A Simpler and Faster Strongly Polynomial Algorithm for Generalized Flow Maximization

Neil Olver*
Department of Econometrics and Operations Research, Vrije Universiteit Amsterdam, and CWI Amsterdam, The Netherlands
n.olver@vu.nl

László A. Végh†
Department of Mathematics, London School of Economics, United Kingdom
L.Vegh@lse.ac.uk

ABSTRACT

We present a new strongly polynomial algorithm for generalized flow maximization. The first strongly polynomial algorithm for this problem was given very recently by Végh; our new algorithm is much simpler, and much faster. The complexity bound $O((m + n \log n)mn \log(n^2/m))$ improves on the previous estimate obtained by Végh by almost a factor $O(n^2)$. Even for small numerical parameter values, our algorithm is essentially as fast as the best weakly polynomial algorithms. The key new technical idea is relaxing primal feasibility conditions. This allows us to work almost exclusively with integral flows, in contrast to all previous algorithms for the problem.

CCS CONCEPTS

• **Theory of computation → Network flows**;

KEYWORDS

Generalized flow, network flow, strongly polynomial

ACM Reference format:
Neil Olver and László A. Végh. 2017. A Simpler and Faster Strongly Polynomial Algorithm for Generalized Flow Maximization. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 12 pages.
DOI: 10.1145/3055399.3055439

1 INTRODUCTION

In the maximum generalized flow problem, we are given a directed graph $G = (V, E)$ with a sink node $t \in V$ and gain factors $\gamma_e > 0$ on the edges. Flow entering at edge e gets rescaled by the factor $\gamma_e > 0$ when traversing the edge. The goal is to maximize the amount of flow sent to the sink. The problem has a rich history: it was first formulated by Kantorovich [15] in 1939, in the same paper where Linear Programming was introduced. We refer the reader to [1, Chapter 15] for applications of the model.

*Partially supported by an NWO Veni grant.
†Supported by EPSRC First Grant EP/M02797X/1.

STOC'17, Montreal, Canada

DOI: 10.1145/3055399.3055439

Early combinatorial algorithms were developed by Dantzig [4] and by Onaga [20]. The first polynomial-time combinatorial algorithm was given by Goldberg, Plotkin, and Tardos [8] in 1991. A large number of weakly polynomial algorithms were developed in the subsequent 20 years, e.g. [2, 6, 10–13, 16, 23, 24, 28, 29, 31, 33]. Let n denote the number of nodes, m the number of edges of the graph, and let be B be the largest integer in the description of the gain factors, capacities, and node demands. Among the previous algorithms, the best running times are the $O(m^{1.5}n^2 \log(nB))$ interior point method by Vaidya [29]; and the $O(mn(m + n \log n) \log B)$ combinatorial algorithm by Radzik [23]. Interior point methods can obtain fast approximate solutions for lossy networks, i.e. if $\gamma_e \leq 1$ for all arcs. The result of Daitch and Spielman [3] finds an additive ε-approximate solution in $\tilde{O}(m^{3/2} \log^2(B/\varepsilon))$, recently improved by Lee and Sidford [17] to $\tilde{O}(m\sqrt{n} \log^{O(1)}(B/\varepsilon))$.[1] However, these results do not obtain an exact solution.

Resolving a longstanding open question, the first strongly polynomial algorithm was given in [32], with running time $O(n^3m^2)$. The main progress in the algorithm is that, within a strongly polynomial number of steps, we can identify at least one arc that must be tight in every dual optimal solution. Consequently, we can reduce the size of the instance by contracting such arcs. The algorithm is based on *continuous scaling*, a novel version of the classical scaling method. The algorithm is technically very complicated. Our new algorithm works along broadly similar lines, and also involves arc contractions as a main vehicle of progress, with path augmentation and relabelling operations being used to find an arc to contract. But our algorithm introduces a number of new conceptual and technical ideas compared to [32] and previous literature.

We give a detailed technical overview and comparison at the beginning of Section 3, after having defined the basic notation and concepts. Here we briefly highlight a key novelty. Unlike all previous combinatorial algorithms, we do not maintain a feasible primal solution (i.e., flow). Instead, we ensure that the dual solution has a certain property that keeps us "within reach" of a feasible primal solution that respects certain complementary slackness conditions. So while our algorithm is a primal-dual algorithm, in a sense it does not keep track of the "real" primal but only a proxy for it. Working with an infeasible primal solution turns out to have major benefits; in particular, we are able to work almost exclusively with integer flows, simplifying matters dramatically.

Our running time bound is $O((m + n \log n)mn \log(n^2/m))$. Besides the substantial improvement over [32], this is also better than the interior point method of Vaidya [29] for arbitrary values of the

[1] The notation $\tilde{O}(.)$ hides further polylog(m) factors.

complexity parameter B, and better than Radzik's combinatorial algorithm [23] if $B = \omega(n^2/m)$.

The context of strongly polynomial Linear Programming. The existence of a strongly polynomial algorithm for Linear Programming (LP) is a central open question. Consider an LP in the following standard form, with $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$.

$$\begin{aligned} \min\ & c^\top x \\ Ax &= b \\ x &\geq 0. \end{aligned} \tag{LP}$$

An LP algorithm is strongly polynomial, if the number of elementary arithmetic operations is bounded by $\text{poly}(n, m)$. Furthermore, the algorithm must be in PSPACE, that is, the numbers occurring in the computations must remain polynomially bounded in the input size. The most general strongly polynomial computability results are due to Tardos [27], and to Vavasis and Ye [30]. In these results, the running time only depends on the matrix A, but not on the right hand side b or on the cost c. Tardos [27] assumes that A is integer, and obtains a running time $\text{poly}(n, m, \log \Delta)$, where Δ is an upper bound on the largest subdeterminant of A. In particular, if all entries in A are integers of size $\text{poly}(n, m)$, this algorithm is strongly polynomial. These are called *"combinatorial LP's"* since most network optimization problems can be expressed with small integer constraint matrices. Vavasis and Ye [30] waive the integrality assumption, replacing Δ with a more general condition number.

A different, natural restriction on (LP) is to impose constraints on the nonzero elements. Assume that every column of the constraint matrix A has only two nonzero entries, but these can be arbitrary numbers. Let $\mathcal{M}_2(n, m) \subseteq \mathbb{R}^{n\times m}$ denote the set of all such matrices. The results [27, 30] do not apply for LPs with such constraint matrices. It is easy to see that every LP can be equivalently transformed to one with at most three nonzeros per column.

For the dual feasibility problem, that is, finding a feasible solution to $A^\top y \geq c$ for $A \in \mathcal{M}_2(m, n)$, Megiddo [18] gave a strongly polynomial algorithm. In fact, the notion of strongly polynomial algorithms was formally defined in the same paper (called "genuinely polynomial"). The primal feasibility problem, that is finding a feasible solution to $Ax = b$, $x \geq 0$ for $A \in \mathcal{M}_2(n, m)$, can be reduced to generalized flow maximization [32, Section 8]. Hence the algorithm in [32] as well as our new algorithm, give a strongly polynomial algorithm for primal feasibility.

It remains an important open question to solve the optimization (LP) for a constraint matrix $A \in \mathcal{M}_2(n, m)$ in strongly polynomial time. This problem reduces to the minimum cost generalized flow problem [14]. As our new algorithm gives a simple and clean solution to flow maximization, we expect that the ideas developed here bring us closer to resolving this problem.

2 PROBLEM AND PRELIMINARIES

Let $\mathbb{R}_+$ and $\mathbb{R}_{++}$ denote the nonnegative and positive reals respectively; similarly let $\mathbb{Z}_+$ and $\mathbb{Z}_{++}$ denote the nonnegative and positive integers. Let $\bar{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$, and similarly for other cases. For a vector x, $\|x\|_p$ denotes its p-norm. For a vector $h \in \mathbb{R}^X$ and any $S \subseteq X$, we use $h(S)$ to denote $\sum_{x\in S} h_x$.

Let $G = (V, E)$ be a simple directed graph, which we assume to be connected in an undirected sense. Let $n := |V|$ and $m := |E|$. For an arc set $F \subseteq E$, let $\overleftarrow{F} := \{ji : ij \in F\}$ denote the reversed arc set, and $\overleftrightarrow{F} := F \cup \overleftarrow{F}$. For a subset $S \subseteq V$, we let $E[S]$ denote the set of arcs with both endpoints inside S. Further, we let $\delta^-(S)$ and $\delta^+(S)$ denote the set of incoming and outgoing arcs, respectively. If $S = \{i\}$, we use the simplified notation $\delta^-(i)$ and $\delta^+(i)$. Let $d_i := |\delta^-(i) \cup \delta^+(i)|$ denote the total degree of i.

An instance of the generalized flow problem is given as $\mathcal{I} = (V, E, t, \gamma, b)$, where (V, E) is a directed graph, $t \in V$ is a sink node, $\gamma \in \mathbb{R}^E_{++}$ is the vector of gain factors, and $b \in \mathbb{R}^{V\setminus\{t\}}$ is the vector of node demands. Let us partition the nodes according to the sign of the demand.

$$\begin{aligned} V^- &:= \{i \in V \setminus \{t\} : b_i < 0\}, \\ V^0 &:= \{i \in V \setminus \{t\} : b_i = 0\}, \\ V^+ &:= \{i \in V \setminus \{t\} : b_i > 0\}. \end{aligned}$$

The *net flow* at a node i is defined as

$$\nabla f_i := \sum_{e\in\delta^-(i)} \gamma_e f_e - \sum_{e\in\delta^+(i)} f_e.$$

We are ready to formulate the generalized flow maximization problem.

$$\begin{aligned} \max\quad & \nabla f_t \\ \text{s.t.}\quad & \nabla f_i \geq b_i \quad \forall i \in V \setminus \{t\} \\ & f \geq 0. \end{aligned} \tag{P}$$

The problem can be formulated in multiple equivalent variants. In particular, a standard formulation is to use arc capacities and zero node demands. All these formulations can be efficiently reduced to (P); in fact, every LP in the form $Ax = b, x \geq 0$ for $A \in \mathcal{M}_2(n, m)$ reduces to (P) (see [32, Section 8] for the reductions). The special case when $\gamma_e = 1$ for all $e \in E$ corresponds to the standard network flow model; we will refer to standard network flows as *regular flows* to differentiate from generalized flows. The dual program can be transformed to the following form. The dual variable for node i would be μ_t/μ_i. Nodes other than t are allowed to have $\mu_i = \infty$; this corresponds to dual values 0.

$$\begin{aligned} \max\quad & \mu_t \sum_{j\in V\setminus\{t\}} \frac{b_j}{\mu_j} \\ \text{s.t.}\quad & \mu_j \geq \gamma_{ij}\mu_i \quad \forall ij \in E \\ & \mu_t \in \mathbb{R}_{++} \\ & \mu_i \in \bar{\mathbb{R}}_{++} \quad \forall i \in V \setminus \{t\}. \end{aligned} \tag{D}$$

Our main result is the following.

