Strongly Polynomial Algorithms for Generalized Flow Maximization

Francis Cangialosi
frankc@mit.edu

Katie Lewis
kmlewis@mit.edu

David Palmer
drp@mit.edu

1 Introduction

1.1 Problem Definition

Let $G = (V, E)$ be a graph. In the generalized maximum flow problem—as in the ordinary maximum flow problem—the aim is to maximize the total flow delivered to a sink node $t \in V$. The difference is that in the generalized problem, each edge e is endowed with a gain factor $\gamma_e > 0$, which scales the flow passing through that edge. Gains might, for example, represent exchange rates between currencies or dissipation rates of a physical quantity. Setting $\gamma_e \equiv 1$ recovers the ordinary maximum flow problem.

The generalized maximum flow problem lacks several nice properties of the ordinary maximum flow problem, which makes it more challenging to solve. For example, the generalized problem is usually thought to be non-integral and, due to the gain factors, the total supply is not necessarily equal to the total demand. Additionally, the maximum flow–minimum cut theorem no longer applies since the flow along a path is not equal at every edge along that path.

Until recently, the best known algorithms were all weakly polynomial. In 2013, Végh developed the first strongly polynomial algorithm [10]. In 2017, Olver and Végh built on this work and developed an algorithm [4] that is faster than Végh's original algorithm by a factor of almost $O(n^2)$, resulting in a running time that is as fast as the best weakly polynomial algorithms even for small parameter values.

1.2 Structure and Contributions of the Paper

In this paper, we aim to familiarize the reader with recent algorithmic developments on the generalized maximum flow problem and to build intuition for the techniques used to achieve a strongly polynomial result. In Section 2, we formalize the problem as a linear program and give an overview of the notation used in the rest of the paper. In Section 3, we develop the key techniques both algorithms use to achieve a strongly polynomial bound for this problem. In Section 4, we describe Végh's original $O(n^3 m^2 \log n)$ strongly polynomial algorithm developed in 2013 and provide intuition for the running time analysis. In Section 5, we describe Olver and Végh's faster $O((m + n \log n)mn \log(n^2/m))$ algorithm and highlight key aspects of the analysis that led to the improved running time efficiency. Finally, in Section 6, we conclude with a brief summary of the techniques of both algorithms and of how they differ.

2 Preliminaries

2.1 Network Notation

Let $G = (V, E)$ be a directed graph with $n = |V|$ and $m = |E|$ (assume $m \geq n$), $t \in V$ be a sink node and $\gamma \in \mathbb{R}^E_{>0}$ be the vector of gains along edges. We follow recent work on the generalized

problem in realizing flow constraints via node demands $b \in \mathbb{R}^{V \setminus t}$, leaving edge capacities unbounded. (Note that the two formulations are equivalent: *cf.* [10].) Thus, an instance of the generalized flow problem is specified as (G, t, γ, b).

Although there is no explicit specification of a source node, nodes with negative demand b act as sources that can create up to $-b$ units of flow, and nodes with positive demand b act as sinks that can consume at least b units of flow. $V \setminus t$ is comprised of nodes with negative demand, which we denote V^s, nodes with positive demand, V^t, and nodes with zero demand, V^0.

For a subset $S \subseteq V$, we denote by $\delta^{\text{in}}(S)$ and $\delta^{\text{out}}(S)$ the sets of incoming and outgoing edges, respectively. Abusing notation, we also write $\delta^{\text{in}}(i)$ and $\delta^{\text{out}}(i)$ for $\delta^{\text{in}}(\{i\})$ and $\delta^{\text{out}}(\{i\})$. The total degree of a node $i \in v$ is then $d_i = |\delta^{\text{in}}(i) \cup \delta^{\text{out}}(i)|$.

The **net flow** at a node i is the amount of flow that reaches that node (scaled by the gain factor on the incoming edges) minus the amount of flow that leaves the node:

$$\nabla f_i \coloneqq \sum_{e \in \delta^{\text{in}}(i)} \gamma_e f_e - \sum_{e \in \delta^{\text{out}}(i)} f_e.$$

The residual graph $G_f = (V, E_f)$ is defined as in the traditional maximum flow problem, except that the reverse residual edges $(j, i) \in E_f$ have inverted gains $\gamma_{ji} = 1/\gamma_{ij}$ and negated flow $f_{ji} = -\gamma_{ij} f_{ij}$. If the cumulative gain of a cycle C ($\prod_{e \in C} \gamma_e$) in G_f is greater than 1, we call C a "flow-generating cycle." This captures the idea of arbitrage: sending a unit of flow around the cycle generates a net surplus. Such cycles could make the problem unbounded as it is always possible to increase the flow without violating any of the constraints. There is extensive prior work on detecting and eliminating such cycles (*cf.* [10]) so for the remainder of this discussion we assume they do not exist.

2.2 Linear Program Formulation

We formulate the generalized flow maximization problem with node demands as in [4]:

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

The dual linear program, given on the left below, has a variable η_i for every node i. Making the change of variables $\eta_i = -\mu_t/\mu_i$ yields the dual optimization problem (used in [4]) on the right.

$$\begin{aligned} \min \quad & \sum_{i \in V \setminus t} b_i \eta_i \\ \text{s.t.} \quad & \gamma_{ij}\eta_j - \eta_i \geq 0 && \forall\, (i,j) \in E \\ & && i, j \neq t \\ & \gamma_{ti}\eta_i \geq -1 && \forall\, (i,t) \in E \\ & -\eta_i \geq \gamma_{it} && \forall\, (t,i) \in E \\ & \eta_i \leq 0 && \forall\, i \in V \setminus t \end{aligned} \quad \xrightarrow{\eta_i = -\mu_t/\mu_i} \quad \begin{aligned} \max \quad & \mu_t \sum_{i \in V \setminus t} \frac{b_i}{\mu_i} \\ \text{s.t.} \quad & \gamma_{ij}\mu_i \leq \mu_j && \forall\, (i,j) \in E \\ & \mu_i \in \mathbb{R}_{>0} \cup \infty && \forall\, i \in V \setminus t \\ & \mu_t \in \mathbb{R}_{>0} \end{aligned} \tag{D}$$

A dual solution μ is known as a *labeling*, and we can speak of feasible and optimal labelings. We make the simplifying assumption that the sink is reachable from every node, which ensures μ will be finite. (See [4] for a simple transformation that allows us to guarantee this assumption.) Imagine that the nodes represent currencies and the gains represent exchange rates between those currencies. A maximum generalized flow is a currency trading strategy that maximizes the amount of currency t at the end. We can interpret a label μ_i as the value of one dollar in currency i. Then the dual constraint $\gamma_{ij}\mu_i \leq \mu_j$ means that the valuation does not permit arbitrage.

