^{(j)})$ for all $j \in [\ell-2]$. Using these two sets of inequalities, we obtain

$$\begin{aligned}
\bar{\delta}^{(j+1)} - \bar{\delta}^{(j+2)} &= \frac{-f(x^{(j+1)}) + \bar{\delta}^{(j+1)} g(x^{(j+1)})}{g(x^{(j+1)})} \\
&= \frac{h(\bar{\delta}^{(j+1)})}{g(\bar{x}^{(j+1)})} \\
&< \frac{1-\alpha}{\alpha} \cdot \frac{h(\bar{\delta}^{(j)})}{g(\bar{x}^{(j)})} \\
&= \frac{1-\alpha}{\alpha} \cdot \frac{-f(\bar{x}^{(j)}) + \bar{\delta}^{(j)} g(\bar{x}^{(j)})}{g(\bar{x}^{(j)})} = \frac{1-\alpha}{\alpha} \left( \bar{\delta}^{(j)} - \bar{\delta}^{(j+1)} \right)
\end{aligned}$$

for all $j \in [\ell-2]$. This gives us

$$\begin{aligned}
\bar{\delta}^{(2)} - \bar{\delta}^{(\ell)} &= \left( \bar{\delta}^{(2)} - \bar{\delta}^{(3)} \right) + \left( \bar{\delta}^{(3)} - \bar{\delta}^{(4)} \right) + \cdots + \left( \bar{\delta}^{(\ell-1)} - \bar{\delta}^{(\ell)} \right) \\
&< \left( \frac{1-\alpha}{\alpha} + \left( \frac{1-\alpha}{\alpha} \right)^2 + \ldots \left( \frac{1-\alpha}{\alpha} \right)^{\ell-2} \right) \left( \bar{\delta}^{(1)} - \bar{\delta}^{(2)} \right) \\
&< \frac{1-\alpha}{2\alpha - 1} \left( \bar{\delta}^{(1)} - \bar{\delta}^{(2)} \right),
\end{aligned}$$

where the coefficient on the last line is the infinite sum of the geometric series. Rearranging yields

$$\delta^* = \bar{\delta}^{(\ell)} > \bar{\delta}^{(2)} - \frac{1-\alpha}{2\alpha-1} \left( \bar{\delta}^{(1)} - \bar{\delta}^{(2)} \right) = \delta^{(i+1)},$$

which contradicts the success of look-ahead in iteration $i$ of Algorithm 1.

Now, pick $k \geq 2$ as the earliest iteration where $g(\bar{x}^{(k+1)}) \leq \alpha g(\bar{x}^{(k)})$. Then, $g(\bar{x}^{(j+1)}) > \alpha g(\bar{x}^{(j)})$ for all $j \in [k-1]$. By repeating the same argument in the previous paragraph, we arrive at the conclusion $\bar{\delta}^{(k+1)} > \delta^{(i+1)}$. Due to the convexity of $h$, we get

$$g(x^{(i+1)}) \leq h'_+(\delta^{(i+1)}) \leq h'_-(\bar{\delta}^{(k+1)}) \leq g(\bar{x}^{(k+1)}) \leq \alpha g(\bar{x}^{(k)}) < \alpha g(\bar{x}^{(1)}) = \alpha g(x^{(i)}),$$

where the strict inequality is due to Lemma 3.1. Thus, iteration $i$ of Algorithm 1 is good. ◻

The next lemma shows us the advantage of using the look-ahead Newton–Dinkelbach method. In particular, $f^{(i)}$ decreases geometrically every other iteration. This is the part of the analysis where we crucially use the invariance in Section 3.1.

**Lemma 3.9.** *Let $(\mathcal{X}, f, g)$ be an instance with optimal value zero. For every iteration $i > 1$, we have*

$$f^{(i+1)} \leq \max\left\{\alpha, \frac{1-\alpha}{\alpha}\right\} f^{(i-1)}.$$

*Proof.* Fix an iteration $i > 1$. Let us first assume that iteration $i$ is good. Then, we have

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

which gives us

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

where the first and last inequality exploited monotonicity (Lemma 3.6). Next, assume that iteration $i$ is bad. From Lemma 3.8, we know that the guess $\delta'$ performed in iteration $i$ was unsuccessful. Letting $\delta := f^{(i)}/g^{(i)}$, this means that

$$\delta - \frac{1-\alpha}{2\alpha - 1}(\delta^{(i)} - \delta) = \delta' < \delta^* = 0.$$

After rearranging, we obtain

$$\frac{f^{(i)}}{g^{(i)}} = \delta < \frac{1-\alpha}{\alpha}\delta^{(i)} \leq \frac{1-\alpha}{\alpha} \cdot \frac{f^{(i-1)}}{g^{(i-1)}},$$

which then gives us

$$f^{(i+1)} \leq f^{(i)} < \frac{1-\alpha}{\alpha} \cdot \frac{g^{(i)}}{g^{(i-1)}} f^{(i-1)} < \frac{1-\alpha}{\alpha} f^{(i-1)}.$$ □

## 4 Linear Fractional Combinatorial Optimization

Before using the look-ahead Newton–Dinkelbach method to solve 2VPI systems, we first illustrate its utility in a simpler setting—the linear case. Here, the functions $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}_{++}$ are of the form $f(x) = c^\top x$ and $g(x) = w^\top x$ for some vectors $c, w \in \mathbb{R}^m$ where $m = |E|$. For each element $e \in E$, we call $c_e$ and $w_e$ its *cost* and *weight* respectively. Many classical combinatorial optimization problems belong to this class, such as finding a minimum cost-to-time ratio cycle and computing a minimum ratio spanning tree. Megiddo was the first to consider this class of problems in its generality. He developed the parametric search technique [19], and showed that it can solve these problems in strongly polynomial time whenever there is a strongly polynomial *affine* algorithm for the *non-fractional* problem, i.e. $\min_{x\in\mathcal{X}} c^\top x$. Radzik [24] gave a strongly polynomial bound on the number of iterations taken by the Newton–Dinkelbach method for these problems. In particular, an upper bound of $O(m^4 \log^2 m)$ was derived. Subsequently, this was reduced to $O(m^2 \log^2 m)$ by Radzik [25], and further improved to $O(m^2 \log m)$ by Wang et al. [33].

In this section, we show that Algorithm 1 can solve these problems faster. Given an instance $(\mathcal{X}, c, w)$, let $\delta^*$ denote its optimal value. We will rely on the invariance of Algorithm 1 when $c$ is replaced by $c - \delta^* w$. Note that the resulting cost function is still linear.

**Theorem 4.1.** *Algorithm 1 converges in $O(m \log m)$ iterations for linear fractional combinatorial optimization problems.*

*Proof.* Let $(x^{(1)}, x^{(2)}, \dots, x^{(k)})$ be a sequence of solutions produced by Algorithm 1. By Lemma 3.6, we have $c^\top x^{(i)} > 0$ for all $i \leq k-2$. Moreover, by Lemma 3.9, we know that $c^\top x^{(i+1)} \leq \beta c^\top x^{(i-1)}$ for all $1 < i < k$, where $\beta := \max\{\alpha, (1-\alpha)/\alpha\}$. Let $j := 2\lceil -1/\log \beta \rceil$. Then, we get $c^\top x^{(i+j-1)} \leq (1/2) c^\top x^{(i-1)}$ for all $1 < i \leq k-j+1$. Thus, applying Lemma 2.4 yields $k = O(m \log m)$. $\square$

## 5 Monotone Two Variable per Inequality Systems

Recall that an M2VPI system can be represented as a directed multigraph $G = (V, E)$. For a node $u \in V$, let $\mathcal{P}_u(G)$ denote the set of loops at $u$ in $G$. Note that $\mathcal{P}_u(G) \neq \emptyset$ as it contains the trivial cycle at $u$. For each $k \in \mathbb{N}$, let $\mathcal{P}_u^k(G)$ denote the subset of loops in $\mathcal{P}_u(G)$ whose length is at most $k$. When the graph $G$ is clear from context, we will omit it from the notation, i.e., use $\mathcal{P}_u$ and $\mathcal{P}_u^k$. We are interested in the following two types of loops in $\mathcal{P}_u^n$

$$\widehat{\mathcal{P}}_u^n := \{P \in \mathcal{P}_u^n : \gamma(P) < 1\} \qquad \widecheck{\mathcal{P}}_u^n := \{P \in \mathcal{P}_u^n : \gamma(P) > 1\}.$$