Theorem 2.1. *There exists a strongly polynomial algorithm, that, for any input instance $\mathcal{I} = (V, E, t, \gamma, b)$, finds optimal solutions to* (P) *and* (D) *in $O((m + n\log n)mn\log(n^2/m))$ arithmetic operations.*

Relabellings. We interpret the dual solutions as *relabellings*, the basic vehicle of our algorithm. This is a standard technique used in the vast majority of generalized flow algorithms. A feasible solution $\mu \in \bar{\mathbb{R}}^V_{++}$ to (D) is called a *feasible labelling*. We define

$$f^\mu_{ij} := \frac{f_{ij}}{\mu_i} \quad \forall ij \in E.$$

The multiplier μ_i can be interpreted as a change of the unit of measurement at node i. An equivalent problem instance is obtained

by defining

$$\gamma^{\mu}_{ij} := \gamma_{ij} \cdot \frac{\mu_i}{\mu_j}, \qquad \nabla f^{\mu}_i := \frac{\nabla f_i}{\mu_i}, \qquad \text{and} \qquad b^{\mu}_i := \frac{b_i}{\mu_i}.$$

We use the convention $\gamma^{\mu}_{ij} = 1$ if $\mu_i = \mu_j = \infty$. Then the feasibility of μ to (D) is equivalent to $\gamma^{\mu}_e \leq 1$ for all $e \in E$. We call an arc $e \in E$ *tight* with respect to μ, if $\gamma^{\mu}_e = 1$. Let E^{μ} and $\overleftrightarrow{E}^{\mu}$ denote the set of tight arcs for μ in E and in $\overleftrightarrow{E}$, respectively.

For a flow $f \in \mathbb{R}^E_+$, we define the residual graph $G_f = (V, E_f)$ with $E_f = E \cup \{ji : ij \in E, f_{ij} > 0\}$. The latter set of arcs are called *reverse arcs*. For a reverse arc ji, we define $\gamma_{ji} := 1/\gamma_{ij}$, and $f_{ji} := -\gamma_{ij} f_{ij}$. By increasing (decreasing) f_{ji} by α on a reverse arc $ji \in E_f$, we mean decreasing (increasing) f_{ij} by α/γ_{ij}.

Let us define the *excess* of a node i under f^{μ} to be the amount $\nabla f^{\mu}_i - b^{\mu}_i$; so f^{μ} is feasible if all nodes have nonnegative excess. We also define the total (positive) excess and the total deficit of f^{μ} as

$$\mathrm{Ex}(f, \mu) := \sum_{i \in V \setminus \{t\}} \max\left\{\nabla f^{\mu}_i - b^{\mu}_i, 0\right\} \tag{1}$$

$$\text{and} \quad \mathrm{Def}(f, \mu) := \sum_{i \in V \setminus \{t\}} \max\left\{b^{\mu}_i - \nabla f^{\mu}_i, 0\right\}. \tag{2}$$

In the analysis, it will be more convenient to work with the following relaxed version of the total excess.

$$\Xi(f, \mu) := \sum_{i \in V \setminus \{t\}} \max\{\nabla f^{\mu}_i - b^{\mu}_i, 2\}. \tag{3}$$

Fitting pairs and optimality. Let $f \in \mathbb{R}^E_+$ and $\mu \in \mathbb{R}^V_{++}$. We say that (f, μ) is a *fitting pair*, if μ is feasible to (D), and $f_e > 0$ implies $\gamma^{\mu}_e = 1$. We also say that f *fits* μ, or μ *fits* f. Equivalently, (f, μ) is a fitting pair if the entire support of f is tight with respect to μ.

Note that the definition requires that μ is finite, and feasible to (D), but not the feasibility of f to (P). In fact, we will allow flows $f \in \mathbb{R}^E_+$ in the algorithm that violate the node balance constraints in (P). Fitting captures a complementary slackness property. For the case when μ is finite, optimality can be described as follows. The lemma is an immediate consequence of complementary slackness.

Lemma 2.2. *Let (f, μ) be a fitting pair such that $\nabla f_i = b_i$ for all $i \in V \setminus \{t\}$. Then f is an optimal solution to* (P) *and μ is an optimal solution to* (D).

Note that for a fitting pair (f, μ), $\gamma^{\mu}_e = 1$ for all $e \in \mathrm{supp}(f)$. Consequently, f^{μ} is a regular flow. Thus, we can use all known results and algorithms to manipulate regular flows. In particular, provided an optimal solution μ to (D) Lemma 2.2 enables finding an optimal solution to (P) by solving a feasible circulation problem on the set of tight arcs for μ (see Section 3.4).

We say that a feasible labelling μ is *safe*, if there exists a feasible solution f to (P) such that (f, μ) is a fitting pair. This will be a crucial property in our algorithm. It can be easily characterized by a cut condition, a simple corollary of Hoffman's theorem ([25, Theorem 11.2]).

Lemma 2.3. *The labelling μ is safe if and only if for every set $X \subseteq V \setminus \{t\}$ with $\delta^-(X) \cap E^{\mu} = \emptyset$, the condition $b^{\mu}_i(X) \leq 0$ holds.*

Initial solutions. The overall scheme of our algorithm will be akin to the two phase simplex method. In the first phase, we obtain a fitting pair $(\bar{f}, \bar{\mu})$ of feasible primal and dual solutions, or conclude that (P) is infeasible or unbounded. In the second phase, we compute an optimal solution, starting from $(\bar{f}, \bar{\mu})$.

The first phase will be implemented by adding a dummy sink and new arcs to the network, so that there is a trivial initial fitting pair. The optimal solution to the first phase problem will be the input fitting pair to the second phase. We describe the first phase in Section 5. Hence, we make the following assumption in the algorithm.

(⋆) An initial fitting pair $(\bar{f}, \bar{\mu})$ is given, where $\bar{f} \in \mathbb{R}^E_+$ is feasible to (P) and $\bar{\mu} \in \mathbb{R}^V_{++}$ is feasible to (D).

For reasons of technical simplicity, we also make the following standard assumption.

(⋆⋆) There is a directed path in E from i to t for every $i \in V$.

Assuming that the objective of (P) is bounded, we can guarantee this assumption by adding new *it* arcs with very small gain factors (e.g. [8, 32]). In Section 5, we provide the explicit construction. The main benefit of this assumption is that every feasible solution to (D) will be finite.

Rounding. Our algorithm will work exclusively with fitting pairs (f, μ) where the relabelled flow f^{μ} is integral. Since the initial fitting pair guaranteed by (⋆) need not have this property, we will need the following Lemma, which follows from elementary integrality properties of the flow polytope.

Lemma 2.4. *Let (f, μ) be a fitting pair. Then there exists a $\tilde{f}$ that fits μ, such that $\tilde{f}^{\mu} \in \mathbb{Z}^E_+$, and $\lfloor \nabla f^{\mu}_i \rfloor \leq \nabla \tilde{f}^{\mu}_i \leq \lceil \nabla f^{\mu}_i \rceil$.*

Proof. Consider the feasible circulation problem on (V, E^{μ}), with lower and upper node demands $\lfloor \nabla f^{\mu}_i \rfloor$ and $\lceil \nabla f^{\mu}_i \rceil$. The flow f^{μ} is a feasible solution; hence, there exists an integer solution $\tilde{g}$, which can be found by a maximum flow algorithm. Then $\tilde{f}_{ij} := \tilde{g}_{ij} \mu_i$ is the desired solution. □

We let the subroutine Round(f, μ) implement the construction in the above proof.

Network structures. Let us now define some network structures relevant for generalized flows. For an arc set $F \subseteq \overleftrightarrow{E}$, we let $\gamma(F) = \prod_{e \in F} \gamma_e$; $\gamma^{\mu}(F)$ is defined similarly. A cycle C is called a *flow generating cycle*, if $\gamma(C) > 1$. For any node i incident to C, we can increase the excess of i by sending flow around C. We note that for any labelling μ, $\gamma^{\mu}(C) = \gamma(C)$. This immediately implies that if (f, μ) is a fitting pair, then E_f may contain no flow generating cycles. Under assumption (⋆⋆), a flow $f \in \mathbb{R}^E_+$ is optimal if and only if $\nabla f_i = b_i$ for all $i \in V \setminus \{t\}$, and the residual graph E_f contains no flow generating cycles.

For a path P between nodes i and j, $\gamma^{\mu}(P) = \gamma(P) \cdot \mu_i/\mu_j$. This implies that for a fitting pair (f, μ), a tight path from i to j is a *highest gain augmenting path*. By augmenting f^{μ} by α on a tight path $P \subseteq E_f$, we mean increasing f_{ij} by $\alpha \mu_i$ for every arc ij in P.