Complementary slackness requires that for an optimal flow f^* and optimal labeling μ^*,

$$f^*_{ij} > 0 \implies \gamma_{ij}\mu^*_i = \mu^*_j. \tag{CS}$$

In our currency trading interpretation, the complementary slackness condition means that the optimal currency trading strategy only makes trades that exactly preserve value.

Given a feasible dual solution μ, we can apply the following **relabeling** transformation to obtain a new problem instance with gains γ^μ and demands b^μ and a primal solution to the new instance:

$$\gamma^\mu_{ij} \coloneqq \gamma_{ij}\frac{\mu_i}{\mu_j} \quad b^\mu_i \coloneqq \frac{b_i}{\mu_i} \quad f^\mu_{ij} \coloneqq \frac{f_{ij}}{\mu_i} \quad \nabla f^\mu_i \coloneqq \frac{\nabla f_i}{\mu_i} \tag{RE}$$

The relabeled dual problem is equivalent to the original dual problem. In particular,

Claim 2.1. *f^μ is a feasible (resp. optimal) solution to the relabeled primal problem with gains γ^μ and demands b^μ if and only if f is a feasible (resp. optimal) solution to the original problem with gains γ and demands b.*

Claim 2.2. *If μ is a feasible solution to the original dual problem and $\bar{\mu}^*$ is an optimal solution to the relabeled dual problem with gains γ^μ and demands b^μ, then the vertex-wise product $\mu\bar{\mu}^*$ is an optimal solution to the original dual problem.*

Thus, the algorithms will use relabeling to simplify the problem while preserving its structure.

Definition 2.1. An edge e is known as **tight** with respect to μ if $\gamma^\mu_e = 1$. For an optimal pair (f, μ), complementary slackness (CS) is exactly the condition that flow only flows along tight edges. We denote the set of tight edges under μ by

$$\tau(\mu) = \{e \mid \gamma^\mu_e = 1\}.$$

The **tight graph** consists of these edges and their endpoints.

Definition 2.2. The **excess** of a flow f at a node i is the net flow left there after subtracting demand, $\nabla f_i - b_i$. When this quantity is negative, it is described as **deficit**. Given f and μ, we can define relabeled excess at a node similarly as $\nabla f^\mu_i - b^\mu_i$.

Given flow values and labels, the **total excess and deficit** of the flow with respect to the labels are defined as follows:

$$\mathrm{Ex}(f,\mu) \coloneqq \sum_{i \in V \setminus t} \max\{\nabla f^\mu_i - b^\mu_i, 0\} \qquad \mathrm{Def}(f,\mu) \coloneqq \sum_{i \in V \setminus t} \max\{b^\mu_i - \nabla f^\mu_i, 0\}$$

3 Motivating Ideas and Techniques

3.1 Solving the Problem Combinatorially

A weakly polynomial algorithm is easily achievable by applying ellipsoid or interior point methods to the linear program (P). To achieve a strongly polynomial bound, the linear program can be reduced to a combinatorial problem.

Suppose an edge (i, j) is known to be tight for any dual-optimal solution μ. This imposes a constraint $\gamma_{ij}^{\mu} = 1$ for all such μ, or, i.e., $\gamma_{ij}\mu_i = \mu_j$. A tight constraint of this form reduces the dimension of the solution space by one and is equivalent to removing one variable from optimization. We might as well remove one of the variables entirely by collapsing the edge (i, j), merging node i into node j with a single dual variable μ_j, and summing the demands b_i and b_j. (When either i or j is t, the other's demand is simply absorbed.) This operation is called an **edge contraction**, and it preserves the dual objective and the constraints on edges other than (i, j). An edge e which is tight under any dual-optimal solution is accordingly called **contractible**.

An algorithm based on edge contractions identifies one contractible edge at a time and contracts it. (As an additional simplification, if the contraction leads to parallel edges, only the one with the highest gain is maintained—there is no reason to send flow along an edge of lower gain over a parallel one of higher gain.) There can be at most $n - 1$ contractions before the graph is left with a single node. The set of guaranteed-tight edges identified over the course of the contractions is acyclic, and it takes the form of either a spanning tree or a spanning forest depending on which of the following termination conditions occurs:

(i) The contracted graph has just one node remaining, i.e., every node has been merged into the sink t. In this case, the set of previously identified contractible edges forms a spanning tree. Given a value for μ_t, the tightness constraints then uniquely determine the values of the optimal μ at all other nodes.

(ii) The contracted graph has multiple nodes, but the (merged, relabeled) demands b_i^{μ} are all zero. In this case, the dual objective on the contracted graph is identically zero. Meanwhile $f \equiv 0$ is primal-feasible on the contracted graph, with a primal objective value of zero. By strong duality, $f \equiv 0$ is optimal.

In this case, the set of previously identified contractible edges forms a spanning forest wherein each tree component is the preimage of one node of the contracted graph. Each component of the tight forest has net demand zero under a suitable relabeling. As such, the flows in separate components are independent, effectively reducing the problem to case (i) for each component. The maximum flow value only depends on the flow in the tree component of t.

Once a set of tight edges has been identified and used to compute dual labels, complementary slackness dictates that any optimal flow f is supported only on the tight edges—i.e., $\gamma_{ij}^{\mu} = 1$ whenever $f_{ij} > 0$. As such, the conservation constraint for the relabeled optimum flow f^{μ} takes the form

$$b_i^{\mu} \leq \nabla f_i^{\mu} = \sum_{(j,i)\in\delta^{\text{in}}(i)} \gamma_{ji}^{\mu} f_{ji}^{\mu} - \sum_{(i,j)\in\delta^{\text{out}}(i)} f_{ij}^{\mu} = \sum_{(j,i)\in\delta^{\text{in}}(i)} f_{ji}^{\mu} - \sum_{(i,j)\in\delta^{\text{out}}(i)} f_{ij}^{\mu}.$$

This is the conservation constraint of an ordinary maximum flow problem with demands b^{μ}. Indeed, one can compute the optimal f^{μ} as an ordinary maximum flow on the tight graph—a strongly polynomial computation. Then it is easy to recover f from f^{μ} (by undoing the scaling in (RE)). Thus, we have shown how to reduce the generalized flow problem to a combinatorial problem—namely, computing an optimal set of tight edges.

Example 3.1. To help visualize the connection between tight edges and optimal solutions, consider the flow network on the left in Figure 3.1. Nodes i and j both have negative demand and can thus act as sources for up to 8 units of flow. By inspection, we can see that the optimal f^* should send 8 units of flow from i to t and 8 units from j to t, yielding 6 units at the sink. On the right, we show the three possible spanning trees of tight edges for this graph. For each tree, starting with $\mu_t = 1$, we can compute μ_i and μ_j based on the tight edges. In (a), the dual is not feasible because $\gamma_{it}\mu_i = 4 \not\leq 1 = \mu_t$. In (b) and (c), all of the dual constraints are feasible, but (c) produces a greater dual objective value (-6) than (b) (-12). (Note that the sign is negative because of our change of variables in (D)). Solving a max flow on just the tight edges of (c) yields the optimal ∇f_t^* we expect: 6. The optimal collection of tight edges encodes all the information necessary to reconstruct the optimal solution.