Consider a loop $P \in \mathcal{P}_u^n$ where $\gamma(P) \neq 1$. As mentioned in Section 2, the loop $P$ induces the valid inequality $y_u \leq c(P) + \gamma(P) y_u$. If $P \in \widehat{\mathcal{P}}_u^n$, then it yields an upper bound on $y_u$, i.e. $y_u \leq c(P)/(1-\gamma(P))$. On the other hand, if $P \in \widecheck{\mathcal{P}}_u^n$, then it yields a lower bound on $y_u$, i.e. $y_u \geq -c(P)/(\gamma(P)-1)$. Let $y_u^{\text{high}}$ and $y_u^{\text{low}}$ denote the best upper and lower bounds induced by these loops respectively, i.e.

$$y_u^{\text{high}} := \min_{P \in \widehat{\mathcal{P}}_u^n} \left\{ \frac{c(P)}{1-\gamma(P)} \right\} \qquad y_u^{\text{low}} := \max_{P \in \widecheck{\mathcal{P}}_u^n} \left\{ \frac{-c(P)}{\gamma(P)-1} \right\}.$$

As pointed out by Aspvall and Shiloach [2], these quantities play a crucial role in the 2VPI feasibility problem. To see how they ultimately lead to a feasible solution or an infeasibility certificate, we refer to the discussion after Theorem 5.9. Notice that finding $y_u^{\text{high}}$ and $y_u^{\text{low}}$ are fractional combinatorial optimization problems, so one could try solving them using the Newton–Dinkelbach method. However, the analysis will be more complex than the previous section, because the functions corresponding to $f$ and $g$ are nonlinear. Nevertheless, we can still derive a strongly polynomial bound on the iteration complexity of the Newton–Dinkelbach method.

Let us focus on the computation of the best upper bound $y_u^{\text{high}}$ for now. The subproblem that needs to be solved in every iteration is

$$h(\delta) := \max_{P \in \widehat{\mathcal{P}}_u^n} \{-c(P) + \delta(1-\gamma(P))\} \tag{$\mathcal{H}_\delta$}$$

Since

$$\arg\max_{P \in \widehat{\mathcal{P}}_u^n} \{-c(P) + \delta(1-\gamma(P))\} = \arg\min_{P \in \widehat{\mathcal{P}}_u^n} \{c(P) + \delta\gamma(P)\},$$

one can try to solve ($\mathcal{H}_\delta$) by running a label-correcting subroutine on $G$. A suitable candidate subroutine is the so-called Grapevine algorithm, developed by Aspvall and Shiloach [2].

Given initial node labels $y \in \bar{\mathbb{R}}^n$ and a specified node $u$, Grapevine runs for $n$ steps. We call an arc $vw \in E$ *violated with respect to* $y$ if $y_v > c_{vw} + \gamma_{vw} y_w$. At every step, the algorithm lowers $y_v$ until there are no violated arcs in $\delta^+(v)$ with respect to $y$. If there is a violated arc, the algorithm also records a most violated arc in $\delta^+(v)$ with respect to $y$ (ties are broken arbitrarily but consistently). We would like to emphasize that $y_u$ is never changed throughout the algorithm. At the end of step $n$, the algorithm traces a walk $P$ from $u$ by following the recorded arcs in reverse chronological order. Note that $P$ could be the singleton $(u)$ if there are no violated arcs in $\delta^+(u)$. Finally, it returns the updated node labels $y \in \bar{\mathbb{R}}^n$ and the walk $P$. Clearly, the running time of Grapevine is $O(mn)$.