3 THE GENERALIZED FLOW ALGORITHM

3.1 Technical Overview

Let us say that an arc $e \in E$ is *contractible*, if e must be tight with respect to *any* optimal dual solution μ^* to (D). The main progress in the algorithm will be identifying a new contractible arc within a strongly polynomial number of iterations, and reducing the size of the graph G by contracting such arcs. Once an optimal dual solution is found in the contracted instance, it extends straightforwardly to the original graph. Hence in a strongly polynomial number of steps, we will be able to find an optimal dual solution. Finally, a primal optimal solution can be obtained via a single (regular) maximum flow computation.

The same scheme was used in [32]. In fact, this is a classical scheme for obtaining strongly polynomial algorithms for the classical minimum-cost circulation problem. The algorithm in [32] and ours are direct descendants of Orlin's algorithm [21]. This runs the classical weakly polynomial Edmonds-Karp scaling algorithm [5] for minimum-cost circulations, only to identify an *"abundant arc"*. Then the network size can be reduced by contracting such an arc. However, the idea of using a modified (rounded) problem instance only to identify a tight constraint goes back to the first strongly polynomial algorithm for minimum-cost circulations by Tardos [26].

Compared to [32], our augmenting path subroutine for identifying a contractible arc is vastly simpler and more efficient. The crucial idea is *relaxing the feasibility of the flow* f in the augmenting path algorithm. That is, nodes i with $\nabla f_i < b_i$ will be allowed. This is a quite radical change compared to all previous algorithms. In fact, "fixing" a node deficit can be very difficult: compensating for just a tiny shortfall in node demands can be at the expense of an arbitrarily large drop in the objective value. We avoid this problem by maintaining that the labelling μ remains *safe* throughout. That is, there exists always a feasible flow f' fitting μ. Standard network flow theory shows, that, given flows f' and f, there exists a flow g with $\nabla f'_i \le \nabla g_i \le \nabla f_i$ for all $i \in V \setminus \{t\}$. Consequently, g is feasible and $\mathrm{Ex}(g,\mu) \le \mathrm{Ex}(f,\mu)$. Then we can use the flow g instead of f to identify contractible arcs.

It turns out that many serious technical difficulties in previous algorithms were due to insisting on flow feasibility. Once feasibility is relaxed, the algorithm suddenly becomes much simpler and more natural. We need to maintain the safety of the labelling, but this happens automatically, without additional effort. The most salient consequences are the following.

- First of all, we can easily *maintain a fitting pair* (f,μ) throughout. In contrast, [32] had to introduce a relaxation of this concept called Δ-feasibility, depending on the current scaling factor Δ. An earlier algorithm that maintained a fitting pair throughout was the algorithm of Goldfarb, Jin, and Orlin [12], however, it came at the expense of maintaining arc imbalances in an intricate bookkeeping framework.
- Although our algorithm can be seen as an enhanced version of the *continuous scaling* technique in [32], the description does not even include a scaling factor, prevalent in the previous combinatorial methods. Instead, we maintain that *the relabelled flow* f^μ *is integral* throughout, except for the very final step when an exact optimum is computed. This is unprecedented in previous algorithms, and surprising because the generalized flow problem is perceived as a genuinely non-integral problem. Let us note that the value of μ_t corresponds to the scaling factor Δ in [32] and other scaling methods; we relax the standard requirement $\mu_t = 1$ so that we can work with integer solutions.
- A main reason for the running time efficiency is a new, *additive* potential analysis, compared to the *multiplicative* analysis in [32]. In both algorithms, the main progress is measured in the potential $\sum_{i \in V\setminus\{t\}} |b_i^\mu|$; once this becomes sufficiently large, the existence of a contractible arc is guaranteed. The running time estimates are given by charging the number of path augmentations against this potential. In [32], this is measured by arguing about the cumulative decrease in the scaling factor Δ in a rather indirect way. Instead, we have a very clean way of arguing that, roughly speaking, every path augmentation decreases the potential by one.

In a strongly polynomial algorithm, one also needs to guarantee that the sizes of numbers remain polynomially bounded in the input size. In [32], this required cumbersome additional rounding steps. In contrast, this can be easily achieved in our new algorithm.

A further distinguishing feature of our algorithm is that we do not use an initial cycle cancelling subroutine. Most combinatorial methods start with the assuming the existence of an initial fitting pair as in ($\star$). In order to obtain this, flow generating cycles have to be eliminated first. Radzik [22] adapted the Goldberg-Tarjan minimum-mean cycle cancelling algorithm [9] to cancel all flow generating cycles in strongly polynomial time. We avoid using this subroutine, and instead perform our algorithm in two phases, as in the two phase simplex algorithm. In the first phase for feasibility, we obtain the fitting pair used as the starting for the second phase. We note that the running time of our algorithm is better than the running time of Radzik's cycle cancelling subroutine [22].

However, this trick of using a two phase implementation is not particular to our current algorithm. In fact, the same scheme could be applied also to the previous algorithms, including [32].

3.2 The Overall Algorithm

Algorithm 1 Maximum Generalized Flow

Input: The instance $\mathcal{I} = (V, E, t, \gamma, b)$ satisfying ($\star\star$), with an initial feasible solution $(\bar{f}, \bar{\mu})$ provided as in ($\star$).
Output: Optimal solutions to (P) and (D).

1: $\Delta \leftarrow \max_{i \in V\setminus\{t\}} \nabla \bar{f}_i^{\bar{\mu}} - b_i^{\bar{\mu}}$; $\mu \leftarrow \bar{\mu}\Delta$.
2: $f \leftarrow$ Round$(\bar{f}, \mu)$.
3: **while** $V^- \cup V^+ \neq \emptyset$ **do**
4: $\quad (f,\mu) \leftarrow$ Produce-Plentiful-Node(f,μ).
5: $\quad (\mathcal{I}, f, \mu) \leftarrow$ Reduce$(\mathcal{I}, f, \mu)$.
6: $\mu \leftarrow$ Expand-to-Original(μ)
7: $f \leftarrow$ Compute-Primal(μ)
8: **return** (f,μ).

The overall algorithm is described in Algorithm 1. We assume ($\star$), that is, an initial solution $(\bar{f}, \bar{\mu})$ is given. The first two lines preprocess this solution into (f, μ), first rescaling using the rounding subroutine. As a result, we obtain a fitting pair (f, μ) such that $f^\mu \in \mathbb{Z}^E_+$, and further,

$$b_i^\mu - 1 \le \nabla f_i^\mu \le b_i^\mu + 2 \quad \forall i \in V \setminus \{t\}. \tag{4}$$

This property follows by the choice of Δ and the properties of the subroutine Round, defined after Lemma 2.4. The bulk of the algorithm iterates between two subroutines, Produce-Plentiful-Node (Section 3.5), and Reduce (Section 3.3.) Produce-Plentiful-Node is a simple augmenting path algorithm, updating the flows and the labels. It is guaranteed to terminate with a "plentiful node" (defined in Section 3.3), which has an incident contractible arc. The subroutine Reduce identifies and contracts such an arc, and updates the flows appropriately.

Finally, once all nodes have $b_i = 0$, we terminate with μ being an optimal dual solution to the current contracted instance; this is witnessed by the optimal primal $f = \mathbf{0}$. The subroutine Expand-to-Original naturally maps μ back to the original graph, and Compute-Primal then computes a matching primal optimum from this. These are described in Section 3.4

The values of n and m will always refer to the number of nodes and arcs in the original input graph. Hence in later stages, the graph will have less than n nodes.

3.3 Arc Contractions

In this section, we describe the subroutine Reduce, that is responsible for the main progress, by contracting arcs of the graph. We first formulate a sufficient condition to identify contractible arcs. Let us call a node $i \in V \setminus \{t\}$ *plentiful* with respect (f, μ), if

$$|b_i^\mu| \ge 3n(d_i + 1). \tag{5}$$

Theorem 3.1. *Let $f \in \mathbb{R}^E_+$ and $\mu \in \mathbb{R}^V_{++}$, such that (f, μ) is a fitting pair with $f^\mu \in \mathbb{Z}^E_+$, μ is safe, and $\Xi(f, \mu) < 2n$. Assume further that there exists a plentiful node i. Then there exists a contractible arc e incident to i, and it can be found in strongly polynomial time.*

We recall that Ξ was defined in (3). We now formulate two simple lemmas in preparation for the proof.

Lemma 3.2. *Let (f, μ) be a fitting pair with $f^\mu \in \mathbb{Z}^E_+$, and assume μ is safe. Then there exists a flow $g \in \mathbb{R}^E_+$ fitting μ with $g^\mu \in \mathbb{Z}^E_+$ and*

$$\lfloor b_i^\mu \rfloor \le \nabla g_i^\mu \le \max\{\nabla f_i^\mu, \lceil b_i^\mu \rceil\} \quad \forall i \in V \setminus \{t\}. \tag{6}$$

Further, such a flow can be found by a single maximum flow computation, and it satisfies $\mathrm{Ex}(g, \mu) \le \Xi(f, \mu)$.

Proof. The safety of μ provides a feasible flow f' that fits μ. The statement is an immediate consequence of [25, Corollary 11.2j] applied to the regular flows f^μ and f'^μ, combined with the integrality of the flow polytope. The last claim follows since $\nabla g_i^\mu > \nabla f_i^\mu$ is only possible if $\nabla f_i^\mu - b_i^\mu < 0$, and in that case $\nabla g_i^\mu < b_i^\mu + 1$. □

A contractible arc can be obtained using the following lemma.

Lemma 3.3. *Let (g, μ) be a fitting pair with μ being safe. If $g_e^\mu > \mathrm{Ex}(g, \mu) + \mathrm{Def}(g, \mu)$ for an arc $e \in E$, then e is contractible.*

We sketch the proof here; full details can be found in the full version [19]. First, using safety of μ, we show that there exists a feasible flow $\tilde{g}$ fitting μ such that $\|g^\mu - \tilde{g}^\mu\|_\infty \le \mathrm{Def}(g, \mu)$, and $\mathrm{Ex}(\tilde{g}, \mu) \le \mathrm{Ex}(g, \mu)$. Then, we show that there exists an optimal g^* with $\|\tilde{g}^\mu - (g^*)^\mu\|_\infty \le \mathrm{Ex}(\tilde{g}, \mu)$. Both parts use simple flow decomposition techniques. The first part is a standard argument for regular flows. The second part uses flow decomposition of generalized flows. Similar claims have been proved in [32, Theorem 5.1] or [23, Lemma 5]; our proof follows the same lines. The safety property of μ is of essence: in the Appendix, we also show an example where the flow decomposition argument fails if safety is not assumed.