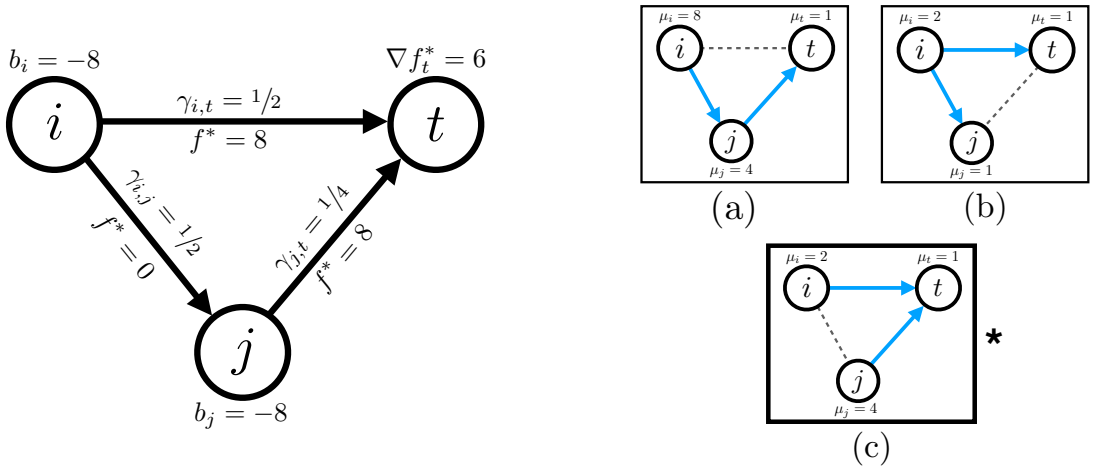

Figure 1: (left) Example flow network. (right) All three possible spanning trees of tight edges and their respective dual solutions. Blue edges are tight ($\gamma^\mu = 1$), gray dashed edges are not.

3.2 Contractibility Certificates

The heart of both strongly polynomial algorithms detailed below is a procedure for identifying contractible edges and then contracting them. The key differences between the two algorithms are in *how* they find contractible edges, which we will describe in Sections 4 and 5. These differences allow Olver and Végh [4] to achieve a better running time bound than Végh [10]. The algorithms have in common the use of primal-dual pairs as certificates for contractibility.

From Section 3.1, complementary slackness tells us that if there is a primal optimal flow f with positive value on some edge e, then e is tight with respect to *every* dual optimal solution—that is, e is contractible. So a primal optimal flow provides a certificate for contractibility. It would seem this is of no use to us, as computing a primal optimal flow is the entire problem we are trying to solve.

However, an optimal flow is a vastly superfluous certificate. Suppose instead that we have any flow g and a bound on the difference between g and an optimal flow g^*. Then if g had a sufficiently large value on some edge e, we would know that $g_e^* > 0$. This idea is formalized in the following lemma, adapted from Lemma 3.3 of [4].

Definition 3.1. Let f be a feasible solution to (P). μ is a **conservative labeling** for f if μ is a feasible solution to (D) and all edges with positive flow are tight. In this case, (f, μ) is called a **conservative pair**.

Lemma 3.1. *Suppose (g, μ) is a conservative pair. Then there is some optimal flow g^* uniformly "close" to g:*

$$\|(g^*)^\mu - g^\mu\|_\infty \leq \mathrm{Ex}(g, \mu).$$

Proof. The intuition is that the optimal flow improves on the feasible flow by sending more of its excess to the sink. Thus, the difference $g^* - g$ is a flow of value at most $\mathrm{Ex}(g, \mu)$ at t. The maximum edge value of the difference is bounded, in turn, by its total flow value.

Since we are working with generalized flows, it will take a little work to formalize this idea. First, among all the optimal solutions to (P), let g^* be the one which minimizes the total (L^1) difference $\|g^* - g\|_1$. Let μ^* be an optimal solution to (D). Then (g^*, μ^*) are a conservative pair—this is precisely the complementary slackness condition they satisfy.

Let $h = g^* - g$. That is,

$$h_{ij} \coloneqq \begin{cases} g^*_{ij} - g_{ij} & \text{if } (i,j) \in E,\ g^*_{ij} > g_{ij} \\ \gamma_{ij}(g_{ij} - g^*_{ij}) & \text{if } (j,i) \in E,\ g_{ji} > g^*_{ji} \end{cases}$$

Observe that if $h_{ij} > 0$, then either $(i,j) \in E$ and $g^*_{ij} > 0$ or $(j,i) \in E$ and $g_{ji} > 0$. In other words, $(i,j) \in E_g$ and $(j,i) \in E_{g^*}$.

Claim 3.2. *h is acyclic.*

Proof of Claim. Suppose that h contains a cycle C. Then consider the net gain around C:

$$1 \geq \gamma^\mu(C) = \gamma(C) = \gamma^{\mu^*}(C) \geq 1,$$

as $\gamma^\mu \leq 1$ on E_g and $\gamma^{\mu^*} \leq 1$ on E_{g^*}. As $\gamma(C) = 1$, we can slightly modify g^* over the whole cycle to decrease $\|h\|_1$, contradicting the choice of g^* to minimize $\|h\|_1$. □

As h is acyclic, it may be decomposed into paths. Because h is supported in E_g and $\gamma^\mu \leq 1$ on E_g, flow can only decrease along these paths. As such, the total flow along any edge is at most the total flow leaving nodes:

$$\|h^\mu\|_\infty \leq \sum_i \max\{0, -\nabla h^\mu_i\} = \sum_i \max\{0, \nabla g^\mu_i - \nabla (g^*)^\mu_i\} \leq \sum_i \max\{0, \nabla g^\mu_i - b^\mu_i\},$$

which is $\mathrm{Ex}(g, \mu)$ as desired. □

The lemma suggests a concrete certificate for contractibility. An edge e is **abundant** with respect to a conservative pair (f, μ) if $f^\mu_e > \mathrm{Ex}(f, \mu)$. Lemma 3.1 shows that abundant edges are contractible.

Any algorithm seeking to find an abundant edge has a dual mandate—(i) bounding the relabeled excess and (ii) obtaining a large relabeled flow value on an edge. Both goals depend on both the primal and dual variables. We can obtain a more useful contractibility certificate by disentangling the two. In particular, both algorithms detailed below employ a node-wise contractibility certificate in which the relevant quantity at a node i is the relabeled demand b^μ_i. A node with a very large value of $|b^\mu_i|$ is known as a **plentiful node**. The specific definition of plentiful node differs between the two algorithms (*cf.* Sections 4.2 and 5.2), but in both cases it is constructed to guarantee that a plentiful node borders an abundant edge.