**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 $P$ 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              $pred(v, i) \leftarrow \arg\min_{vw\in\delta^+(v)}\{c_{vw} + \gamma_{vw} y_w\}$      /* Break ties arbitrarily */
6          else
7              $pred(v, i) \leftarrow \emptyset$
8      foreach $v \in V \setminus u$ do
9          $y_v \leftarrow y'_v$
10 Let $P$ be the walk obtained by tracing from $pred(u, n)$
11 return $(y, P)$
```

To solve ($\mathcal{H}_\delta$), we supply Grapevine with initial node labels $y \in \bar{\mathbb{R}}^n$ defined by $y_u := \delta$ and $y_v := \infty$ for all $v \neq u$. This choice guarantees that the return walk $P$ is a loop at $u$. However, we need to be careful here because $P$ might not be from $\widehat{\mathcal{P}}^n_u$. Indeed, Grapevine solves the following subproblem instead:

$$\phi(\delta) := \max_{P\in\mathcal{P}^n_u} \{-c(P) + \delta(1 - \gamma(P))\}\ , \tag{$\Phi_\delta$}$$

that is, optimizing over the larger set $\mathcal{P}^n_u$. Since $\phi$ is the upper envelope of finitely many linear functions, it is convex and piecewise linear. Unlike $h$ however, the function $\phi$ does not need to be increasing. Indeed, $\phi \geq h$ and they coincide at points where the right derivative of $\phi$ is positive.

We are ready to describe the Newton–Dinkelbach method for computing $y^{\text{high}}_u$. It is similar to the general method described in Section 3, except that our oracle here, Grapevine, solves ($\Phi_\delta$) instead of the actual subproblem ($\mathcal{H}_\delta$). At the beginning of iteration $i \in \mathbb{N}$, we have an *iterate-loop* pair $(\delta^{(i)}, P^{(i)})$, where $P^{(i)} \in \widehat{\mathcal{P}}^n_u$ is an optimal solution to the subproblem ($\mathcal{H}_\delta$) for $\delta = \delta^{(i)}$. If $h(\delta^{(i)}) = 0$, then we are done as the loop $P^{(i)}$ induces the upper bound $y^{\text{high}}_u$. Otherwise, we set $\delta^{(i+1)} := c(P^{(i)})/(1-\gamma(P^{(i)}))$ and use Grapevine to solve the subproblem ($\Phi_\delta$) for $\delta = \delta^{(i+1)}$. Let $P^{(i+1)} \in \mathcal{P}^n_u$ be the loop returned by Grapevine.

We now demonstrate how to handle the discrepancy between $h$ and $\phi$. Observe that

$$-c(P^{(i+1)}) + \delta^{(i+1)}(1 - \gamma(P^{(i+1)})) = \phi(\delta^{(i+1)}) \geq h(\delta^{(i+1)}) \geq -c(P^{(i)}) + \delta^{(i+1)}(1 - \gamma(P^{(i)})) = 0.$$

If $\gamma(P^{(i+1)}) < 1$, then $h(\delta^{(i+1)}) = \phi(\delta^{(i+1)})$ and a new iteration begins. Otherwise, the Newton–Dinkelbach method terminates. If $\phi(\delta^{(i+1)}) = 0$, then the previous loop $P^{(i)}$ is an optimal solution to our best upper bound problem because we have equality throughout. On the other hand, if $\phi(\delta^{(i+1)}) > 0$, then this constitutes an infeasibility cerficate for our instance. Indeed, if $\gamma(P^{(i+1)}) = 1$, then $c(P^{(i+1)}) < 0$, so $P^{(i+1)}$ is a negative unit-gain loop. Otherwise, we have $\gamma(P^{(i+1)}) > 1$, and $P^{(i+1)} \cup P^{(i)}$ forms a negative biloop because

$$\frac{c(P^{(i)})}{1-\gamma(P^{(i)})} = \delta^{(i+1)} < \frac{-c(P^{(i+1)})}{\gamma(P^{(i+1)})-1}.$$

Unlike in Section 3, we do not initialize $\delta$ arbitrarily. Instead, we pick $\delta$ a priori such that $y_u^{\text{high}} \leq \delta$. In particular, if $\widehat{\mathcal{P}}_u^n \neq \emptyset$, then $\delta$ can be chosen to be $c(P)/(1-\gamma(P))$ for any $P \in \widehat{\mathcal{P}}_u^n$. On the other hand, if $\widehat{\mathcal{P}}_u^n = \emptyset$, then $y_u^{\text{high}} = \infty$. Hence, initializing $\delta$ reduces to detecting a loop in $\widehat{\mathcal{P}}_u^n$. This can be done efficiently using the multiplicative version of a negative cycle detection subroutine like Bellman–Ford, by treating the gain factors as arc costs.

## 5.1 Strongly Polynomial Analysis of the Newton–Dinkelbach Method

Before presenting the label-correcting algorithm for M2VPI systems, we first analyze the Newton-Dinkelbach method (without look-ahead) as described in the previous subsection. The ideas we develop here will be useful later on when we analyze the label-correcting algorithm, which is based on the look-ahead Newton-Dinkelbach method. First, let us bound the number of good iterations encountered. Recall from Section 3 that an iteration $i \in \mathbb{N}$ is good if $1-\gamma(P^{(i+1)}) \leq \alpha(1-\gamma(P^{(i)}))$ for some constant $\alpha \in (1/2, 1)$ and bad otherwise.

**Lemma 5.1.** *The number of good iterations is at most $O(m \log n)$.*

*Proof.* Let $\mathcal{P}$ be a sequence of loops at $u$ which are produced by the Newton–Dinkelbach method. Let $\mathcal{P}^*$ be the subsequence of $\mathcal{P}$ formed by the loops in good iterations. We construct the sequence $\mathcal{P}^{**} := (P^{(1)}, P^{(2)}, \ldots, P^{(k)})$ from $P^*$ as follows. The loop $P^{(1)}$ is the first loop in $\mathcal{P}^*$. For every $i \geq 1$, the loop $P^{(i+1)}$ is the earliest loop after $P^{(i)}$ in $\mathcal{P}^*$ such that

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

Clearly, $\mathcal{P}^{**}$ is a subsequence of $\mathcal{P}^*$. Moreover, there are at most $\lceil -2/\log\alpha \rceil$ loops between $P^{(i)}$ and $P^{(i+1)}$ in $\mathcal{P}^*$ for all $i < k$. Thus, it suffices to prove that $k = O(m \log n)$. We claim that $\gamma(P^{(i+1)}) \geq \sqrt{\gamma(P^{(i)})}$ for all $i < k$. To prove this, consider the function $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ defined by $\rho(x) := (\sqrt{x}-1)^2/2$. Expanding out the square gives us

$$0 \leq \rho(x) = \frac{x - 2\sqrt{x}+1}{2} = (1-\sqrt{x}) - \frac{1}{2}(1-x),$$

which implies that $(1-x)/2 \leq 1-\sqrt{x}$ for all $x \in \mathbb{R}_+$. Applying this result to the first expression, we obtain

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

which proves the claim. Next, enumerate the arcs of each walk 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 as $P^{(i)} \in \widehat{\mathcal{P}}_u^n$ for all $i \in [k]$. Let $c \in \mathbb{R}_+^m$ be the vector defined by $c_e = |\log \gamma_e|$ for all $e \in E$. In addition, for every $i \in [k]$, define the vector

$x^{(i)} \in \{-1, 0, 1\}^m$ as

$$x_e^{(i)} = \begin{cases} 1 & \text{if } e \in E(P^{(i)}) \text{ and } \gamma_e < 1, \\ -1 & \text{if } e \in E(P^{(i)}) \text{ and } \gamma_e \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Then, we obtain

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

for all $i < k$. Since $\left|\operatorname{supp}(x^{(i)})\right| \leq n$ for all $i \in [k]$, we conclude that $k = O(m \log n)$ by Lemma 2.5. $\square$

It is left to bound the number of bad iterations encountered by the Newton–Dinkelbach method. We approach this by arguing that in a strongly polynomial number of iterations, an arc will no longer appear in future loops produced by the Newton–Dinkelbach method. First, we need to define a new notion of *negative cycles.*

**Definition 5.2.** Given node labels $y \in \bar{\mathbb{R}}^n$, we say that a cycle $C$ is *negative with respect to* $y_u$ if there exists a path $P$ from a node $v \in V(C)$ to node $u$ such that

$$c(CP) + \gamma(CP) y_u < c(P) + \gamma(P) y_u \leq y_v.$$

A cycle is said to be *negative with respect to* $y$ if it is negative with respect to $y_u$ for some node $u$.

In the definition above, we slightly abused notation by treating the cycle $C$ as a $u$-$u$ walk. Note that a trivial cycle can never be negative. However, the path $P$ can be a singleton $(u)$, in which case the inequality simplifies to $c(C) + \gamma(C) y_u < y_u$. Negative cycles are closely related to the behaviour of label-correcting algorithms. In particular, given input node labels $y \in \bar{\mathbb{R}}^n$, if there is no negative cycle with respect to $y$, then a generic label-correcting algorithm which performs arc updates will terminate with feasible node labels. This is akin to the termination criteria of the Bellman-Ford algorithm. In fact, our definition generalizes the notion of negative cycles in the shortest path setting. If a unit-gain cycle $C$ is negative with respect to $y_v$, then

$$c(P) + \gamma(P) y_v > c(CP) + \gamma(CP) y_v = c(C) + \gamma(C)(c(P) + \gamma(P) y_v) = c(C) + c(P) + \gamma(P) y_v,$$

which implies that $c(C) < 0$. Conversely, let $C$ be a unit-gain cycle with $c(C) < 0$. Then, $C$ is negative with respect to node labels $y \in \bar{\mathbb{R}}^n$ if and only if there exists a node $u$ reachable from $C$ such that $y_u < \infty$.

**Lemma 5.3.** *Let $y \in \bar{\mathbb{R}}^n$ be input node labels to* Grapevine. *If there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y$, then* Grapevine *returns a shortest path from $u$ in $G \setminus \delta^-(u)$ or a shortest cycle at $u$ in $G$ with respect to $y$.*

*Proof.* Let $z \in \bar{\mathbb{R}}^n$ be the node labels and $P$ be the walk returned by Grapevine. Recall that $y_u = z_u$ as Grapevine does not change the label at $u$. Since there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y$, $P$ is either a $u$-$w$ path for some node $w$, or a cycle at $u$. Note that $y_w = z_w$ in the former. We also have $z_v \leq c_{vw} + \gamma_{vw} z_w$ for all $vw \in E \setminus \delta^+(u)$, with equality on $E(P) \setminus \delta^+(u)$. Thus, $P$ is a shortest $u$-$w$ path in $G \setminus \delta^-(u)$ or a shortest cycle at $u$ in $G$ with respect to $y$. $\square$