Proof of Theorem 3.1. By Lemma 3.2, we can obtain a flow g fitting μ such that $\mathrm{Ex}(g, \mu) \le \Xi(f, \mu) < 2n$. By construction, $\mathrm{Def}(g, \mu) < n - 1$. Now if $i \in V^-$, then

$$\sum_{e \in \delta^+(i)} g_e^\mu \ge -\nabla g_i^\mu \ge -b_i^\mu - \mathrm{Ex}(g, \mu) > 3n|\delta^+(i)|,$$

implying the existence of an arc $e \in \delta^+(i)$ with $g_e^\mu > 3n$. If $i \in V^+$, then

$$\sum_{e \in \delta^-(i)} g_a^\mu \ge \nabla g_i^\mu \ge b_i^\mu > 3n|\delta^-(i)|,$$

implying the existence of an arc $e \in \delta^-(i)$ with $g_e^\mu > 3n$. Applying Lemma 3.3 completes the proof. □

The Reduce subroutine. We are now ready to describe the subroutine Reduce (Algorithm 2). This implements the steps of the proof of Theorem 3.1 to identify a contractible arc. Once identified, such an arc $a = pq$ is contracted in the obvious way into the node q: we move the arcs incident to p to q, and update the gain factors appropriately.

Algorithm 2 Subroutine Reduce

Input: An instance $\mathcal{I} = (V, E, t, \gamma, b)$ and a fitting pair (f, μ) with $f^\mu \in \mathbb{Z}^E_+$.
Output: A contracted instance with a fitting pair (g, μ) for it, with g^μ integral.

Compute a flow $g^\mu \in \mathbb{Z}^E_+$ satisfying (6).
while $\exists e = pq \in E : g_e^\mu > \mathrm{Ex}(g, \mu) + \mathrm{Def}(g, \mu)$ **do** ▷ Contract arc e.
  for $i \in \delta^-(p) \setminus \{q\}$ **do**
    Replace arc ip by a new arc iq; $\gamma_{iq} \leftarrow \gamma_{ip}\gamma_{pq}$.
  for $i \in \delta^+(p) \setminus \{q\}$ **do**
    Replace arc pi by a new arc qi; $\gamma_{qi} \leftarrow \gamma_{qi}/\gamma_{pq}$.
  if $t \notin \{p, q\}$ **then** $b_q \leftarrow b_q + \gamma_{pq} b_p$.
  if $t = p$ **then** rename q to t.
  $V \leftarrow V \setminus \{p\}$.
  If parallel arcs are created, keep only one with the highest gain factor from each bundle.
return $(\mathcal{I}, g, \mu)$.

3.4 Uncontracting and Computing the Optimum

Once an optimal solution is found, the subroutine Expand-to-Original(μ) reverts all contractions, starting with the last one. When uncontracting the arc pq, we define $\mu_p = \mu_q/\gamma_{pq}$. The following is an easy consequence of the definition of a contractible arc; a full proof is given in the Appendix.

Lemma 3.4. *Expand-to-Original(μ) yields an optimum dual solution to the original instance, as long as μ is an optimal solution to the contracted instance.*

The subroutine Compute-Primal(μ) computes an optimal primal solution f based on the finite optimal dual solution μ, by solving the following circulation problem on the tight arcs of μ. We set node demands $b'_i = b^\mu_i$ for $i \in V \setminus \{t\}$ and $b'_t = -\sum_{i\in V\setminus\{t\}} b^\mu_i$ for the sink node t. If g is a feasible solution to this problem, then $f_{ij} = g_{ij}\mu_i$ is an optimal solution to (P), since it is feasible and satisfies complementary slackness with μ.

3.5 Obtaining a Plentiful Node

Algorithm 3 Produce-Plentiful-Node

Input: Fitting pair (f, μ) with $f^\mu \in \mathbb{Z}^E_+$, in a network with $V^- \cup V^+ \neq \emptyset$.

Output: Fitting pair (f, μ) with $f^\mu \in \mathbb{Z}^E_+$, such that there exists at least one plentiful node in V.

1: **while** there are no plentiful nodes **do**
2: *Augmentation part of the iteration*
3: $Q \leftarrow \{t\} \cup \{j \in V \setminus \{t\} : \nabla f^\mu_i < b^\mu_i\}$.
4: $\bar{S} \leftarrow \{i \in V : \exists$ a tight i-Q-path in $E_f\}$.
5: **while** there exists a node $i \in \bar{S} \cap V^-$ with $\nabla f^\mu_i \geq b^\mu_i + 1$ **do**
6: Augment f^μ by sending 1 unit from i to a vertex in Q along tight arcs.
7: Update Q and $\bar{S}$.
8: *Label update part of the iteration*
9: $S \leftarrow$ the connected component of $\bar{S}$ w.r.t. $\overleftrightarrow{E^\mu}$ containing t.
10: $\alpha_0 \leftarrow \min_{e\in\delta^-(S)} 1/\gamma^\mu_e$.
11: For each $i \in S \cap V^-$, choose $\alpha_i \in \left[\frac{1-\nabla f^\mu_i}{-b^\mu_i}, \frac{2-\nabla f^\mu_i}{-b^\mu_i}\right)$.
12: For each $i \in S \cap V^+$, choose $\alpha_i \in \left[\frac{3n(d_i+1)}{b^\mu_i}, \frac{3n(d_i+1)+1}{b^\mu_i}\right)$.
13: $\alpha \leftarrow \min\{\alpha_0, \min_{i\in S\cap(V^-\cup V^+)} \alpha_i\}$.
14: $f_e \leftarrow f_e/\alpha$ for all $e \in E[S]$.
15: $\mu_i \leftarrow \mu_i/\alpha$ for all $i \in S$.
16: **return** (f, μ).

Algorithm 3 gives the description of Produce-Plentiful-Node. The main objective of the subroutine is to make sure a plentiful node (as in (5)) appears. We terminate once such a node is found.

Each iteration consists of a primal update part followed by a dual update part. In the primal update part, path augmentations on f^μ along tight arcs are performed, where each path augmentation sends a unit of flow from a node with excess at least one to either the sink, or to a node with negative excess. For reasons that will become clear in the potential analysis, we only consider path augmentations that start from a vertex in V^-.

Assume no further path augmentations are possible. Let $\bar{S}$ be the set of nodes that can reach t or a node with negative excess on a tight path in the auxiliary graph, and let $S \subseteq \bar{S}$ be the undirected connected component in $E^\mu[\bar{S}]$ containing t.[2] Then $\delta(S)$ contains no incoming tight arcs, and hence there is no incoming or outgoing flow from S. In the label update step, we scale down all labels μ_i inside S, as well as all flow within S, by the same factor. This ensures that f^μ remains completely unchanged. The scaling stops when either:

(i) An edge entering S becomes tight (step 10),
(ii) a node's excess $\nabla f^\mu_i - b^\mu_i$ increases above 1 (step 11), or
(iii) a node in V^+ becomes plentiful (step 12).

In the latter two cases, we allow some flexibility; it is fine if a node's excess increases above 1, but we ensure it never exceeds 2; and we allow a node to slightly exceed its plentiful threshold. This allows room to manoeuvre in the numerical implementation (discussed in Section 6).

We also note that we can store only the relabelled flow f^μ during the algorithm, rather than f itself. This will remain conveniently integral.

4 ANALYSIS

To prove Theorem 2.1, it suffices to prove it assuming conditions ($\star$) and ($\star\star$). This follows from Section 5; after some initial (inexpensive) preprocessing, Algorithm 1 is run once for feasibility, and then again for optimization.

We first show that Algorithm 1 is correct: if it terminates, it terminates with an optimal solution. To bound the number of arithmetic operations, we first bound the number of operations per augmentation (to be defined momentarily), and then the total number of augmentations.

Finally, we must show that the size of the numbers in the calculations remains polynomially bounded in the input size (in other words, the algorithm runs in PSPACE). This requires minor technical modifications to the algorithm as stated; all of this we delay to Section 6.

Define an *augmentation* to be either a path augmentation, as performed in lines 5–7 in Produce-Plentiful-Node, or what we will call a *null augmentation*: an event when a node $i \in V^-$ for which $\nabla f^\mu_i < b^\mu_i$ at the start of a label update, but $\nabla f^\mu_i \geq b^\mu_i$ after. Unlike path augmentations, they do not modify the solution at all; they are defined purely for accounting purposes. As revealed in the analysis, null augmentations share some important features with path augmentations.

4.1 Correctness

In this section we prove the following.

Theorem 4.1. *If Algorithm 1 terminates, it terminates with an optimal primal-dual pair.*

[2] The same argument would work with $S = \bar{S}$; this selection becomes relevant for numerical stability, discussed in Section 6. It makes no difference for the purposes of this section and the next.

We start by showing some basic properties of Produce-Plentiful-Node.

Lemma 4.2. *The following properties hold in every label update step in Produce-Plentiful-Node.*

(i) f^μ *remains unchanged.*
(ii) $\alpha > 1$ *and finite.*
(iii) (f, μ) *remains a fitting pair.*
(iv) If $i \in (V \setminus S) \cup V^0$, *then* b_i^μ *does not change.*
(v) If $i \in S \cap V^-$, *then* b_i^μ *decreases. After the label update,* $\nabla f_i^\mu \leq b_i^\mu + 2$.
(vi) If $i \in S \cap V^+$, *then* b_i^μ *increases. If* $\alpha = \alpha_i$ *for such an* i, *then* $3n(d_i + 1) \leq b_i^\mu \leq 3n(d_i + 1) + 1$ *after the label update.*

Proof. (i) Consider any arc $ij \in E$. If $i, j \notin S$, then f_{ij} and μ_i are both unchanged; if $i, j \in S$, then f_{ij} and μ_i are both scaled by α. Arcs $ij \in \delta^-(S)$ cannot be tight by the definition of S. For every $ij \in \delta^+(S)$, $f_{ij} = 0$, as otherwise $ji \in E_f$ would be a tight arc entering S. In all cases, f_{ij}^μ is unchanged.