Aiming to produce a plentiful node as a contractibility certificate, both algorithms consist of alternating *scaling* and *augmentation* phases. The scaling phases adjust the dual variables μ to increase the values $|b^\mu_i|$. The complementary augmentation phases spread or cancel excesses to keep them bounded.

4 The Initial Strongly Polynomial Algorithm

A high-level overview of Végh's initial algorithm [10] is shown below. Progress is parameterized by a scaling factor Δ, which controls the amount of flow that can be augmented at each iteration and the threshold for determining which edges are contractible. Sections 4.1 and 4.2 expand on the details and provide intuition. Section 4.3 sketches the runtime analysis of the algorithm, showing how it achieves a strongly polynomial bound.

(1) Find an initial feasible flow $\bar{f}$ and set initial Δ to the max relabeled excess of any node.

(2) Compute an initial conservative labeling, μ, and find initial conservative pair (f, μ).

(3) Find contractible edges and contract until stopping condition is met (*cf.* Section 3.1). Contractible edges are found by repeatedly applying one of the following operations:

   (a) Augment Δ units of flow.
   (b) Increase μ and decrease Δ by a factor $\alpha > 1$.
   (c) Send flow from isolated parts of the graph directly to the sink.

(4) Compute optimal primal solution from set of tight edges (*cf.* Section 3.1).

4.1 Finding and Maintaining Conservative Pairs

Following from primal-dual slackness, the optimality conditions for the primal are (i) $\nabla f_i^\mu - b_i^\mu = 0$ for all $i \in V \setminus t$ and (ii) $\gamma_{ij}^\mu = 1$ if $f_{ij} > 0$. The first condition is achieved by incrementally reducing a uniform upper bound on node excess. The second condition is achieved by approximating a conservative pair (Definition 3.1) with increasing fidelity over the course of the algorithm.

To find an initial conservative pair (f, μ), we first need a feasible flow. $\bar{f} \equiv 0$ is a trivial feasible solution to the capacitated version of the primal problem; this can be transformed into an initial feasible solution f for our uncapacitated problem by following the transformation in [1] (linear in the number of edges). Next, we need a feasible μ. Starting with $\mu_t := 1$, we set μ_i to the maximum value of $1/\gamma_e$ for e on any path from i to t. This can be efficiently computed by using a multiplicative version of Dijkstra's algorithm (or equivalently, using Dijkstra's algorithm with edge costs $-\log \gamma_e$—see Section 5.3 below). Finally, we modify f to restrict flow to tight edges under μ. This requires computing a standard maximum flow on a modified graph (with lower/upper bounds and an added source) and then zeroing flow on non-tight edges.

Armed with an initial conservative pair, we set the initial value of Δ to $\max_{i \in V \setminus T} \nabla f_i^\mu - b_i^\mu$, the maximum relabeled excess. This value is chosen because Δ controls the amount of flow sent along each augmenting path, which can be no more than the maximum excess. Δ is monotonically decreasing from here on.

The algorithm maintains a relaxed **Δ-conservative pair**, which allows small amounts of flow (less than Δ units) along non-tight edges. The excess at each node is required to be at least the amount of incoming flow along non-tight edges. Thus, at any time, the flow can be made conservative (while remaining feasible) by zeroing flow along non-tight edges. As Δ decreases, the amount of flow on these non-tight edges decreases so that the Δ-conservative pair approaches a conservative pair.

4.2 Identifying Contractible Edges

As discussed in Section 3.1, the algorithm makes progress by finding contractible edges. As discussed in Section 3.2, an edge is guaranteed to be contractible when it is abundant. The algorithm maintains

a pair (f, μ) with a uniform bound on the excess $\nabla f_i^\mu - b_i^\mu \le (4d_i + 8)\Delta$, whence $\mathrm{Ex}(f, \mu) \le 16m\Delta$. Thus, an edge (i, j) with $f_{ij}^\mu \ge 17m\Delta$ is abundant. As long as this bound is maintained, the following plentiful node certificate, which depends only on the dual variables μ, guarantees the existence of an adjacent contractible edge:

Definition 4.1. A node i is **plentiful** if $\left|b_i^\mu\right| \ge 20nm\Delta$.

Claim 4.1. *If i is a plentiful node, then there is a contractible edge adjacent to it.*

Proof. Suppose by way of contradiction that none of the edges adjacent to i are contractible; this means that each edge e must have $f_e^\mu < 17m\Delta$. Consider the two cases below and note that $m \ge n \ge d_i + 1$ (2.1).

1. First suppose $b_i^\mu > 0$. Then we obtain a contradiction to the feasibility of f:

$$\nabla f_i^\mu - b_i^\mu = \sum_{j \in \delta^{\mathrm{in}}(i)} \gamma_{ji}^\mu f_{ji}^\mu - \sum_{j \in \delta^{\mathrm{out}}(i)} f_{ij}^\mu - b_i^\mu < 17 d_i m\Delta - 20nm\Delta < 0.$$

2. On the other hand, if $b_i^\mu < 0$, then we obtain a contradiction to the bound $\nabla f_i^\mu - b_i^\mu \le (4d_i + 8)\Delta$:

$$\nabla f_i^\mu - b_i^\mu = \sum_{j \in \delta^{\mathrm{in}}(i)} \gamma_{ji}^\mu f_{ji}^\mu - \sum_{j \in \delta^{\mathrm{out}}(i)} f_{ij}^\mu - b_i^\mu > -17 d_i m\Delta + 20nm\Delta \ge 3nm\Delta + 17m\Delta. \quad \square$$

The algorithm maintains several sets of nodes. Let L be the set of "low excess" nodes with $\nabla f_i^\mu - b_i^\mu < (d_i + 1)\Delta$. Let H be the set of "high excess" nodes. A node enters H when its excess hits the upper bound $4(d_i + 2)\Delta$, and it stays in H until its excess falls below $(d_i + 2)\Delta$. These bounds are illustrated in Figure 2.

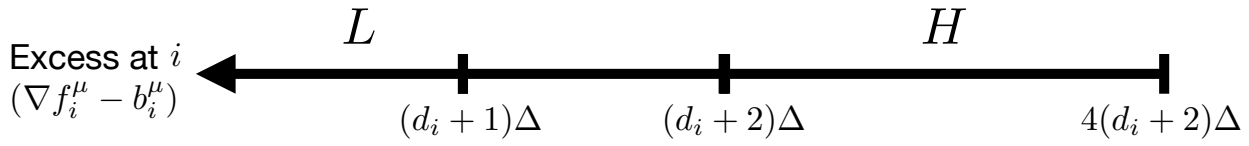

Figure 2: Number line showing values of excess when node i is in set L or H (*not to scale*).