The next property is crucial in our arc elimination argument.

**Definition 5.4.** Let $\mathcal{P} = (P^{(1)}, P^{(2)}, \dots, P^{(k)})$ be a sequence which consists of paths from $u$ or cycles at $u$. 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

$$\gamma(P^{(i)}_{uv}) \leq \gamma(P^{(j)}_{uv}).$$

**Lemma 5.5.** *Let $\mathcal{Y}$ be a sequence of pointwise nonincreasing input node labels to* Grapevine*, and $\mathcal{P}$ be the corresponding sequence of returned walks from $u$. If there is no negative cycle in $G \setminus \delta^+(u)$ with respect to every $y \in \mathcal{Y}$, then $\mathcal{P}$ satisfies subpath monotonicity at $u$.*

*Proof.* Denote $\mathcal{Y} = (y^{(1)}, y^{(2)}, \dots, y^{(k)})$ as the input node labels. Let $\mathcal{Z} = (z^{(1)}, z^{(2)}, \dots, z^{(k)})$ and $\mathcal{P} = (P^{(1)}, P^{(2)}, \dots, P^{(k)})$ be the corresponding sequence of node labels and walks returned by Grapevine respectively. Note that $y^{(i)}_u = z^{(i)}_u$ for all $i \in [k]$. For every $i \in [k]$, since there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y^{(i)}$, $P^{(i)}$ is a shortest path from $u$ in $G \setminus \delta^-(u)$ or a shortest cycle at $u$ in $G$ with respect to $y^{(i)}$ by Lemma 5.3. Now, pick $P^{(i)}$ and $P^{(j)}$ such that $i < j$ and they share a node $v \neq u$. We know that $z^{(j)} \leq z^{(i)}$ because $\mathcal{Y}$ is pointwise nonincreasing. Moreover, the subpaths $P^{(i)}_{uv}$ and $P^{(j)}_{uv}$ are shortest $u$-$v$ paths in $G \setminus \delta^-(u)$ with respect to $z^{(i)}$ and $z^{(j)}$ respectively. If $z^{(i)}_v = z^{(j)}_v$, then $P^{(i)}_{uv} = P^{(j)}_{uv}$ due to the consistency of Grapevine's tie-breaking. So, we may assume that $z^{(i)}_v > z^{(j)}_v$. Define the function $\psi : [z^{(j)}_v, z^{(i)}_v] \to \mathbb{R}$ as

$$\psi(x) := \inf \left\{ c(P) + \gamma(P)x : P \text{ is a } u\text{-}v \text{ walk in } G \setminus \delta^-(u) \right\}.$$

Since $\psi(z^{(i)}_v) = c(P^{(i)}_{uv}) + \gamma(P^{(i)}_{uv}) z^{(i)}_v$ and $\psi(z^{(j)}_v) = c(P^{(j)}_{uv}) + \gamma(P^{(j)}_{uv}) z^{(j)}_v$ are finite, the function $\psi$ is well-defined. Then, subpath monotonicity follows from the concavity of $\psi$. $\square$

**The gauge symmetry of cost modification.** In the analysis of bad iterations, the invariance developed in Section 3 plays a crucial role. Let $\pi \in \mathbb{R}^n$ be node potentials. We will modify the instance such that the cost of a loop $P$ at $u$ is given by $c(P) - \pi_u(1 - \gamma(P))$. Define the *modified cost* $c^\pi \in \mathbb{R}^m$ as follows:

$$c^\pi_{ij} := c_{ij} + \gamma_{ij}\pi_j - \pi_i \qquad \forall ij \in E.$$

For any $u$-$v$ walk $P$ in $G$ with $V(P) := (v_1, v_2, \dots, v_k)$, its modified cost is given by

$$\begin{aligned}
c^\pi(P) &= \sum_{i=1}^{k-1} \left( \prod_{j=1}^{i-1} \gamma_{v_j v_{j+1}} \right) c^\pi_{v_i v_{i+1}} \\
&= \sum_{i=1}^{k-1} \left( \prod_{j=1}^{i-1} \gamma_{v_j v_{j+1}} \right) \left( c_{v_i v_{i+1}} + \gamma_{v_i v_{i+1}} \pi_{v_{i+1}} - \pi_{v_i} \right) \\
&= \sum_{i=1}^{k-1} \left( \prod_{j=1}^{i-1} \gamma_{v_j v_{j+1}} \right) c_{v_i v_{i+1}} + \prod_{i=1}^{k-1} \gamma_{v_i v_{i+1}} \pi_{v_k} - \pi_{v_1} \\
&= c(P) + \gamma(P)\pi_v - \pi_u.
\end{aligned}$$

In particular, if $P$ is loop at $u$, the above expression becomes

$$c^\pi(P) = c(P) - \pi_u(1 - \gamma(P)), \tag{1}$$

as desired. Thus, we can invoke the invariance promised by Lemma 3.5.

It is worth pointing out that Lemma 3.4 also holds if we initialize our node labels differently when solving the subproblem ($\Phi_\delta$). Instead of using the initialization $y_u := \delta$ and $y_v := \infty$ for all $v \neq u$, we just need to make sure that $y_u = \delta$ and $y_v$ is sufficienctly high for all $v \neq u$ so that Grapevine returns a loop at $u$. For node potentials $\pi \in \mathbb{R}^n$, the modified $(y - \pi)$-cost of a $u$-$v$ walk $P$ is given by

$$\begin{aligned} c^\pi(P) + \gamma(P)(y_v - \pi_v) &= c(P) + \gamma(P)\pi_v - \pi_u + \gamma(P)(y_v - \pi_v) \\ &= c(P) + \gamma(P)y_v - \pi_u, \end{aligned}$$

which is equal to the $y$-cost of $P$ translated by a constant. Hence, by our choice of $y$, Grapevine also solves the subproblem ($\Phi_\delta$) on $(G, c^\pi, \gamma)$ when initialized with $y - \pi$. Furthermore, if there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y$, then the node labels returned by Grapevine (except for the label at $u$) can be used as input node labels to Grapevine in the next iteration. This gives a fixed way of setting the labels at $v \neq u$ after the first iteration, assuming that the absence of negative cycles is maintained. In light of these observations, we may call a sequence of loops *eligible for* $(G, c, \gamma, y)$, instead of just for $(G, c, \gamma, y_u = \delta)$.

The next lemma bounds the number of bad iterations under the assumption that there is no negative cycle in $G \setminus \delta^+(u)$ with respect to the input node labels to Grapevine.

**Lemma 5.6.** *Let $\mathcal{P}$ be a sequence of loops at $u$ produced by the Newton–Dinkelbach method, and $\mathcal{Y}$ be the corresponding sequence of input node labels to* Grapevine. *If there is no negative cycle in $G \setminus \delta^+(u)$ with respect to every $y \in \mathcal{Y}$, then the number of bad iterations is at most $O(m \log n)$.*

*Proof.* Let $(G, c, \gamma)$ be an M2VPI system. Denote $\mathcal{Y} = (y^{(1)}, y^{(2)}, \ldots, y^{(k)})$ as the input node labels to Grapevine. Let $\mathcal{Z} = (z^{(1)}, z^{(2)}, \ldots, z^{(k)})$ and $\mathcal{P} = (P^{(1)}, P^{(2)}, \ldots, P^{(k)})$ be the corresponding sequence of node labels and loops returned by Grapevine. Recall that $y^{(i)}_u = z^{(i)}_u$ for all $i \in [k]$ as Grapevine does not change the label at $u$. Without loss of generality, we may assume that $y^{(i)}$ is finite for all $i > 1$. Pick an iteration $j \in [k]$ such that more than $-\log(2n)/\log(1-\alpha)$ bad iterations have elapsed. Consider the modified cost $c' \in \mathbb{R}^m$ defined by $c'_{vw} := c_{vw} + \gamma_{vw} z^{(j)}_w - z^{(j)}_v$ for all $vw \in E$.

By Lemma 3.4, $\mathcal{P}$ is an eligible sequence for $(G, c', \gamma, y^{(1)} - z^{(j)})$. Thus, we may assume that the Newton–Dinkelbach method is run on the instance $(G, c', \gamma)$ and initialized with $y^{(1)} - z^{(j)}$ instead. It is easy to see that the corresponding input node labels to Grapevine in iteration $i$ are given by $y^{(i)} - z^{(j)}$.

For every $i \in [k]$, there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y^{(i)}$. So, $P^{(i)}$ is a shortest cycle at $u$ with respect to $y^{(i)}$ by Lemma 5.3. Since $\mathcal{Y}$ is pointwise nonincreasing, $\mathcal{P}$ also satisfies subpath monotonicity at $u$ by Lemma 5.5. Define the vector $r \in \mathbb{R}^m_+$ as

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

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

**Claim 5.7.** *We have $h^{(j)} < \|r \circ c'\|_\infty$.*

*Proof.* Recall that $\circ$ means pointwise multiplication. For every $i \in [k]$, we have

$$h^{(i)} = -c(P^{(i)}) + y^{(i)}_u(1 - \gamma(P^{(i)})) = -c'(P^{(i)}) + (y^{(i)}_u - z^{(j)}_u)(1 - \gamma(P^{(i)})).$$

By applying the definition of $y_u^{(i)}$, we can upper bound this quantity by

$$h^{(i)} = -c'(P^{(i)}) + \frac{1-\gamma(P^{(i)})}{1-\gamma(P^{(i-1)})}c'(P^{(i-1)}) \leq \left|c'(P^{(i)})\right| + \left|c'(P^{(i-1)})\right| \leq 2n \left\|r \circ c'\right\|_\infty .$$

Lemma 3.1 tells us that $h^{(i)}$ is nonnegative and monotonically decreasing. Moreover, it decreases geometrically by a factor of $1-\alpha$ during bad iterations from Lemma 3.2. Hence, by our choice of $j$, we obtain

$$h^{(j)} < (1-\alpha)^{-\log(2n)/\log(1-\alpha)} \cdot 2n \left\|r \circ c'\right\|_\infty = \left\|r \circ c'\right\|_\infty .$$

□

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

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

We show that $d \geq 0$. Pick an arc $vw \in E$. If $v \neq u$, then $d_{vw} \geq 0$ because there is no negative cycle in $G \setminus \delta^+(u)$ with respect to $y^{(j)}$. Otherwise, if $v = u$, let $Q$ denote the $w$-$u$ walk which gave $z_w^{(j)}$ its value. Then, we obtain

$$\begin{aligned} d_{uw} &= c_{uw} + \gamma_{uw} z_w^{(j)} - z_u^{(j)} - c(P^{(j)}) + y_u^{(j)}(1-\gamma(P^{(j)})) \\ &= c_{uw} + \gamma_{uw}(c(Q) + \gamma(Q) y_u^{(j)}) - \left(c(P^{(j)}) + \gamma(P^{(j)}) y_u^{(j)}\right) \geq 0 \end{aligned}$$

as $P^{(j)}$ is a shortest cycle at $u$ with respect to $y^{(j)}$. The following claim will be handy later on.

**Claim 5.8.** *We have $\left\|r \circ d\right\|_\infty \geq \left\|r \circ c'\right\|_\infty$.*

*Proof.* Let $f = \arg\max_{e\in E} |r_e c'_e|$. The claim is trivial unless $f \in \cup_{i=1}^k E(P^{(i)})$ and the tail of $f$ is $u$. Since $h^{(j)} \geq 0$ and $d_f = c'_f + h^{(j)}$, it suffices to show that $c'_f \geq 0$. For the purpose of contradiction, suppose that $c'_f < 0$. Since $d_f \geq 0$, this implies that $|c'_f| \leq h^{(j)} < \left\|r \circ c'\right\|_\infty$ using Claim 5.7. By the definition of $r$, $r_f = 1$ because $f$ is the first arc of any cycle in $\mathcal{P}$ which uses it. However, this implies that