(ii) Since S has no tight incoming arcs and μ is feasible at the start of the label update, certainly $\alpha_0 > 1$; by (1), it is finite, unless $S = V$. Consider any $i \in S \cap V^-$. We have $\nabla f_i^\mu - b_i^\mu < 1$ initially, since the augmentation part of the iteration terminated. Since ∇f_i^μ is unchanged and $b_i < 0$, $\alpha_i > 1$ and finite. Finally, consider any $i \in S \cap V^+$. Since i was not plentiful at the start of the iteration, and $b_i > 0$, it is clear that $\alpha_i > 1$ and finite. Altogether, $\alpha > 1$. Moreover, since α_0 is finite if $S \neq V$, and since $V^+ \cup V^- \neq \emptyset$, α is finite.

(iii) We need to check that $\mu_j \geq \gamma_{ij}\mu_i$ is maintained for all $ij \in E$, and with equality if $f_{ij} > 0$. This is clear for i, j both in S or both outside of S, since μ_i and μ_j are both scaled by the same amount. If only one of $i, j \in S$, then as observed above, $f_{ij} = 0$. If $i \notin S$ and $j \in S$, then the required inequality follows from the definition of α_0 (and that $\alpha \leq \alpha_0$). And if $i \in S$ and $j \notin S$, it follows because $\alpha > 1$.

(iv) This is trivial.

(v) That b_i^μ decreases follows from $\alpha > 1$, and the bound on $\nabla f_i^\mu - b_i^\mu$ from the definition of α_i.

(vi) This follows immediately from $\alpha > 1$ and the definition of α_i. □

It follows immediately from Lemma 4.2(i) that f^μ remains integral throughout all iterations, since the augmentation part clearly maintains integrality. It is also clear that (f, μ) remain always a fitting pair.

Somewhat magically, despite the fact that we make no effort to maintain the feasibility of f, or even keep deficits bounded, safety is preserved.

Lemma 4.3. *The labelling* μ *remains safe throughout the algorithm.*

Proof. At initialization, (f, μ) is a fitting pair and f is a feasible flow, witnessing that μ is safe. Safety is obviously maintained during iterations of Reduce. The nontrivial part is to show that it is also maintained during the label update steps in Produce-Plentiful-Node. Assume μ is safe before a label update, and let μ' denote the updated labels. That is, $\mu'_i = \mu_i$ for $i \notin S$, and $\mu'_i = \mu_i/\alpha$ for $i \in S$, where $\alpha > 1$. We must show that the condition in Lemma 2.3 prevails for μ'_i.

For a contradiction, assume there exists a subset $X \subseteq V \setminus \{t\}$ such that $\delta^-(X) \cap E^{\mu'} = \emptyset$, and $b^{\mu'}(X) > 0$. We call such a set *violated*.

Claim. $f(\delta^-(X \setminus \bar{S})) = 0$.

Proof. Consider an arc ij with $f_{ij} > 0$. Then both ij and ji are in E^μ, and hence also in $E^{\mu'}$. Since X is violated, $ij \notin \delta^-(X)$. And by the definition of $\bar{S}$, $ji \notin \delta^-(\bar{S})$, or equivalently, $ij \notin \delta^+(\bar{S})$. Since $\delta^-(X \setminus \bar{S})) \subseteq \delta^-(X) \cup \delta^+(\bar{S})$, this proves the claim. □

Claim. $b^\mu(X \setminus \bar{S}) > 0$.

Proof. We know that $\delta^-(S) \cap E^\mu = \emptyset$, since by the definition of S it has no incoming tight arcs. Similarly, $\delta^-(\bar{S} \setminus S) \cap E^\mu = \emptyset$. We also have that $E^\mu \cap E[\bar{S}] \subseteq E^{\mu'} \cap E[\bar{S}]$, and so

$$\delta^-(X) \cap E[\bar{S}] \cap E^\mu \subseteq \delta^-(X) \cap E^{\mu'} = \emptyset.$$

Thus $\delta^-(X \cap S) \cap E^\mu = \emptyset$ and $\delta^-(X \cap (\bar{S} \setminus S)) \cap E^\mu = \emptyset$. Since μ is safe, it follows that $b^\mu(X \cap S) \leq 0$ and $b^\mu(X \cap (\bar{S} \setminus S)) \leq 0$. Since X is violated with respect to μ',

$$0 < b^{\mu'}(X) = \tfrac{1}{\alpha} b^\mu(X \cap S) + b^\mu(X \cap (\bar{S} \setminus S)) + b^\mu(X \setminus \bar{S}).$$

Thus $b^\mu(X \setminus \bar{S}) > 0$. □

Now since $\bar{S} \supseteq Q$, $\nabla f_i^\mu \geq b_i^\mu$ for all $i \in X \setminus \bar{S}$. The lemma now follows, since

$$f^\mu(\delta^-(X \setminus \bar{S})) - f^\mu(\delta^+(X \setminus \bar{S})) = \sum_{i \in X \setminus \bar{S}} \nabla f_i^\mu \geq b^\mu(X \setminus \bar{S}) > 0,$$

contradicting the first claim. □

We give a needed bound on Ξ.

Lemma 4.4. $\Xi(f, \mu) < 2n - 1$ *holds throughout the algorithm. Consequently,* $\mathrm{Ex}(f, \mu) < 2n - 1$ *holds throughout.*

Proof. At initialization, we see $\Xi(f, \mu) = 2(n-1)$, since $\nabla f_i^\mu < b_i^\mu + 2$ by (4). We show that $\Xi(f, \mu)$ is non-increasing throughout the algorithm. Consider a call to Produce-Plentiful-Node. A path augmentation may increase ∇f_i^μ only for nodes $i \in Q$, that is, if $\nabla f_i^\mu < b_i^\mu$. Hence, $\Xi(f, \mu)$ may not increase at path augmentations. We claim that no term in $\Xi(f, \mu)$ increases during label update steps. The only change could be b_i^μ. For $i \in V \setminus (S \cap V^-)$, b_i^μ is non-increasing. For $i \in S \cap V^-$, Lemma 4.2(iv) implies that $\nabla f_i^\mu - b_i^\mu \leq 2$ after the label update.

Let us now consider a call to Reduce. The subroutine starts by constructing a flow $g^\mu \in \mathbb{Z}_+^E$ satisfying (6). It follows easily from (6) that $\Xi(g, \mu) \leq \Xi(f, \mu)$. The subroutine returns an image of g under a series of contractions. The proof is complete by showing that $\Xi(g, \mu)$ is non-increasing during contractions. If we contract the arc pq with $t \notin \{p, q\}$, then for the corresponding values g', b', and μ' after the contraction, we have $b'^{\mu'}_q = b_p^\mu + b_q^\mu$, $\nabla g'^{\mu'}_q = \nabla g_p^\mu + \nabla g_q^\mu$. The values for any $i \in V \setminus \{p, q, t\}$ are unchanged. If $t \in \{p, q\}$, then the corresponding term disappears from $\Xi(g, \mu)$. Hence $\Xi(g', \mu') \leq \Xi(g, \mu)$ follows easily. □

We are now ready to prove Theorem 4.1. The safety of the labelling is guaranteed through the entire algorithm by Lemma 4.3. The subroutine Reduce only contracts arcs that are tight in every dual optimal solution μ^* to (D), according to Theorem 3.1, Lemma 4.4, and the safety of the labelling. Lemma 3.4 shows that we can uncontract the final solution to an optimal solution μ^* to (D) in the original instance. Lemma 2.2 shows that the primal solution found by Compute-Primal is optimal to (P).

4.2 Bounding the Work per Augmentation

Theorem 4.5. *If the total number of path augmentations in Algorithm 1 is T, then the algorithm can be implemented in $O((m + n \log n)T)$ arithmetic operations.*

It is easy to see that the time complexity is dominated by the time spent in Produce-Plentiful-Node. So we will focus only on the operations in this routine.

We first give an easy bound of $O(n^2)$ arithmetic operations between two augmentations in Produce-Plentiful-Node. We observe that, between any two augmentations, S can only extend, and it becomes larger at every label update. Indeed, as long as there is no null augmentation, no vertex is removed from Q; and as long as there is no path augmentation, no arc in $E[S]$ is removed from E_f. The subroutine Produce-Plentiful-Node terminates with a plentiful node if $\alpha = \alpha_i$ for a node $i \in S \cap V^+$; and a path augmentation happens once $\alpha = \alpha_i$ for a node $i \in S \cap V^-$. If $\alpha = \alpha_0$, then S is extended by one element. Hence S can be extended at most $O(n)$ times between two augmentations. Each such step can be easily implemented in $O(n)$ time, giving a simple bound of $O(n^2)$ for the steps between two augmentations.

This can be improved by a careful implementation of the algorithm (which is not precisely as written but yields the same path augmentations and eventual label updates). We observe that the label update steps are essentially a multiplicative variant of Dijkstra's algorithm, i.e., for finding highest gain augmenting paths instead of shortest paths. We also have further constraints: for every $i \in V^+ \cup V^-$, there is an upper bound on the time they can spend in the set S of reached nodes. In the description of Produce-Plentiful-Node, we modify all labels μ_i in S every time S is extended; this could result in $O(n^2)$ label modifications. However, it suffices to change the labels μ_i at the end of the label update part. With the use of Fibonacci heaps [7], the subroutine can be implemented in time $O(m + n \log n)$. A formal description of the modified subroutine can be found in the full version of this paper [19].