Finally, let R be the set of nodes reachable along tight paths from H. (In particular, $H \subseteq R$.) The algorithm generally seeks to spread excess from H to L along tight paths:

- **(3)a Augmentation:** Each iteration of step (3) first looks to augment Δ units of flow on tight paths from H to $R \cap L$. This redistributes excess in order to improve the balance of excess across the graph. When the algorithm can no longer augment, there must either be (i) no reachable low excess nodes to augment to or (ii) no high excess nodes to push flow from.
- **(3)b Scaling:** The scaling phase resolves both potential problems by increasing μ in R and decreasing Δ by the same factor $\alpha > 1$.
  - (i) Increasing μ in R causes γ_{ij}^μ to increase for edges from R to $V \setminus R$. This can cause an edge crossing this cut to become tight, growing R. In particular, this may make a low-excess node reachable.
  - (ii) Δ can decrease until some node i has excess equal to the upper bound $4(d_i + 2)\Delta$. At this point i has become a high-excess node.

Claim 4.2. *The ratio $|b_i^\mu|/\Delta$ is non-decreasing during scaling and increases for nodes $i \in V \setminus R$. (Note: it is also constant during the other phases.)*

Proof. $\alpha > 1$ by construction. The ratio does not change for nodes in R since both $b_i^\mu = b_i/\mu_i$ and Δ are divided by α. In $V \setminus R$, the ratio is multiplied by α because Δ is divided by α. □

- **(3)c De-isolation:** If $V \setminus R$ does not contain any nodes with large values of $|b_i^\mu|/\Delta$, then scaling will be ineffective in producing a plentiful node. As such, if $|b_i^\mu|/\Delta < 1/n$ for every node in $V \setminus R$, we delay scaling and focus on pushing flow directly to the sink from "isolated" nodes (i.e. those that are not reachable along a tight path from H). This goal is achieved by computing a max flow along tight edges in $V \setminus R$, which replaces the current flow on edges inside $V \setminus R$; flow on edges going from $V \setminus R$ to R is zeroed, and all other values remain the same. Reducing excess of nodes in $V \setminus R$ potentially allows the next scaling phase to last longer and thus increase $|b_i^\mu|/\Delta$ more. Simultaneously, zeroing the flow from $V \setminus R$ to R reduces the excess of nodes in R, which might open them up to accept augmentations in a future iteration.

4.3 Sketch of Runtime Analysis

In this section we analyze a few key ideas that lead to a strongly polynomial runtime. The full (and lengthy) set of proofs for the runtime analysis can be found in [1].

As noted in Section 3.1, there can only be $n-1$ contractions. Therefore the analysis aims to show a contractible edge can be found in strongly polynomial time. We define the following potential function to bound the number of iterations of step (3) defined at the beginning of Section 4.

$$\Psi := \sum_{i \in H} \left\lfloor \frac{\nabla f_i^\mu - b_i^\mu}{\Delta} - (d_i + 1) \right\rfloor$$

Recall that H denotes the set of "high excess" nodes. The potential increases by $4(d_i+2) - (d_i+1) = 3d_i + 7$ when a node is added to H (this is called an *expanding iteration*) and decreases by $(d_i+2) - (d_i+1) = 1$ when a node leaves H (a *shrinking iteration*). Once a node leaves H, Δ must decrease by at least a factor of two before that node can enter H again; this allows us to bound the total number of expanding iterations. Since H is initially empty ($\Psi = 0$), we can use the bound on expanding iterations to bound the total increase in potential. Furthermore, since a node in H must have $\nabla f_i^\mu - b_i^\mu \geq \Delta(d_i + 2)$, Ψ is always nonnegative. This allows us to use the total increase in potential to bound the total decrease in potential, and, in turn, the number of shrinking iterations.

This yields strongly polynomial bounds on the number of expanding and shrinking iterations; however, we must also prove that there are only a few "neutral" iterations in between the expanding and shrinking iterations. This is where we note the importance of the De-Isolation subroutine. For this analysis, let D be the set of nodes with $|b_i^\mu|/\Delta \geq 1/n$. On each iteration, if there is not an augmentation (shrinking iteration), then either Scaling alone or De-Isolation and Scaling will be called. If Scaling is called, then $V \setminus R \cap D$ contained at least one node i, which will have its $|b_i^\mu|/\Delta$ increased by α. Since $i \in D$, $|b_i^\mu|/\Delta \geq 1/n$, so i will become plentiful after $O(\log(n^2 m)) = O(\log n)$ halvings of Δ. This bounds the number of pure scalings in terms of the number of expanding iterations. If De-Isolation is called first, then the excess of nodes in $V \setminus R$ will decrease enough such that the next Scaling round will either grow R, allowing an augmentation (shrinking iteration) on the next iteration, or grow D. Note that the latter can only happen $n-1$ times because once a node enters D, it cannot leave ($|b_i^\mu|/\Delta$ is non-decreasing) unless it is contracted.

The analysis above gives us a strongly polynomial bound on the total number of iterations. It takes at most $O(mn \log n)$ iterations to find a contractible edge, and there are at most $O(n)$ iterations (due to the stopping condition). The running time per iteration is dominated by De-Isolation, which takes $O(nm)$ time. Therefore, this gives an upper bound of $O(n^3 m^2 \log n)$ for the total runtime (all other operations including initialization are absorbed into this runtime).

5 A Faster Strongly Polynomial Algorithm

The most recent strongly polynomial algorithm developed by Olver and Végh [4] takes the same overall approach as the original algorithm [10], but introduces several novel ideas that improve the running time by a factor of almost $O(n^2)$. In this section, we describe the algorithm and provide intuition for the improved running time.

5.1 Working with Infeasible Flows

The algorithm described in Section 4 maintains a feasible flow at all times. In order to produce a contractibility certificate, it has to keep the excess at each node bounded. In order to do that, it has to spread flow around, leveling out the excesses. In contrast, Olver and Végh [4] maintain a possibly infeasible flow, introducing the possibility of deficit at a node. Rather than spreading excesses around to level them, this algorithm sends them to cancel with deficits. Whereas leveling excesses could require augmenting between all pairs of nodes (roughly n^2 pairs to consider), cancelling excess only requires identifying a corresponding deficit for each excess (roughly $O(n)$). This gives some intuition for why relaxing the feasibility requirement might be appealing.

More precisely, the algorithm maintains a **fitting pair** of primal and dual solutions (f, μ). This is a relaxation of the idea of a conservative pair from Section 3.2, wherein f is not required to be feasible:

Definition 5.1. A (not necessarily feasible) flow f is said to **fit** a feasible dual solution μ if it satisfies the putative complementary slackness condition: $f_{ij} > 0 \implies \gamma_{ij}^{\mu} = 1$. In this case, we refer to (f, μ) as a **fitting pair**.

In lieu of maintaining a feasible f at each step, the algorithm ensures that μ is **safe**—i.e., that a feasible f fitting μ *exists*—by preserving a cut property. To simplify the exposition, we start with a definition.