$$\left|c'_f\right| = \left|r_f c'_f\right| = \left\|r \circ c'\right\|_\infty ,$$

which is a contradiction. □

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

$$0 \geq y_u^{(i+1)} - y_u^{(j)} = \frac{c(P^{(i)})}{1-\gamma(P^{(i)})} - y_u^{(j)} = \frac{c'(P^{(i)})}{1-\gamma(P^{(i)})} = \frac{d(P^{(i)}) - h^{(j)}}{1-\gamma(P^{(i)})}$$

where we used (1) in the second equality. This implies that $d(P^{(i)}) \leq h^{(j)} < \left\|r \circ c'\right\|_\infty$. However, it contradicts

$$d(P^{(i)}) \geq r_f d_f = \left\|r \circ d\right\|_\infty \geq \left\|r \circ c'\right\|_\infty ,$$

where the first inequality is due to our choice of $i$ and the nonnegativity of $d$, while the second inequality is due to Claim 5.8. Repeating the argument above for $m$ times yields the desired bound on the number of bad iterations. □

We are now ready give a strongly polynomial bound on the total number of iterations performed by the Newton–Dinkelbach method.

**Theorem 5.9.** *The Newton–Dinkelbach method computes* $y_u^{\mathrm{high}}$ *or returns an infeasibility certificate in* $O(mn\log n)$ *iterations.*

*Proof.* From Lemma 5.1, we know that the number of good iterations is $O(m\log n)$. However, we cannot apply Lemma 5.6 directly, as there could be negative cycles in $G \setminus \delta^+(u)$ with respect to the input node labels to Grapevine. To address this issue, we define a new 'layered' digraph $H = (W, F)$ as follows. The node set $W$ consists of $u$ and $n-1$ columns of nodes, where each column contains a copy of $V \setminus u$. There is an arc from $u$ to a node $v$ in column 1 if and only if $uv \in E$. Similarly, there is an arc from a node $v$ in column $n-1$ to $u$ if and only if $vu \in E$. Each node $v$ in any column $i$ is adjacent to node $w$ in column $i+1$ if and only if $vw \in E$. The costs and gain factors of these arc are the costs and gain factors of the associated arc in $G$. In addition, each node $v$ in any column $i$ is adjacent to node $v$ in column $i+1$ with cost 0 and gain factor 1.

Every loop in $\mathcal{P}_u(H)$ has length exactly $n$. Moreover, there is a natural surjective mapping from $\mathcal{P}_u(H)$ to $\mathcal{P}_u^n(G)$ which preserves costs and gain factors. Thus, we may assume that the sequence of loops produced by the Newton–Dinkelbach method on $G$ and $H$ are the same. This means that it suffices to bound the number of bad iteration in $H$. Since the subgraph $H \setminus \delta^+(u)$ is acyclic, there are $O(mn\log n)$ bad iterations according to Lemma 5.6. ☐

**From the $y^{\mathrm{high}}$ values to feasible solutions.** We now show that based on the computation of $y^{\mathrm{high}}$, a feasible solution to an M2VPI system can be computed easily. We summarize the classical arguments already used by Aspvall and Shiloach [2].

Let us first assume that the M2VPI system is bounded from above. For every node $u$, define

$$y_u^{\max} := \min\left\{c(P) + \gamma(P)y_v^{\mathrm{high}} : P \text{ is a } u\text{-}v \text{ walk of length at most } n \text{ in } G\right\}.$$

These can be computed efficiently using a generic label-correcting algorithm. They are also finite due to our boundedness assumption. Aspvall and Shiloach showed that $y^{\max}$ is the unique pointwise maximal solution to the system if and only if the system is feasible.

If the M2VPI system is unbounded from above, then we need to do slightly more work. This is because there exists a vertex $u \in V$ for which $y_u^{\max} = \infty$. Let $\overleftarrow{G} := (V, \overleftarrow{E})$ be the reversed graph of $G$. For every node $u$, define

$$y_u^{\min} := \max\left\{c(P) + \gamma(P)y_v^{\mathrm{low}} : P \text{ is a } u\text{-}v \text{ walk of length at most } n \text{ in } \overleftarrow{G}\right\}.$$

Aspvall and Shiloach proved that if the M2VPI system is feasible, the interval $[y_u^{\min}, y_u^{\max}]$ is the projection of the feasible region onto the coordinate $y_u$. If the system is infeasible, then there exists a negative unit-gain cycle at $u$ or $y_u^{\max} < y_u^{\min}$ for some $u \in V$. To obtain a feasible solution, we fix a coordinate $y_u \in [y_u^{\min}, y_u^{\max}]$, update $y^{\min}$ and $y^{\max}$ using the generic label-correcting algorithm, and repeat.

It is left to show how to compute $y^{\mathrm{low}}$ using the Newton-Dinkelbach method. We reduce this problem to computing $y^{\mathrm{high}}$ on a different but related instance. For every arc $e \in E$, define the cost and gain factor of its reversed arc $\overleftarrow{e}$ in $\overleftarrow{G}$ as $-c_e/\gamma_e$ and $1/\gamma_e$ respectively. Then, for a $u$-$v$ walk $P$

in $G$ with $E(P) = (e_1, e_2, \dots, e_k)$, the cost and gain factor of its reversed walk $\overleftarrow{P}$ in $\overleftarrow{G}$ are

$$c(\overleftarrow{P}) = \sum_{i=1}^{k} \left( \prod_{j=i+1}^{k} \frac{1}{\gamma_{e_j}} \right) \left( \frac{-c_{e_i}}{\gamma_{e_i}} \right) = \sum_{i=1}^{k} \left( \frac{1}{\gamma(P)} \prod_{j=1}^{i-1} \gamma_{e_j} \right) (-c_{e_i}) = \frac{-c(P)}{\gamma(P)}$$

$$\gamma(\overleftarrow{P}) = \prod_{i=1}^{k} \frac{1}{\gamma_{e_i}} = \frac{1}{\gamma(P)}$$

respectively. This yields the following relation

$$y_u^{\text{low}} = \max_{P \in \widecheck{\mathcal{P}}_u^n(G)} \left\{ \frac{-c(P)}{\gamma(P) - 1} \right\} = \max_{P \in \widecheck{\mathcal{P}}_u^n(G)} \left\{ \frac{-c(P)}{\gamma(P)} \cdot \frac{\gamma(P)}{\gamma(P) - 1} \right\} = \max_{P \in \widehat{\mathcal{P}}_u^n(\overleftarrow{G})} \left\{ \frac{c(\overleftarrow{P})}{1 - \gamma(\overleftarrow{P})} \right\}.$$

If $y_u^{\text{high}}$ or $y_u^{\text{low}}$ is finite, then we know that there is no negative unit-gain loop at $u$. This is because $\phi(y_u^{\text{high}})$ or $\phi(y_u^{\text{low}})$ is zero from the termination criteria of the Newton–Dinkelbach method. On the other hand, if $y_u^{\text{high}} = \infty$ and $y_u^{\text{low}} = -\infty$, then $\widehat{\mathcal{P}}_u^n = \widecheck{\mathcal{P}}_u^n = \emptyset$. This implies that $\phi = \xi$ for some $\xi \in \mathbb{R}$ because $\mathcal{P}_u^n \neq \emptyset$ as it contains the trivial cycle at $u$. In this case, we need to check for the existence of a negative unit-gain loop at $u$, i.e. whether $\xi > 0$. This can be done by running GRAPEVINE initialized with node labels $y \in \bar{\mathbb{R}}^n$, where $y_u \in \mathbb{R}$ is chosen arbitrarily and $y_v := \infty$ for all $v \neq u$. If GRAPEVINE returns a nontrivial loop $P \in \mathcal{P}_u^n$, then $\gamma(P) = 1$ and $c(P) + \gamma(P) y_u < y_u$. So $P$ is a negative unit-gain loop, which certifies the infeasibility of our system by Theorem 2.3. Otherwise, we have $\xi \leq 0$, which implies that there is no negative unit-gain loop at $u$.

Since the total runtime is dominated by the computation of $y^{\text{high}}$ and $y^{\text{low}}$, we have the following result.

**Corollary 5.10.** *The Newton–Dinkelbach method solves the feasibility of 2VPI linear systems in $O(mn^2 \log n)$ iterations.*

In the next subsection, we use the accelerated Newton–Dinkelbach method coupled with a better implementation to bring down the iteration complexity to $O(mn)$.

## 5.2 A Label-Correcting Algorithm

In the previous subsection, we proved that the Newton–Dinkelbach method for computing $y_u^{\text{high}}$ converges in a strongly polynomial number of iterations. However, the analysis of bad iterations was carried out on an auxiliary graph, which resulted in an upper bound that is roughly $n$ times the number of good iterations. In this subsection, we present a better implementation which facilitates the analysis of bad iterations on the input graph directly. Moreover, we replace the standard Newton–Dinkelbach method with the look-ahead version developed in Section 3. This culminates in a label-correcting algorithm for M2VPI systems with $O(mn)$ iteration complexity.

Instead of computing $y_u^{\text{high}}$ separately for each node $u$, we now compute the vector $y^{\max}$ directly. Recall that if $y^{\max}$ is finite, then it is the unique pointwise maximal solution if and only if the system is feasible. Throughout the algorithm, we maintain node labels $y \in \bar{\mathbb{R}}^n$, which will remain valid upper bounds for each variable. For every node $u \in V$, its label is initialized to $y_u := c(P)/(1-\gamma(P))$ for some $P \in \widehat{\mathcal{P}}_u^n(G)$, or $y_u := \infty$ if $\widehat{\mathcal{P}}_u^n(G) = \emptyset$. As mentioned at the beginning of this section, finding a loop in $\widehat{\mathcal{P}}_u^n(G)$ can be done using the multiplicative Bellman-Ford algorithm. We also maintain a subgraph of $G$, which initially is $G_0 := (V, \emptyset)$. The algorithm is divided into $n$ *phases*. In phase $k \in [n]$, we start by picking a node $u$ and adding all its incident arcs in $G$ to $G_{k-1}$, resulting in a larger subgraph $G_k$. Then, we update the label at $u$ by propagating the labels at other nodes to it via $\delta^+(u)$. In the remaining part of the phase, we apply Algorithm 1 on $G_k$ to compute the best upper bound on $u$ induced by the loops in $\mathcal{P}^n_u(G_k)$.

**Algorithm 3:** Label-correcting algorithm for M2VPI systems