4.3 Bounding the Number of Augmentations

In this section, we will set up the required potential analysis, and prove a strongly polynomial bound on the number of augmentations. This analysis will be further improved in Section 4.4 to obtain the running time bound needed for Theorem 2.1.

Lemma 4.6. *For any $i \in V^-$, $b_i^\mu - 1 < \nabla f_i^\mu$ holds at any point of the algorithm. Once $b_i^\mu \le \nabla f_i^\mu$ holds in Produce-Plentiful-Node, this property is maintained until the end of the subroutine.*

Proof. When procedure Produce-Plentiful-Node is called, $b_i^\mu - 1 < \nabla f_i^\mu$ holds for every $i \in V \setminus \{t\}$. This true at the initialization, since the input flow was feasible, and the rounding subroutine may decrease ∇f_i^μ by < 1. In every Reduce step, we construct g as a flow satisfying (6), with lower bound $\lfloor b_i^\mu \rfloor \le \nabla g_i^\mu$; this property is preserved throughout the contractions. For the second claim, Lemma 4.2(v) implies that $\nabla f_i^\mu - b_i^\mu$ can only increase for $i \in V^-$ at label updates. Further, if $\nabla f_i^\mu < b_i^\mu + 1$, then no augmentation step will decrease ∇f_i^μ. So once $b_i^\mu \le \nabla f_i^\mu$ holds for $i \in V^-$, this property will be maintained until the end of the procedure. □

We measure progress via the potential

$$\Psi(\mu) := -\sum_{i \in V^-} b_i^\mu .$$

Let us now examine how $\Psi(\mu)$ changes during iterations of Produce-Plentiful-Node.

Lemma 4.7. *During Produce-Plentiful-Node, the potential $\Psi(\mu)$ is increasing. If r augmentations are performed, then $\Psi(\mu)$ increases by at least $\min\{r - 4n, 0\}$.*

Proof. Monotonicity is straightforward. We measure the number of augmentations by another potential:

$$\Phi(f, \mu) := \sum_{i \in V^-} \nabla f_i^\mu - b_i^\mu .$$

Call a path augmentation that begins at a node in V^- and ends at a node in $V \setminus V^-$ a *helpful* augmentation, and all other augmentations (including all null augmentations) *unhelpful*. Lemma 4.6 implies that for every vertex $i \in V^-$, there can be at most one augmentation ending at i. Hence, there can be at most n unhelpful augmentations. Every helpful augmentation decreases $\Phi(f, \mu)$ by one, and hence during the r augmentations, $\Phi(f, \mu)$ decreases by at least $r - n$.

Again by Lemma 4.6, we see that $-n < \Phi(f, \mu)$ throughout. Further, $\Phi(f, \mu) \le \mathrm{Ex}(f, \mu) < 2n$ holds. Therefore, the value of $\Phi(f, \mu)$ must increase by at least $r - 4n$ to counter the decrease caused by helpful augmentations. The value of $\Phi(f, \mu)$ can only increase during label updates, which also increase the value of $\Psi(\mu)$ by the same amount. The proof is complete. □

Lemma 4.8. *$\Psi(\mu) = O(mn)$ throughout the algorithm.*

Proof. We prove that $|b_i^\mu| < 3nd_i + 3n + 3$ for every $i \in V^-$ holds at every iteration. To verify this, note that Produce-Plentiful-Node stops once a plentiful node is found. Hence $|b_i^\mu| \le 3n(d_i + 1)$ before the final label update; by Lemma 4.6, $b_i^\mu - 1 \le \nabla f_i^\mu$ holds. During the final label update, the value $|b_i^\mu|$ can only increase for $i \in S \cap V^-$, and $\nabla f_i^\mu \le b_i^\mu + 2$ after it completes, by Lemma 4.2(v). Hence $|b_i^\mu|$ can increase by at most 3. □

Lemmas 4.7 and 4.8 together imply a bound $O(mn)$ on the number of augmentations in a single execution of Produce-Plentiful-Node, giving a bound of $O(mn^2)$ augmentations throughout the algorithm.

4.4 A Refined Bound on the Number of Augmentations

In this section, we prove the following more refined bound needed for Theorem 2.1.

Theorem 4.9. *There are at most $O(mn\log(n^2/m))$ augmentations throughout the execution of Algorithm 1.*

We first observe what we require in terms of the potential Ψ.

Lemma 4.10. *If the total decrease of Ψ due to contractions over the algorithm is Δ, then the number of augmentations is $\Delta + O(mn)$.*

Proof. Lemma 4.8 shows that $\Psi = O(mn)$. Clearly $\Psi(\mu)$ is always nonnegative, in particular at the start. Thus the total increase in Ψ during iterations of Produce-Plentiful-Node is at most $\Delta + O(mn)$. Lemma 4.7 shows that the number of augmentations in the algorithm is bounded by the total increase Ψ plus $4n^2$ (since there are at most n calls to Produce-Plentiful-Node). The lemma follows. □

Our goal in this subsection is thus to show that the total decrease in Ψ during the algorithm is $O(mn\log(n^2/m))$. At any stage of the algorithm, a node $i \in V$ has a preimage $\Gamma_i \subseteq U$, where U denotes the node set at the start of the algorithm. Define

$$\tau_i := \sum_{j\in\Gamma_i}(3nd_j + 3n + 3)$$

(here, since j is a node in U, d_j is referring to the degree of j in the original graph). Let $M := \sum_{i\in U\setminus\{t\}}\tau_i$; then $M = \Theta(mn)$. Also define $N := 3n^2 + 3$.

Let us assume that the graph does not contain any plentiful nodes at the beginning. Should there be a plentiful node, the first call of Produce-Plentiful-Node is void, and Reduce is called immediately. Clearly after a call to Reduce, we have

$$|b_i^\mu| \le 3n(d_i + 1) + 3 \le \min(\tau_i, N) \qquad \text{for any } i \in V \setminus \{t\}.$$

We use here that we maintain a simple graph by removing parallel arcs created by contractions.

Let V, μ refer to their values before some call to Reduce. Let Π be the partition of V describing the contractions made; so each part of Π is a vertex after Reduce completes. Consider any nontrivial part P of Π; we wish to bound the change in potential associated with P.

First, if $t \in P$, then the decrease in Ψ due to P is not more than $\sum_{i\in P\cap V^-}\tau_i$. Now suppose $t \notin P$. Let us write $b_P^\mu := \sum_{i\in P} b_i^\mu$, which is the node demand of the image of P; note that $|b_P^\mu| \le \tau_P$. The decrease in Ψ due to P is

$$\begin{aligned}&\min\{b_P^\mu, 0\} + \sum_{i\in P\cap V^-}|b_i^\mu| \\ &= \min\Big\{\sum_{i\in P\cap V^+} b_i^\mu, \sum_{i\in P\cap V^-}|b_i^\mu|\Big\} \\ &\le \min\Big\{\sum_{i\in P\cap V^+}\min\{\tau_i, N\}, \sum_{i\in P\cap V^-}\min\{\tau_i, N\}\Big\}.\end{aligned}$$

We may thus bound the total decrease in Ψ throughout the algorithm by the total profit in the following game. We begin with the multiset $W = \{\tau_i : i \in U \setminus \{t\}\}$. The following moves are possible:

- An element $z \in W$ can be removed, yielding a profit of z. (This corresponds to the situation above where $t \in P$; all elements of P are removed.)
- Two disjoint multisets $R_1, R_2 \subseteq W$ can be chosen. All elements in $R_1 \cup R_2$ are removed, and replaced by the single element equal to the sum of all the elements in $R_1 \cup R_2$. This move yields a profit of

$$\min\Big\{\sum_{z\in R_1}\min\{z, N\}, \sum_{z\in R_2}\min\{z, N\}\Big\}.$$

(This corresponds to the situation where $t \notin P$.)

The game ends when W is empty.

Lemma 4.11. *The maximum possible profit in this game is no more than $M(\log(2N|U|/M) + 3)$, where $M = \sum_{z\in W} z$ and the initial size of W is $|U| - 1$.*

Proof. Let W_0 denote the initial multiset. At some later state of the game, for any $y \in S$ let Γ_y denote the multiset of elements of W_0 which have been merged together to form y. (So $y = \sum_{z\in\Gamma_y} z$.)

It is clear that the total profit due to removal moves is at most M. It is exactly M if only a single removal is made right at the end of the game, which we assume from now on.

It is also clear that we may assume that in all merges, $|R_1| = |R_2| = 1$; if either set is larger, we can split the single merge into multiple merges and the profit will only increase. We divide the merges into three types:

(1) merges where both elements are less than N;
(2) merges where one element is at least N, the other less than N; and
(3) merges where both elements are at least N.

We can assume that the merges of type 1 are all done before any merges of type 2 or 3, by reordering moves if necessary. So let W_1 denote the state of the game after all type 1 merges are complete. Note that $y < 2N$ for all $y \in W_1$.

Each $y \in W_1$ with $y < N$ will be involved in exactly one type 2 merge, with a profit of y. So the total profit from type 2 merges is certainly not more than $\sum_{y\in W_1} y = M$.

Let $q = |\{y \in W_1 : y \ge N\}|$. Then there are $q - 1$ merges of type 3. Moreover $q \le \sum_{y\in W_1} y/N = M/N$. So the total profit of type 3 merges is at most $(q-1)N \le M$.