Definition 5.2. Recall that $\tau(\mu)$ denotes the set of tight edges under a dual labeling μ. Given μ, an **inward-loose cut** is a set $S \subseteq V$ that does not have any tight incoming edges ($\delta^{\text{in}}(S) \cap \tau(\mu) = \varnothing$).

If f is a flow fitting μ, then there is no flow into inward-loose cuts because there are no tight edges to carry such flow. If f is feasible, then it follows that the total demand inside an inward-loose cut is nonpositive—otherwise f would have to satisfy this demand by flowing into the cut. It turns out that this is a sufficient condition for the existence of a feasible fitting flow:

Lemma 5.1 (Certificate for Safety)**.** *μ is safe if and only if for any inward-loose cut $S \subseteq V \setminus t$,*

$$\sum_{i \in S} b_i^{\mu} \leq 0.$$

This is a corollary of Hoffman's Circulation Theorem ([7], Theorem 11.2), a standard result on existence of (ordinary, not generalized) flows, applied to the tight graph under μ.

5.2 A Refined Contractibility Certificate

So long as the algorithm maintains the safety of μ, we can obtain a feasible fitting flow f at any time by running an ordinary maximum flow algorithm restricted to the tight graph of μ. Moreover, a generalization of Lemma 3.1 applies to fitting pairs:

Lemma 5.2. *Suppose (g, μ) is a fitting pair with μ safe. Then there is some feasible flow $\tilde{g}$ uniformly "close" to g:*

$$\|\tilde{g}^\mu - g^\mu\|_\infty \leq \mathrm{Def}(g, \mu) \quad \textit{and} \quad \mathrm{Ex}(\tilde{g}, \mu) \leq \mathrm{Ex}(g, \mu).$$

Proof Idea. The proof is very similar to that of Lemma 3.1. However, as both f and $\tilde{f}$ fit the same dual μ, the analysis need only consider f^μ and $\tilde{f}^\mu$ as regular flows on $\tau(\mu)$. This simplifies the argument considerably. We omit the full proof for brevity. □

Corollary 5.2.1. *Suppose (g, μ) is a fitting pair with μ safe. Then there is some optimal flow g^* uniformly "close" to g:*

$$\|(g^*)^\mu - g^\mu\|_\infty \leq \mathrm{Ex}(g, \mu) + \mathrm{Def}(g, \mu).$$

Proof. Follows immediately from Lemma 5.2, Lemma 3.1, and the triangle inequality. □

Corollary 5.2.1 suggests an appropriate definition of abundant edge when working with infeasible flows: if (g, μ) is a fitting pair with μ safe and $g_e^\mu > \mathrm{Ex}(g, \mu) + \mathrm{Def}(g, \mu)$, then e is contractible. As in Section 4.2, there is a corresponding node-wise certificate depending only on the dual variables:

Definition 5.3. A node i is **plentiful** with respect to the current primal-dual solution pair (f, μ) if $|b_i^\mu| \geq 3n(d_i + 1)$.

Theorem 5.3. *Let (f, μ) be a fitting pair such that*

(i) μ is safe.

(ii) Relabeled excess is uniformly bounded: $\nabla f_i^\mu - b_i^\mu \leq 2$ everywhere.

Let i be a plentiful node with respect to (f, μ). Then a contractible edge neighboring i can be found in strongly polynomial time.

Proof. First we compute a flow with bounded excess and deficit. Then we use it to obtain an abundant edge.

Since μ is safe, there is some flow f' satisfying $b_i^\mu \leq (f')_i^\mu$ for all i and $f'_e > 0$ only if e is tight. Meanwhile, $\nabla f_i^\mu \leq b_i^\mu + 2$. By an ordinary maximum flow computation on the tight edges defined by μ (*cf.* [7], Corollary 11.2), it is possible to compute a flow g which is not necessarily feasible but satisfies

$$\lfloor b_i^\mu \rfloor \leq b_i^\mu \leq \nabla(f')_i^\mu \leq \nabla g_i^\mu \leq \nabla f_i^\mu \leq \max\{\nabla f_i^\mu, \lceil b_i^\mu \rceil\}.$$

Clearly, $\mathrm{Ex}(g, \mu) \leq 2n$ and $\mathrm{Def}(g, \mu) \leq n - 1$. (Note that the sink t is excluded from total excess and deficit.) Moreover, if f is integral, then the outer bounds above are integral; so g can be made integral.

Suppose that $i \in V^s$. (The other case is similar.) We have

$$\sum_{e \in \delta^{\mathrm{in}}(i)} g_e^\mu \geq \sum_{e \in \delta^{\mathrm{in}}(i)} g_e^\mu - \sum_{e \in \delta^{\mathrm{out}}(i)} g_e^\mu = -\nabla g_i^\mu \geq -b_i^\mu - \mathrm{Ex}(g, \mu),$$

where the last inequality is because $\nabla g_i^\mu - b_i^\mu$ appears as a term in $\mathrm{Ex}(g, \mu)$ if it is positive. Now

$$-b_i^\mu - \mathrm{Ex}(g, \mu) > 3n(d_i + 1) - 2n > 3nd_i > 3n|\delta^{\mathrm{in}}(i)|$$

So there is some edge $e \in \delta^{\text{in}}(i)$ with $g_e^\mu > 3n > \text{Ex}(g,\mu) + \text{Def}(g,\mu)$. This is an abundant and contractible edge. □

In practice, the algorithm maintains a fitting pair (f,μ) satisfying assumptions (i) and (ii) of Theorem 5.3. When it encounters a plentiful node i, it computes a new integral flow g as described in the proof, uses it to identify an abundant edge neighboring i, and then contracts the edge. After contraction, (g,μ) is used to initialize the search for the next plentiful node.

5.3 Producing a Plentiful Node

In order to find a plentiful node for a fitting pair (f,μ), the algorithm uses a subroutine which alternates between the following augmentation and scaling phases until a node becomes plentiful:

- **Augmentation:** Reduce the total excess and deficit of the fitting flow f by augmenting as much flow as possible, one unit at a time, along tight paths from nodes with excess to nodes with deficit. (Augmenting along tight paths maintains a fitting pair.) Formally, let T be the union of the sink t and all nodes with deficit ($\nabla f_i^\mu < b_i^\mu$) and let S ("sources") be the set of all nodes i for which T is reachable from i along a tight path in the residual graph. Continue augmenting from S to T as long as there is some $i \in S$ for which the following conditions hold:

  (i) i has at least one unit of excess ($\nabla f_i^\mu \geq b_i^\mu + 1$); and

  (ii) $b_i < 0$ (this condition is important in the potential analysis).

- **Scaling:** Work toward producing a plentiful node by scaling the dual μ to increase $|b^\mu|$. Any excesses created during this phase must be cancelled by augmentation. In order to make this possible, only nodes in the "reachable" set S are scaled.