```
input  : An M2VPI system $(G, c, \gamma)$.
output: The vector $y^{\max}$ or the string INFEASIBLE.
1  Initialize the graph $G_0 \leftarrow (V, \emptyset)$
2  Initialize the node set $S \leftarrow \emptyset$
3  foreach $u \in V$ do                                   /* Initialize node labels */
4      if $\widehat{\mathcal{P}}^n_u(G) = \emptyset$ then
5          $y_u \leftarrow \infty$
6      else
7          $y_u \leftarrow c(P)/(1-\gamma(P))$ for any $P \in \widehat{\mathcal{P}}^n_u(G)$
8  for $k = 1$ to $n$ do
9      $S \leftarrow S \cup \{u\}$ for some $u \in V \setminus S$
10     $G_k \leftarrow G_{k-1} \cup \delta(u)$
11     $y_u \leftarrow \min\left\{y_u, \min_{uv\in\delta^+(u)}\{c_{uv} + \gamma_{uv} y_u\}\right\}$
12     $(y, P) \leftarrow$ ModifiedGrapevine($G_k$,$y$,$u$)
13     if $V(P) = \emptyset$ then
14         return INFEASIBLE
15     while $E(P) \neq \emptyset$ do
16         if $\gamma(P) \geq 1$ then
17             return INFEASIBLE
18         $\bar{y} \leftarrow y$, $\bar{P} \leftarrow P$                 /* Store current labels-loop pair */
19         $y_u \leftarrow c(P)/(1-\gamma(P))$
20         $(y, P) \leftarrow$ ModifiedGrapevine($G_k$,$y$,$u$)
21         if $V(P) = \emptyset$ then
22             return INFEASIBLE
23         if $1-\gamma(P) > \alpha(1-\gamma(\bar{P}))$ then                /* Bad iteration */
24             $y' \leftarrow \bar{y}$
25             $y'_u \leftarrow y_u - \frac{1-\alpha}{2\alpha-1}(\bar{y}_u - y_u)$          /* Guess next iterate */
26             $(y', P') \leftarrow$ ModifiedGrapevine($G_k$,$y'$,$u$)
27             if $V(P') \neq \emptyset$ and $\gamma(P') < 1$ then
28                 $y \leftarrow y'$, $P \leftarrow P'$
29 return $y$
```

An iteration of Algorithm 3 refers to a repetition of the `while` loop. It is based on the same logic behind Algorithm 1. Every time we encounter a bad iteration, we perform the look-ahead procedure (lines 23–28). Just like in the previous subsection, we also want to detect infeasibility of the system on the fly. To achieve this, we slightly modify the Grapevine subroutine as follows. Let $y \in \bar{\mathbb{R}}^n$ be the node labels and $P$ be the walk at the end of a Grapevine run. Before it terminates and returns $(y, P)$, we add one additional check. Namely, if there is a violated arc in $G \setminus \delta^+(u)$ with respect to $y$, then we overwrite $P$ with the empty walk, i.e. $V(P) = \emptyset$. We call this subroutine ModifiedGrapevine. Observe that if the returned walk $P$ is a singleton, i.e., $V(P) = \{u\}$, then $y$ is a feasible solution to the system. In Algorithm 3, if ModifiedGrapevine returns a loop $P$

such that $V(P) = \emptyset$ or $\gamma(P) \geq 1$, then we terminate and conclude infeasibility. The only exception is if this happens during look-ahead, in which case we simply abandon our guess and proceed to the next iteration immediately.

The following lemma illustrates the advantage of this implementation.

**Lemma 5.11.** *Given input node labels $y \in \bar{\mathbb{R}}^n$,* ModifiedGrapevine *returns the empty walk in phase $k$ if and only if there exists a negative cycle $C$ in $G_k \setminus \delta^+(u)$ with respect to $y$. Moreover, $C$ is flow-generating and negative with respect to $y_u$.*

*Proof.* Let $y \in \bar{\mathbb{R}}^n$ be the input node labels to ModifiedGrapevine at some point during phase $k$. Let $u$ be the node added at the beginning of phase $k$. First, we prove the following claim.

**Claim 5.12.** *For every edge $vw \in E(G_k) \setminus \delta(u)$, we have $y_v \leq c_{vw} + \gamma_{vw} y_w$.*

*Proof.* The claim is clearly true for $k = 1$. It is also true if this is the first call to ModifiedGrapevine in phase $k$. This is because $y$ is a feasible solution to the subsystem $(G_{k-1} \setminus \delta(u), c, \gamma)$. So, we may assume that this call to ModifiedGrapevine occured during an iteration in phase $k$. Let $(\bar{y}, \bar{P})$ be the stored label-loop pair at the start of this iteration. We know that $V(\bar{P}) \neq \emptyset$, as otherwise Algorithm 3 would have terminated before this iteration. So there are no violated arcs in $G_k \setminus \delta^+(u)$ with respect to $\bar{y}$. Since $\bar{y}_v = y_v$ for all $v \neq u$, it follows that there are no violated arcs in $G_k \setminus \delta(u)$ with respect to $y$. □

($\Rightarrow$) Suppose that ModifiedGrapevine returns the empty walk when it is run on $G_k$ with input node labels $y$. Let $z \in \bar{\mathbb{R}}^n$ denote the returned node labels. Then, there exists a violated arc $vw \in E(G_k) \setminus \delta^+(u)$ with respect to $z$. Let $R$ be the walk in $G_k \setminus \delta^+(u)$ which gave $z_w$ its value. Then, $R$ is not simple because it has length $n$. From the discussion in the first paragraph, we also know that $R$ is a $w$-$u$ walk. Decompose the walk into $R = QCP$, where $Q$ is a $w$-$v$ walk, $C$ is a nontrivial cycle at $v$, and $P$ is a $v$-$u$ path. Then, $C$ is a negative cycle with respect to $y_u$ in $G_k \setminus \delta^+(u)$ because

$$c(CQ) + \gamma(CQ)y_u < c(Q) + \gamma(Q)y_u \leq y_v.$$

($\Leftarrow$) Suppose that there exists a negative cycle $C$ with respect to $y$ in $G_k \setminus \delta^+(u)$. From the discussion in the first paragraph, $C$ is negative with respect to $y_u$. Let $P$ be a $v$-$u$ path in $G_k \setminus \delta^+(u)$ such that $v \in V(C)$ and

$$c(CP) + \gamma(CP)y_u < c(P) + \gamma(P)y_u \leq y_v.$$

We also have $y_v \leq c(C) + \gamma(C)y_v$ because $u \notin V(C)$. We claim that $\gamma(C) > 1$. Indeed, if $\gamma(C) = 1$, then we obtain $0 \leq c(C) < 0$ from the previous two inequalities. On the other hand, if $\gamma(C) < 1$, then we get the following contradiction

$$y_v \leq \frac{c(C)}{1 - \gamma(C)} < c(P) + \gamma(P)y_u \leq y_v.$$

Let us run ModifiedGrapevine on $G_k$ with input node labels $y$. Denote $z \in \bar{\mathbb{R}}^n$ as the returned node labels. Then, we have

$$z_v \leq c(P) + \gamma(P)y_u < \frac{-c(C)}{\gamma(C) - 1},$$

which gives us $z_v > c(C) + \gamma(C)z_v$. This implies that there exists an arc $st \in E(C)$ such that $z_s > c_{st} + \gamma_{st}z_t$. So, ModifiedGrapevine returns the empty walk. □

**Theorem 5.13.** *Algorithm 3 is correct.*

*Proof.* Let $(G, c, \gamma)$ be an M2VPI system. First, assume that Algorithm 3 returns a vector $y \in \bar{\mathbb{R}}^n$. It is easy to see that $y$ is an upper bound on every feasible solution of the system. Moreover, we have $y_v \leq c_{vw} + \gamma_{vw} y_w$ for all $vw \in E$. Now, decompose $G$ into strongly connected components. The node labels in each component are either all finite or infinite. If a component $C$ has infinite node labels, then there is no flow-absorbing loop in every component reachable from it. Thus, $y_v = y_v^{\max} = \infty$ for all $v \in C$. Now, let $S \subseteq V$ be the nodes with finite node label. Then, $y\big|_S \in \mathbb{R}^S$ is a pointwise maximal solution to the subsystem $(G[S], c, \gamma)$. It follows that $y_v = y_v^{\max}$ for all $v \in S$.

Next, assume that Algorithm 3 returns the string `INFEASIBLE` in phase $k$. Let $P$ be the loop which caused this outcome, and let $y \in \bar{\mathbb{R}}^n$ be the input node labels to ModifiedGrapevine which returned this loop. We know that this ModifiedGrapevine run did not occur during look-ahead. First, let us assume that $P = \emptyset$. By Lemma 5.11, there exists a negative cycle $C$ with respect to $y$ in $G_k \setminus \delta^+(u)$. Moreover, $C$ is flow-generating and negative with respect to $y_u$. Let $Q$ be a $v$-$u$ path in $G_k \setminus \delta^+(u)$ such that $v \in V(C)$ and

$$c(CQ) + \gamma(CQ) y_u < c(Q) + \gamma(Q) y_u \leq y_v.$$