All that remains is to bound the profit of the type 1 merges. We will use the following charging argument. If y, z are merged, we charge the resulting profit $\min(y, z)$ to the elements of Γ_y if $|\Gamma_y| \le |\Gamma_z|$, and to the elements of Γ_z otherwise. When an element $x \in W_0$ is charged, it is charged no more than x. Moreover, for any $y \in W_1$ and any $x \in \Gamma_y$, x cannot be charged to more than $\log|\Gamma_y|$ times. This is because if $x \in \Gamma_y$ is charged to upon merging y and z, then $|\Gamma_{y+z}| \ge 2|\Gamma_y|$. Thus the total profit of type 1 merges is at most

$$\sum_{y\in W_1} y\log|\Gamma_y|.$$

Applying Jensen's inequality,

$$\begin{aligned}\frac{1}{M}\sum_{y\in W_1} y\log|\Gamma_y| &\le \log\Big(\frac{1}{M}\sum_{y\in W_1} y|\Gamma_y|\Big) \\ &\le \log\Big(\frac{1}{M}\sum_{y\in W_1} 2N|\Gamma_y|\Big) \\ &= \log(2N(|U|-1)/M).\end{aligned}$$

(The second inequality exploits that $y < 2N$ for all $y \in W_1$.) Summing the bounds on the profits yields the claim. □

Theorem 4.9 immediately follows.

5 PHASE ONE: FINDING A FEASIBLE SOLUTION

The algorithm described in Section 3 assumes (⋆) on the existence of an initial fitting pair $(\bar{f}, \bar{\mu})$, as well as (⋆⋆) on the existence of an arc from every node to the sink. As explained in Section 2, our algorithm runs in two phases, similarly to the two-phase simplex algorithm. In the first phase, our goal is to find a feasible fitting pair $(\bar{f}, \bar{\mu})$ to the original problem, and the second phase will solve the flow maximization problem. We will also add the additional arcs to satisfy (⋆⋆) in the first phase.

For the feasibility problem, we construct a modified problem instance $\mathcal{I}' = (V', E', t', \gamma', b')$, where (⋆) and (⋆⋆) hold; guaranteeing an initial solution will be straightforward. A pair of optimal primal and dual solutions to the modified problem instance correspond to a fitting pair in the original instance $\mathcal{I} = (V, E, t, \gamma, b)$. The first phase may also terminate concluding that the original instance is infeasible or unbounded.

The modified instance is constructed in two steps. In the first step, we remove some of the original nodes, and construct a feasible solution μ to (D) on the remaining node set as follows.

Step 1: Flooded nodes and feasible labels. Let us call a node $i \in V$ *flooded*, if there exists a flow generating cycle $C \subseteq E$ (that is, $\gamma(C) > 1$), along with a path $P \subseteq E$ connecting a node of C to i. We can use (C, P) to generate arbitrary amounts of excess flow at node i; hence arbitrary demand b_i can be met at a flooded node i. Let $Z \subseteq V$ denote the set of flooded nodes. If $t \in Z$, then the maximum flow amount is unbounded; however, this does not guarantee feasibility by itself, since we also need to satisfy demands of nodes in $V \setminus Z$.

Our algorithm starts by identifying the set Z. A flow generating cycle is a negative cycle with respect to the cost function $c_e = -\log \gamma_e$. Hence we can adapt any negative cycle detection subroutine (e.g. [1, Chapter 5.5]) to find a negative cycle, or conclude the none exists in $O(nm)$ time. We can use a multiplicative adaptation of the cycle detection algorithms to avoid computations with logarithms. If a flow generating cycle C is found, then we include all nodes incident to C into Z, as well as all other nodes that can be reached on a directed path from C. We remove every vertex added to Z from V, and repeat the same process. In $O(n)$ iterations, we correctly identify Z; thus $V \setminus Z$ contains no flow generating cycles. The output of the final cycle detection algorithm on $V \setminus Z$ provides labels $\mu \in \mathbb{R}_{++}^{V \setminus Z}$, such that μ is feasible to (D) restricted to $V \setminus Z$. In particular, one can define

$$\mu_i := \min\{1/\gamma(P) \colon P \text{ is a directed walk starting from } i\}, \quad (7)$$

where for the empty walk $P = \emptyset$ we define $\gamma(P) = 1$. Due to this definition, $\mu_i \leq 1$ for all $i \in V \setminus Z$. Note that Z may or may not contain the sink node t; at this point, we ignore the objective in (D).

Step 2: adding a new sink. Let us now construct the new instance $\mathcal{I}' = (V', E', t', \gamma', b')$ as follows.

$$\begin{aligned} V' &:= (V \setminus Z) \cup \{t'\} \quad \text{for a new sink node } t', \\ E' &:= E[V \setminus Z] \cup \{t'j \colon j \in V^+ \setminus Z\} \cup \{jt' \colon j \in V \setminus Z\}. \end{aligned}$$

We define $b'_i := b_i$ for all $i \in V \setminus (Z \cup \{t\})$, and $b_t := -M$ for a very large $M > 0$ if $t \in V \setminus Z$. We set $\gamma'_e := \gamma_e$ for all $e \in E[V \setminus Z]$. For the new arcs $t'j$, we let $\gamma'_{t'j} := \mu_j \leq 1$, for the labels μ obtained in Step 1.

The new arcs jt' are added in order to satisfy (⋆⋆). We set capacity $\gamma'_{jt'} := \gamma^*$ for all $jt' \in E'$, where γ^* is defined by

$$\Gamma := \max\left\{\gamma_e, \frac{1}{\gamma_e} \colon e \in E[V \setminus Z]\right\}, \qquad \gamma^* := \frac{1}{\Gamma^{n-1} + 1}. \quad (8)$$

Let us now define the initial fitting pair $(\bar{f}', \bar{\mu}')$ required by (⋆). We set $\bar{\mu}'_t := 1$, and $\bar{\mu}'_j := \mu_j$ for all $j \in V \setminus Z$. We let $\bar{f}'_e := 0$ for all arcs $e \in E[V \setminus Z] \cup \delta^-(t')$, and $\bar{f}'_{t'j} := b_j/\mu_j$ for the arcs in $\delta^+(t')$. Note that this flow is feasible, since $\nabla \bar{f}'_j = \max\{b_j, 0\}$ for every $j \in V \setminus Z$. Also, $\bar{f}'$ fits $\bar{\mu}'$, since all arcs in $\delta^+(t')$ are tight.

We now apply Algorithm 1 to the instance $\mathcal{I}'$ with the initial fitting pair $(\bar{f}', \bar{\mu}')$. Let (f', μ') denote the solution returned.

Lemma 5.1. *The original instance $\mathcal{I}$ is feasible if and only if $f'(\delta^+(t')) = 0$. Furthermore, if the instance is feasible and $t \in Z$, then* (P) *is unbounded. If $t \notin Z$, then the objective is bounded.*

Proof. Let us first assume $f'(\delta^+(t')) = 0$. Then f' restricted to $V \setminus Z$ is feasible. The flow generating cycles in Z can provide arbitrary large ∇f_i values for every $i \in Z$. Hence we can extend f' to be feasible in all such nodes. Further, we can achieve an arbitrary large objective value if $t \in Z$. If $t \notin Z$, then the objective value in (D) for μ' in V' gives a finite upper bound on ∇f_t, hence the problem is bounded.

Assume now $f'(\delta^+(t')) > 0$; we show that $\mathcal{I}$ is infeasible. If there is a feasible $\tilde{f}$ on $V \setminus Z$, then adding 0 on all arcs incident to t' yields a feasible solution in $\mathcal{I}'$ with $\nabla \tilde{f}_{t'} = 0$; consequently, the optimum value of (P) for $\mathcal{I}'$ is nonnegative. We let

$$Q := \{j \in V \setminus Z \colon f'_{tj} > 0\}, \qquad W := \{j \in V \setminus Z \colon f'_{jt} > 0\}.$$

We have $Q \neq \emptyset$ by the assumption, which in turn implies $W \neq \emptyset$, because the optimum value is nonnegative. Define $\bar{W} \subseteq V \setminus Z$ as the set of nodes that can be reached from W on a directed path in $E_f[V \setminus Z]$ (including also non-tight arcs.) We claim that $\bar{W} \cap Q = \emptyset$. To show this, assume for a contradiction that there is a path P from a node $j \in W$ to a node $i \in Q$. We claim that the cycle concatenating jt', P, and it' would be a flow generating cycle, a contradiction to the feasibility of μ'. Indeed, $\gamma'_{jt'} = 1/\gamma^* > 1/\gamma'(P)$ by the choice of γ^* in (8), and $\gamma'_{it'} = 1/\gamma'_{t'i} \geq 1$ by construction.

Let $S := V \setminus (Z \cup \bar{W})$. Due to assumption (⋆⋆) ensured by the arcs f_{jt}, all nodes in S have $\nabla {f'}_i^{\mu} = b_i^{\mu}$. Due to the nodes in Q, it follows that $\sum_{i \in S} b_i^{\mu} > 0$. We can thus increase the objective in (D) arbitrarily by setting $\tilde{\mu}_i := \infty$ if $i \in Z \cup \bar{W}$, and $\tilde{\mu}_i = \alpha \mu_i$ for $i \in S$, for any arbitrary large α. □

Hence if $t \in Z$, we terminate by concluding that the objective is unbounded. Otherwise, if the returned μ' is finite, then removing t' and the incident arcs provides a fitting pair on $V \setminus Z$.

In the second phase, we run Algorithm 1 on the following instance $\mathcal{I}''$. Let

$$V'' := V \setminus Z, \qquad E'' := E[V \setminus Z] \cup \{jt : jt \notin E\}.$$

We let $b''_i := b_i$ for all $i \in V''$, $\gamma''_e := \gamma_e$ for all $e \in E[V \setminus Z]$, and $\gamma''_{jt} := \gamma^*$ for the new arcs, with γ^* as in (8). Let us call the new arcs *jt auxiliary arcs*. The initial fitting pair will be the output of phase one, restricted to $V \setminus Z$.

Consider now the optimal solutions (f, μ) returned in the second phase for $\mathcal{I}''$. We need to map them back to optimal solutions (f^*, μ^*) for $\mathcal{I}$. For the primal optimal solution, let us return $f^*_e := f_e$ for all $e \in E[V \setminus Z]$. Inside $E[Z]$, we use the flow generating cycles to satisfy all demands in Z. For the dual optimal solution, if $f_{jt} = 0$ for all auxiliary arcs, we simply return $\mu^*_i := \mu_i$ for all $i \in V \setminus Z$, and $\mu^*_i := \infty$ for $i \in Z$.