  Notionally, a scaling factor $\alpha > 1$ is increased continuously, and $\mu_i \leftarrow \mu_i/\alpha$ for $i \in S$. α increases until either there is a plentiful node in S or some node in S has an excess of at least 1, triggering an augmentation phase. Flow on edges entirely within S is also scaled by $f_e \leftarrow f_e/\alpha$, so that f_{ij}^μ does not change when $i,j \in S$.

Intuitively, scaling is akin to filling a basin with water. As illustrated in the example in Figure 3, imagine a basin with various levels, where the height of the water represents α and each level corresponds to a vertex on the boundary of S. The area under water represents S. As α rises, an incoming edge, for example (i,j) at label (b), becomes tight. This in turn causes i to be added to S, as T is now reachable along a tight path from i. Updating S during scaling maintains the invariant that S is inward-loose. Eventually, some node $k \in S$ (either a newly added node or one that was

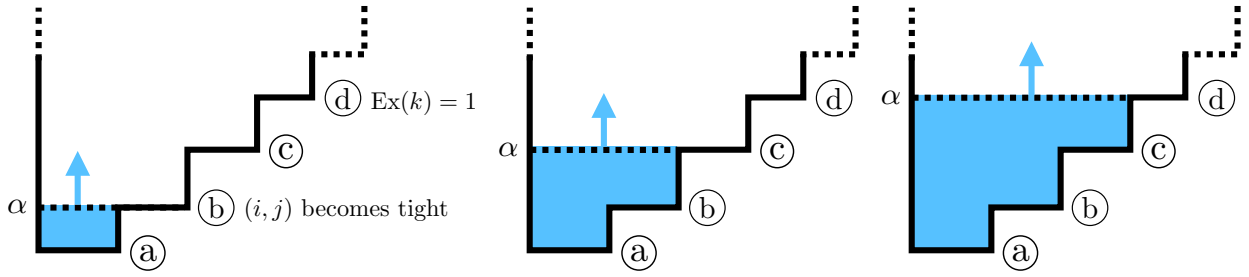

Figure 3: Informal analogy showing an example of how scaling α expands S just enough so that the next augmentation phase can make progress.

there previously) either becomes plentiful or is scaled to the point where its relabeled excess is 1, for example at label (d) in the figure. In the latter case, augmentation can make progress, so the next round of augmentation begins.

In practice, α and S can be updated via a modified Dijkstra's algorithm as follows. Let $c_{ij} = -\log \gamma_{ij}$ be the "cost" of edge (i, j). Then (i, j) is tight exactly when

$$c_{ij} = \log \mu_i - \log \mu_j.$$

In other words, tight paths are shortest paths, $\log \mu$ is a distance function, and S is a "frontier." Scaling μ by α^{-1} in S amounts to adding $\log \alpha$ to the distance of every node in S. Each new node added to S is the closest node under the costs c_{ij}, and it is encountered when $\log \alpha$ rises to reach its distance. Dijkstra's algorithm with costs c_{ij} will compute the maximum distance $\log \alpha$, and it can be terminated when either a plentiful node or an excess greater than one is encountered.

5.4 Analysis

We now demonstrate some invariants preserved by the algorithm. In particular, we show that the algorithm maintains assumptions (i) and (ii) of Theorem 5.3, so that a plentiful node guarantees the existence of a contractible edge.

Lemma 5.4. *Relabeled flow f^μ remains unchanged during each epoch.*

Proof. As seen above, the scaling is designed so that $f^\mu_{ij} = f_{ij}/\mu_i$ stays constant when both i and j are in S. Moreover, it maintains the invariant that there is no flow across the cut S—so flow across S is vacuously preserved under scaling. Finally, scaling does not affect edges entirely outside S. □

Corollary 5.4.1. *f^μ is always integral.*

Proof. The flow is initially rounded to be integral (Section 5.5 below). The augmentation step clearly maintains integrality of f^μ because we always augment by 1 unit at a time. By Lemma 5.4, scaling never changes the value of f^μ at all. Finally, at a contraction, a new integral flow is computed. □

Because f^μ is always integral and excess is always immediately augmented away, the excess at every node stays bounded by 2.

Lemma 5.5. *(f, μ) remains a fitting pair throughout each epoch.*

Proof. Suppose that, at the beginning of the epoch, f fits μ. That is, μ is dual-feasible, and whenever $f_{ij} > 0$, $\gamma^\mu_{ij} = 1$. Augmentation only occurs along tight edges, thus maintaining the fitting property. Scaling only changes $\gamma^\mu_{ij} = \gamma_{ij}\mu_i/\mu_j$ when (i, j) crosses between S and $V \setminus S$. But there is no flow crossing this cut, so scaling never violates the fitting property. Moreover, as soon as an edge crossing the cut becomes tight, S is expanded. Thus, the constraint $\gamma^\mu_{ij} \leq 1$ is never violated during scaling, i.e., μ remains feasible. □

Lemma 5.6. *μ remains safe throughout each epoch, i.e. there is a corresponding feasible primal solution even though we haven't kept track of it.*

Proof Sketch. Only the scaling rounds modify μ, so it suffices to show that one scaling step, during which at most one node is added to S, preserves safety. Assume that μ is initially safe, and assume by way of contradiction that the updated labeling μ' is no longer safe. That is, there exists some inward-loose cut $X \subseteq V \setminus t$ with net positive demand $b^{\mu'}(X) > 0$.

Divide X into $X \cap S$ and $X \setminus S$, where S is defined by the original μ. Recall that scaling only affects $X \cap S$, so we can write the net demand under μ' in terms of the demands under μ as follows:

$$b^{\mu'}(X) = \frac{1}{\alpha} b^{\mu}(X \cap S) + b^{\mu}(X \setminus S)$$

Either $b^{\mu}(X \cap S)$ or $b^{\mu}(X \setminus S)$ must be positive.

Suppose $b^{\mu}(X \cap S) > 0$. Let $E[S]$ denote the set of edges with both endpoints in S. Observe that $\delta^{\text{in}}(X \cap S) \subseteq (\delta^{\text{in}}(X) \cap E[S]) \cup \delta^{\text{in}}(S)$. By definition of S, $\delta^{\text{in}}(S) \cap \tau(\mu) = \varnothing$. Moreover, $E[S] \cap \tau(\mu) \subseteq E[S] \cap \tau(\mu')$. So

$$\delta^{\text{in}}(X) \cap E[S] \cap \tau(\mu) \subseteq \delta^{\text{in}}(X) \cap \tau(\mu) = \varnothing.$$

So $X \cap S$ is an inward-loose cut with positive net demand under μ, contradicting the safety of μ.

The case where $b^{\mu}(X \setminus S) > 0$ is similar, though slightly more complicated. □

In summary, we have shown that augmentation and scaling maintain a fitting pair (f, μ) with μ safe and $\text{Ex}(f, \mu)$ bounded, while increasing $|b_i^{\mu}|$. This will eventually produce a plentiful node.