Since $y_u$ is finite, let $D$ be the flow-absorbing loop from which $y_u$ derives its value, i.e. $y_u = c(R) + \gamma(R)(c(D)/(1 - \gamma(D)))$ for some walk $R$ from $u$ in $G_k$. Note that $D$ might not be a loop in $G_k$, i.e. when the label $c(D)/(1 - \gamma(D))$ was assigned during initialization of Algorithm 3. Then, $C \cup QR \cup D$ is a negative biloop in $G$ because

$$c(QR) + \gamma(QR) \frac{c(D)}{1 - \gamma(D)} = c(Q) + \gamma(Q) y_u < \frac{-c(C)}{\gamma(C) - 1}.$$

Thus, the system is infeasible by Theorem 2.3.

Finally, let us assume that $\gamma(P) \geq 1$. Observe that $P \neq (u)$, as otherwise phase $k$ would have ended. Hence, we have $c(P) + \gamma(P) y_u < y_u$. If $\gamma(P) = 1$, then $c(P) < 0$. Thus, $P$ is a negative unit-gain loop, which forms an infeasibility certificate to the system by Theorem 2.3. So, we may assume that $\gamma(P) > 1$. Since $y_u$ is finite, let $D'$ be the flow-absorbing loop from which $y_u$ derives its value, i.e. $y_u = c(R') + \gamma(R')(c(D')/(1 - \gamma(D')))$ for some walk $R'$ from $u$ in $G_k$. Just like in the previous paragraph, note that $D'$ might not be a cycle in $G_k$. Then, $P \cup R' \cup D'$ is a negative biloop in $G$ because

$$c(R') + \gamma(R') \frac{c(D')}{(1 - \gamma(D')} = y_u < \frac{-c(P)}{\gamma(P) - 1},$$

Therefore, the system is infeasible according to Theorem 2.3. □

**Theorem 5.14.** *Algorithm 3 terminates in $O(mn)$ iterations.*

*Proof.* Let $(G, c, \gamma)$ be an M2VPI system. Fix a phase $k \in [n]$, and let $u$ be the node added to $S$ in this phase. Let $\mathcal{P} := (P^{(1)}, P^{(2)}, \ldots, P^{(\ell)})$ be a sequence of loops at the start of every iteration in phase $k$, and $\mathcal{Y} := (y^{(1)}, y^{(2)}, \ldots, y^{(\ell)})$ be the corresponding sequence of input node labels to ModifiedGrapevine. Without loss of generality, we may assume that $y^{(i)}$ is finite for all $i > 1$. Observe that $V(P^{(i)}) \neq \emptyset$ for all $i \in [\ell]$. Consequently, by Lemma 5.11, there is no negative cycle in $G_k \setminus \delta^+(u)$ with respect to $y^{(i)}$ for all $i \in [\ell]$. Then, Lemma 5.3 tells us that $P^{(i)}$ is a cycle at $u$ for all $i \in [\ell]$.

First, let us assume that Algorithm 3 terminates with the string `INFEASIBLE` in phase $k$. By Lemma 5.1, there are $O(m \log n)$ good iterations in this phase. Since there is no negative cycle in

$G_k \setminus \delta^+(u)$ with respect to the node labels in $\mathcal{Y}$, we can apply Lemma 5.6 to bound the number of bad iterations by $O(m \log n)$. Thus, we have $\ell = O(m \log n)$.

Next, let us assume that Algorithm 3 completes phase $k$ without terminating. Then, $y^{(\ell)}$ is a feasible solution to the subsystem $(G_k, c, \gamma)$. Let $m_k := |E(G_k)|$. To finish the proof, it suffices to show that $\ell = O(m_k)$. Consider the modified cost $c^* \in \mathbb{R}^{m_k}$ defined by $c^*_{vw} := c_{vw} + \gamma_{vw} y^{(\ell)}_w - y^{(\ell)}_v$ for all $vw \in E(G_k)$. Observe that $c^* \geq 0$ due to the feasibility of $y^{(\ell)}$. By Lemma 3.4, $\mathcal{P}$ is an eligible sequence for $(G_k, c^*, \gamma, y^{(1)} - y^{(\ell)})$. Thus, we may assume that Algorithm 3 is run on the instance $(G, c^*, \gamma)$ instead and starts the first iteration of phase $k$ with node labels $y^{(1)} - y^{(\ell)}$. It is easy to see that for each $i \in [\ell]$, the corresponding node labels at the start of iteration $i$ are given by $y^{(i)} - y^{(\ell)}$. In particular, the last node labels are all zero. So, letting $\beta := \max\{\alpha, (1-\alpha)/\alpha\}$ and applying Lemma 3.9 yields $c^*(P^{(i+1)}) \leq \beta c^*(P^{(i-1)})$ for all iterations $1 < i < \ell$.

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

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

By Lemma 5.5, the sequence $\mathcal{P}$ satisfies subpath monotonicity at $u$ because there is no negative cycle in $G_k \setminus \delta^+(u)$ with respect to the node labels in $\mathcal{Y}$. Hence, $r_{vw}$ is equal to the gain factor of the $u$-$v$ subpath of the last cycle in $\mathcal{P}$ which contains $vw$. Let $0 \leq r_1 c^*_1 \leq r_2 c^*_2 \leq \cdots \leq r_{m_k} c^*_{m_k}$ be the elements of $r \circ c$ in nondecreasing order, and define $d_i := \sum_{j=1}^{i} r_j c^*_j$. 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 $\lceil -2/\log\beta \rceil$ cycles from $\mathcal{P}$.

Pick $j < m_k$. Among all the cycles 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

$$c^*(P^{(i+\lceil -2/\log\beta\rceil)}) \leq \frac{1}{2} c^*(P^{(i)}) \leq \frac{1}{2} d_{j+1} \leq d_j.$$

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

$$c^*(P^{(i+\lceil -2/\log\beta\rceil)}) \leq \frac{1}{2} c^*(P^{(i)}) \leq \frac{1}{2} d_{j+1} = d_{j+1} - \frac{1}{2} d_{j+1} = r_{j+1} c^*_{j+1} + d_j - \frac{1}{2} d_{j+1} < r_{j+1} c^*_{j+1}.$$

By subpath monotonicity, the cycles from $P^{(i+\lceil -2/\log\beta\rceil)}$ onwards do not contain an arc whose cost is at least $c^*_{j+1}$. Therefore, their costs are at most $d_j$ each. □

By applying Algorithm 3 on the reversed graph $\overleftarrow{G}$, one can also compute $y^{\min}$ in $O(mn)$ iterations. Thus, following the discussion at the end of the previous subsection, we obtain the following result.

**Corollary 5.15.** *Algorithm 3 solves the feasibility of 2VPI linear systems in $O(mn)$ iterations.*

Since the runtime of every iteration is dominated by the runtime of ModifiedGrapevine, we also have the following result.

**Corollary 5.16.** *Algorithm 3 solves the feasibility of 2VPI linear systems in $O(m^2n^2)$ time.*

## 6 Deterministic Markov Decision Processes

In this section, we replace ModifiedGrapevine with Dijkstra's algorithm [6] 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 [18]; we will supply the details. Recall that an instance can be described by a directed multigraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^m$ and *discount* factors $\gamma \in (0, 1]^m$. Since the discount factor of every cycle is at most 1, there are no bicycles in $G$. Consequently, by Theorem 2.3, 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.

In every phase of Algorithm 3, we use the following variant of Dijkstra's algorithm to recompute a shortest cycle at $u$ with respect to node labels $y \in \bar{\mathbb{R}}^n$ when $y_u$ decreases. It is slightly modified from the standard well-known algorithm in order to handle discount factors. Even though Algorithm 4 returns a shortest path tree to $u$, it can be easily adapted to our setting. Let $u$ be the node added at the beginning of phase $k$ in Algorithm 3. We split the node $u$ into two nodes $u$ and $u'$, where $u$ inherits the incoming arcs while $u'$ inherits the outgoing arcs. Now, let $(y, P)$ be the label-loop pair at the start of an iteration in phase $k$. Set the label at $u'$ as $y_{u'} := c(P) + \gamma(P)y_u$. Then, it is easy to verify that the reduced cost of every edge in this graph is nonnegative. Finally, the parameter $\varepsilon$ is the amount by which $y_u$ decreases before ModifiedGrapvine is called, e.g. $\varepsilon = y_u - c(P)/(1 - \gamma(P))$ when look-ahead is not performed.

**Algorithm 4:** Recompute shortest paths to $u$ with respect to $y$

**input :** A digraph $G = (V, E)$ with arc costs $c \in \mathbb{R}^E$ and discount factors $\gamma \in (0, 1]^E$, 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 $\varepsilon > 0$

**output:** An in-tree $T$ rooted at $u$ and node labels $z \in \mathbb{R}^V$ such that $z \leq y$, $z_u = y_u - \varepsilon$ and $c_{vw} + \gamma_{vw}z_w - z_v \geq 0$ for every $vw \in E$, with equality on every arc of $T$.