Finally, assume $f_{jt} > 0$ on some auxiliary arcs. As in the proof of Lemma 5.1, let $W := \{j \in V \setminus Z : f'_{jt} > 0\}$, and let $\bar{W}$ be the set of all nodes reachable from W on a directed path in $E_f[V \setminus Z]$. The same argument shows that $t \notin W$; also, there are no arcs in E leaving W. We get an optimal dual solution by setting $\mu^*_i := \mu_i$ for all $i \in V \setminus (Z \cup \bar{W})$, and $\mu^*_i := \infty$ for all $i \in Z \cup \bar{W}$.

Remark. The algorithm in [32] used Radzik's [22] strongly polynomial cycle-cancelling subroutine to obtain an initial fitting pair. The argument presented here is also applicable to the algorithm in [32], and thus cycle-cancelling can be avoided. In fact, many arguments in this section have already been used in [32, Section 8].

6 BOUNDING ENCODING LENGTHS

A final step to showing that our algorithm is strongly polynomial is to demonstrate that all numbers appearing during the algorithm have size polynomially bounded in the input size. This was a major challenge in the previous strongly polynomial algorithm [32]. For our algorithm, we will see that this is relatively straightforward.

Consider an instance $\mathcal{I} = (V, E, t, \gamma, b)$, such that $\gamma \in \mathbb{Q}^E_{++}$ and $b \in \mathbb{Z}^V \setminus \{t\}$. Let B be an integer that strictly exceeds both $|b_i|$ and the largest numerator or denominator appearing in any gain factor γ_e. We show that every step of the algorithm can be implemented such that the numbers during the computations remain rational numbers, with numerator and denominator at most $4n^2B^{2n}$.

In order to satisfy ($\star\star$), we add auxiliary arcs jt with $\gamma_{jt} = \gamma^*$ as defined in (8) (see Section 5). Clearly, $\gamma^* \in \mathbb{Q}$, with numerator 1 and denominator at most B^n.

We do not need to work with the flow f directly, but maintain the relabelled flow f^μ instead. This remains integral throughout, except at the very beginning and in the final computation of a primal solution. Moreover, the values f^μ_e are strongly polynomially bounded. If f^μ_e were ever as large as $3n^2$, its endpoints would be plentiful nodes until such time as it was contracted.

Let us now turn to the labels μ. We will maintain the following property.

(*A*) $\mu_i \in \mathbb{Q}_{++}$, $\mu_i \le 2nB^n$, and has denominator $\le 4n^2B^{2n}$ for every $i \in V$.

In order to achieve this, some minor changes in Produce-Plentiful-Node are needed. Let us call a node i an *anchor* if μ_i is an integer multiple of $1/(4n^2)$. In the choice of α_i for $i \in S \cap (V^- \cup V^+)$, we have flexibility in steps 11 and 12 when choosing α_i. Let us always select α_i such that i becomes an anchor. This is always possible, since, as long as a node i is not plentiful, we have $|b^\mu_i| \le 3n(d_i + 1) + 1 \le 4n^2$.

Furthermore, let us only change the flow on augmenting paths starting from anchors; and let us only terminate upon finding a plentiful node if it is an anchor (this happens automatically for plentiful nodes in V^+) which are anchors. For example, if a node $i \in V^-$ enters S with $\nabla f^\mu_i \ge b^\mu_i + 1$, but it is not an anchor, we do not immediately execute the path augmentation in Step 6. Instead, we first move to the label update part, setting α_i such that i becomes an anchor.

We show that with this modification, (*A*) is maintained throughout the algorithm. We need to guarantee this property at initialization; let us postpone this, and first show that if the property already holds, it is maintained in the next step of the algorithm.

The subroutine Reduce trivially maintains (*A*): it only changes the μ_i's by removing some of them. Let us now turn to Produce-Plentiful-Node, and assume (*A*) holds. We show that (*A*) will hold before every path augmentation and at termination. While we do not verify the property at every extension of S, the enhanced variant described at the end of Section 4.2 (see also the full version [19]) only updates the labels at these events.[3]

Since labels may only decrease, the upper bound is trivially maintained. We stop a series of label updates either because a path augmentation is in order, or because a plentiful node is found. According to the above described modification of the algorithm, this means that the set S includes an anchor j. The labels of nodes outside S did not change, hence (*A*) holds for them. Every $i \in S$ has a tight path P_i to the anchor j in $\overleftrightarrow{E^\mu}[S]$. We note that this is where we leverage step 9, i.e., selecting S as an undirected connected component of $\bar{S}$, instead of the more obvious choice $S = \bar{S}$. Then $1 = \gamma^\mu(P) = \gamma(P)\mu_i/\mu_j$, and thus $\mu_i = \mu_j/\gamma(P)$. Now μ_j is an integer multiple of $1/(4n^2)$, and γ_e is rational with numerator at most B for every original arc or reversed original arc. The path P may contain one auxiliary arcs or reversed auxiliary arcs, or one of each type. Recall that γ_e for auxiliary arcs has denominator at most B^n. If there is an auxiliary and a reverse auxiliary arc, their gain factors cancel out. Hence, property (*A*) for i follows.

It remains to show that we can obtain initial labels satisfying (*A*). The initial labels are constructed differently in the two phases. In phase one, we define them as in (7); it is straightforward that all $\mu_i \le 1$, and the μ_i's have denominator at most B^{n-1}. In line 1 of Algorithm 1, they get multiplied by $\Delta = \max_{i \in V \setminus \{t\}} \nabla f^\mu_i - b^\mu_i$. The initial flow f exactly satisfies the demands $i \in V^+$, and has $\nabla f^\mu_i = 0$ if $i \in V^- \cup V^0$. It follows that $\Delta \le B^n$. Consequently, the initial labels in the first phase satisfy (*A*), with the stronger property that $\mu_i \le B^n$ for all $i \in V$. Since the labels are non-increasing, this upper bound is maintained throughout the first phase.

The initial labels for the second phase are obtained from the first phase, and thus satisfy (*A*) with $\mu_i \le B^n$ for all $i \in V$. The value of Δ can be bounded by $\mathrm{Ex}(f, \mu) < 2n$ at this point; hence we have $\mu_i \le 2nB^n$ for all $i \in V$, and therefore (*A*) holds.

Acknowledgements. We are very grateful to José Correa and Andreas Schulz for many interesting discussions which led to this

[3] For the interim updates, a bound of $4n^2B^{3n}$ on the denominators easily follows, assuming (*A*) holds before every path augmentation and at termination.

work. Part of this work was done while the authors were participating in the Hausdorff Trimester on Discrete Mathematics in Fall 2015.

REFERENCES

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

[2] E. Cohen and N. Megiddo. New algorithms for generalized network flows. *Mathematical Programming*, 64(1):325–336, 1994.

[3] S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM Symposium on Theory of Computing (STOC)*, pages 451–460. ACM, 2008.

[4] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.

[5] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

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

[7] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[8] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351, 1991.

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

[10] D. Goldfarb and Z. Jin. A faster combinatorial algorithm for the generalized circulation problem. *Mathematics of Operations Research*, 21(3):529–539, 1996.

[11] D. Goldfarb, Z. Jin, and Y. Lin. A polynomial dual simplex algorithm for the generalized circulation problem. *Mathematical Programming*, 91(2):271–288, 2002.

[12] D. Goldfarb, Z. Jin, and J. B. Orlin. Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 22(4):793–802, 1997.

[13] D. Goldfarb and Y. Lin. Combinatorial interior point methods for generalized network flow problems. *Mathematical Programming*, 93(2):227–246, 2002.

[14] D. S. Hochbaum. Monotonizing linear programs with up to two nonzeroes per column. *Operations Research Letters*, 32(1):49–58, 2004.

[15] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Publication House of the Leningrad State University*, page 68, 1939. English translation in *Management Science* 6(4):366-422, 1960.

[16] S. Kapoor and P. M. Vaidya. Speeding up Karmarkar's algorithm for multicommodity flows. *Mathematical Programming*, 73(1):111–127, 1996.

[17] Y. T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow, part ii. In *55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. IEEE, 2014.

[18] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing*, 12(2):347–353, 1983.

[19] N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. Preprint, arXiv:1611.01778, 2016.

[20] K. Onaga. Dynamic programming of optimum flows in lossy communication nets. *IEEE Transactions on Circuit Theory*, 13:308–327, 1966.

[21] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[22] T. Radzik. Approximate generalized circulation. *Technical Report93-2, Cornell Computational Optimization Project, Cornell University*, 1993.

[23] T. Radzik. Improving time bounds on maximum generalised flow computations by contracting the network. *Theoretical Computer Science*, 312(1):75–97, 2004.

[24] M. Restrepo and D. P. Williamson. A simple GAP-canceling algorithm for the generalized maximum flow problem. *Mathematical Programming*, 118(1):47–74, 2009.

[25] A. Schrijver. *Combinatorial optimization - Polyhedra and Efficiency*. Springer, 2003.

[26] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[27] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, pages 250–256, 1986.

[28] É. Tardos and K. D. Wayne. Simple maximum flow algorithms in lossy networks. In *Proceedings of IPCO, Lecture Notes in Computer Science*, volume 1412, pages 310–324, 1998.

[29] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 332–337. IEEE, 1989.

[30] S. A. Vavasis and Y. Ye. A primal-dual interior point method whose running time depends only on the constraint matrix. *Mathematical Programming*, 74(1):79–120, 1996.

[31] L. A. Végh. Concave generalized flows with applications to market equilibria. *Mathematics of Operations Research*, 39(2):573–596, 2014.

[32] L. A. Végh. A strongly polynomial algorithm for generalized flow maximization. *Mathematics of Operations Research*, 2016. Articles in Advance.

[33] K. D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. *Mathematics of Operations Research*, pages 445–459, 2002.