```
1  $y_u \leftarrow y_u - \varepsilon$
2  Define modified 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 $R \leftarrow \{u\}$ 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             $pred(v) \leftarrow w$
13             $R \leftarrow R \cup \{v\}$
14 Let $T$ be the in-tree defined by $pred()$
15 $z \leftarrow y + z$
16 return $T$ and $z$
```

An iteration of Algorithm 4 refers to a repetition of the `while` loop. In the pseudocode above, observe that $\bar{c}_{vw} \geq 0$ for all $vw \in E \setminus \delta^-(u)$. As we are only interested in the shortest $u'$-$u$ path, we can stop the algorithm as soon as $u'$ enters the set $S$.

**Lemma 6.1.** *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 $z\big|_S$ 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 $v_j = \mathit{pred}(v_{k+1})$ 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 $z_v \leq z_{v_{k+1}}$. □

An efficient implementation of Dijkstra's algorithm using Fibonacci heaps was given by Fredman and Tarjan [10]. It can also be applied to our setting, with the same running time of $O(m + n \log n)$. Thus, we obtain a faster running time for Algorithm 3 in this setting.

**Corollary 6.2.** *Algorithm 3 solves deterministic MDPs in $O(mn(m + n \log n))$ time.*

## 7 Concluding Remarks

We presented a new strongly polynomial algorithm for 2VPI systems, as well as an enhanced version of the Newton–Dinkelbach method for fractional combinatorial optimization.

Whereas a number of strongly polynomial algorithms have already been known for the 2VPI problem [3, 17, 20], all of them relied on some form of binary or parametric search. In contrast, we present a label-correcting algorithm that can be seen as a more direct extension of classical shortest path/negative cycle detection algorithms.

There are two possible extensions where this new approach may lead to progress. The first one is solving LPs of the form $\min c^\top x, Ax = b, x \geq 0$ with $A \in \mathcal{M}_2(n, m)$; recall that the 2VPI problem corresponds to finding a feasible dual solution. The optimization problem is equivalent to the minimum-cost generalized flow problem, see e.g. [15, 34]. We expect that combined with the ideas from generalized flow maximization [22, 32], our techniques may lead to further progress.

The second direction is undiscounted nondeterministic Markov Decision Processes, and more generally, solving systems of the form $A^\top y \leq c$, where $A$ is a pre-Leontief matrix. A matrix is *pre-Leontief* if every column contains at most one positive entry. Such systems (if bounded) have unique pointwise maximal solutions [4], giving hope for a label-correcting algorithm to succeed.

## Acknowledgements

We are grateful to Daniel Dadush, who suggested us to study the strong polynomiality of the Newton–Dinkelbach method for 2VPI systems. The third author would like to thank Neil Olver for several inspiring discussions on 2VPI systems, in particular, on gauge symmetries of the problem.

## References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, 1993. 2

[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. 2, 4, 13, 20

[3] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 23(6):1313–1347, 1994. 2, 27

[4] R. W. Cottle and A. F. Veinott Jr. Polyhedral sets having a least element. *Math. Program.*, 3(1):238–249, 1972. 27

[5] D. Dadush, S. Huiberts, B. Natura, and L. A. Végh. A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. In *Proceedings of the 52th Annual ACM SIGACT Symposium on Theory of Computing (to appear)*, 2020. 6

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 26

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

[8] 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. 2, 6

[9] 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. 3

[10] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. 27

[11] 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. 6

[12] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989. 5

[13] 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

[14] T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1–16, 2013. 6

[15] D. S. Hochbaum. Monotonizing linear programs with up to two nonzeroes per column. *Oper. Res. Lett.*, 32(1):49–58, 2004. 27

[16] 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. 2, 6

[17] 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. 2, 4, 27

[18] 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, 26

[19] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979. 2, 4, 12

[20] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.*, 12(2):347–353, 1983. 2, 4, 27

[21] C. G. Nelson. An $n^{\log n}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem. Technical Report CS-TR-78-689, Dept. Computer Science, Stanford University, Stanford, CA, 1978. 2

[22] N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 100–111, 2017. 6, 27

[23] I. Post and Y. Ye. The simplex method is strongly polynomial for deterministic markov decision processes. *Math. Oper. Res.*, 40(4):859–868, 2015. 3, 5

[24] T. Radzik. Newton's method for fractional combinatorial optimization. In *33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992. 4, 5, 12

[25] 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. 4, 5, 7, 8, 12

[26] 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. 5

[27] R. E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981. 2, 3, 7

[28] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998. 1

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

[30] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 6

[31] S. A. Vavasis and Y. Ye. A primal-dual interior point method whose running time depends only on the constraint matrix. *Math. Program.*, 74:79–120, 1996. 6

[32] L. A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.*, 42(1):179–211, 2017. 6, 27

[33] Q. Wang, X. Yang, and J. Zhang. A class of inverse dominant problems under weighted $\ell_\infty$ norm and an improved complexity bound for Radzik's algorithm. *J. Global Optimization*, 34(4):551–567, 2006. 5, 12

[34] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Math. Oper. Res.*, 27(3):445–459, 2002. 27

[35] Y. Ye. A new complexity result on solving the Markov decision problem. *Math. Oper. Res.*, 30(3):733–749, 2005. 5

[36] Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. *Math. Oper. Res.*, 36(4):593–603, 2011. 5

## A Missing Proofs

*Proof of Lemma 2.5.* Consider the polyhedron $P \subseteq \mathbb{R}^m$ defined by the following constraints:

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

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 the matrix $A'$ by replacing the $i$-th column with the vector $b'$. Recall that the determinant of a matrix $M \in \mathbb{R}^{m\times m}$ is equal to

$$\det M = \sum_{\sigma\in S_m}\left(\operatorname{sgn}(\sigma)\prod_{i=1}^m M_{i,\sigma(i)}\right).$$

Since every row of the matrix $A'_i$ has at most $2n$ nonzero entries, we have $|\det A'_i| \leq a_i^m(2n)^m$, where $a_i$ is the maximum absolute value of an entry in matrix $A'_i$. Observe that the entries of matrix $A$ and vector $b$, and consequently the entries of every matrix $A'_i$, are integers from the interval $[-3,3]$. Thus, $|\det A'_i| \leq (6n)^m$ for all $i \in [m]$. As the matrix $A'$ is nonsingular, we also have $|\det A'| \geq 1$, which implies that $c'_i \leq (6n)^m$ for all $i \in [m]$. Finally, using the constraints which define the polyhedron $P$, we obtain

$$1 = (x^{(k)})^\top c' \leq \frac{(x^{(1)})^\top c'}{2^{k-1}} \leq \frac{n(6n)^m}{2^{k-1}}.$$

So, $k \leq \log(6^m n^{m+1}) + 1 = O(m\log n)$ as desired. □