5.5 Initialization

The algorithm needs to start with a fitting pair (f, μ) where μ is safe, f is integral, and f has bounded excess, as required in Theorem 5.3.

To do this, the algorithm starts by computing a conservative pair $(\bar{f}, \mu)$, thus ensuring that μ is safe. This feasibility problem can be solved by running the same generalized flow algorithm on a carefully constructed graph whose optimum is a feasible solution to the original problem. This is akin to the way the simplex algorithm is initialized.

Next, μ is scaled so that the relabeled excess is uniformly bounded. Then, the flow is "rounded" to be integral by computing an ordinary flow f^{μ} on the tight edges defined by μ under the constraints

$$\lfloor \nabla \bar{f}_i^{\mu} \rfloor \leq \nabla f_i^{\mu} \leq \lceil \nabla \bar{f}_i^{\mu} \rceil.$$

The result is a fitting pair such that f^{μ} is integral and satisfies

$$b_i^{\mu} - 1 \leq \nabla f_i^{\mu} \leq b_i^{\mu} - 2.$$

The algorithm proper begins with this fitting pair.

5.6 Running Time Analysis

The algorithm consists of three basic types of operations: scaling, augmentation, and contraction. The number of contractions is bounded by n. Contraction itself requires computing a regular flow and performing a bounded number of graph operations, but strongly polynomial algorithms already exist for these. So the goal is to bound the running time (number of arithmetic operations) between any two contractions.

Augmentation itself consists of a graph search on the tight subgraph, an $O(m)$ operation. Between rounds of augmentation, there is a scaling round. Each scaling phase stops when either (i) a node has become plentiful, leading to a contraction, or (ii) a node has an excess greater than 1, leading to a new augmentation round. If the scaling is done by Dijkstra's algorithm as described in Section 5.3, the cost of each scaling phase (and thus the overall cost per augmentation) is $O(m + n \log n)$.

It remains to bound the total number of augmentations the algorithm may perform. We'd like to show that after a strongly polynomial number of augmentations, the algorithm is guaranteed to find a plentiful node.

Lemma 5.7. *After $O(mn)$ augmentations, the algorithm is guaranteed to find a plentiful node.*

First, a preliminary lemma. We denote the period between successive contraction steps as an epoch.

Lemma 5.8. *At any node $i \in V^s$, the deficit is at most one. Moreover, at most one augmentation ends at such a node during any epoch.*

Proof. At the beginning of the algorithm, the flow is feasible, so $\nabla f_i^\mu - b_i^\mu \geq 0$. Rounding decreases ∇f_i^μ by less than one. Similarly, during each contraction, the adjusted flow is computed with $\nabla g_i^\mu \geq \lfloor b_i^\mu \rfloor$ everywhere.

Between contractions, the deficit can be modified by scaling and augmentation steps. Since $i \in V^s$, $b_i^\mu < 0$, and so b_i^μ can only decrease during scaling. Meanwhile, augmentations only decrease deficits. Once $\nabla f_i^\mu \geq b_i^\mu$, no augmentation ending at i will be contemplated unless i experiences a deficit again. The only way for this to happen would be for augmentations beginning at i to reduce ∇f_i^μ below b_i^μ. However, an augmentation will only begin at i if $\nabla f_i^\mu - b_i^\mu \geq 1$, and will reduce ∇f_i^μ by exactly one. Hence, i will never again experience a deficit during the epoch, and so there will be no further augmentations ending at i. □

Proof of Lemma 5.7. Define two potential functions

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

By Lemma 5.8, $\Phi(f, \mu) \geq -\operatorname{Def}(f, \mu) \geq -n$ throughout the algorithm. Similarly, $\Phi(f, \mu) \leq \operatorname{Ex}(f, \mu) \leq 2n$. Finally, it follows from Lemma 5.8 that at most n augmentations may end in V^s. All augmentations start in V^s, so all other augmentations decrease Φ. So after r augmentations, Φ has decreased by at least $r - n$. Its total decrease is bounded by $2n - -n = 3n$, so there must be a corresponding increase of at least $r - 4n$ during scaling. So Ψ also increases by at least $r - 4n$, as f^μ is constant under scaling.

Therefore, after $O(mn)$ augmentations, Ψ is sufficiently large so as to guarantee the existence of a plentiful node in V^s. □

We have shown that there are at most n epochs of $O(mn)$ augmentations each, with each augmentation costing $O(m + n \log n)$. This implies an $O(mn^2(m + n \log n))$ time bound overall. With a more careful potential analysis, the bound can be improved to $O(mn \log(n^2/m)(m + n \log n))$.

6 Conclusion

We presented and compared two strongly polynomial algorithms for the maximum generalized flow problem. Both algorithms take a combinatorial approach by computing a set of tight edges consistent with an optimal dual solution. In both cases, this set is built up inductively by contracting edges. Finally, in both cases, a pair of primal and dual solutions is used to construct certificates of contractibility. Olver and Végh [4] achieve an improvement of almost $O(n^2)$ by relaxing the feasibility of the primal flow component of the pair. This allows excess to be cancelled with deficit, making it much easier to keep both bounded and thus reducing the time required to find each contractible edge.

References

[1] László A. Végh. "A Strongly Polynomial Algorithm for Generalized Flow Maximization". In: *Mathematics of Operations Research* 42 (2016).

[2] Andrew V Goldberg and Serge a Plotkin. "Combinatorial Algorithms for the Generalized Circulation Problem". In: *Operations Research* (1989), pp. 1–35. ISSN: 0364-765X. DOI: 10.1287/moor.16.2.351.

[3] Andrew V. Goldberg, Serge A. Plotkin, and Éva Tardos. "Combinatorial Algorithms for the Generalized Circulation Problem". In: *Math. Oper. Res.* 16.2 (Apr. 1991), pp. 351–381. ISSN: 0364-765X. DOI: 10.1287/moor.16.2.351. URL: http://dx.doi.org/10.1287/moor.16.2.351.

[4] Neil Olver and László 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 - STOC 2017* (2017), pp. 100–111. DOI: 10.1145/3055399.3055439. URL: http://dl.acm.org/citation.cfm?doid=3055399.3055439.

[5] James B. Orlin. "A Faster Strongly Polynomial Minimum Cost Flow Algorithm". In: *Acm Stoc* 41.2 (1988), pp. 377–387. ISSN: 1098-6596. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[6] Tomasz Radzik. "Improving time bounds on maximum generalised flow computations by contracting the network". In: *Theoretical Computer Science* 312.1 (2004), pp. 75–97. ISSN: 03043975. DOI: 10.1016/S0304-3975(03)00403-1.

[7] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media, 2002.

[8] Éva Tardos and Kevin D. Wayne. "Simple generalized maximum flow algorithms". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1412 (1998), pp. 310–324. ISSN: 16113349. DOI: 10.1007/3-540-69346-7_24.

[9] K. Truemper. "On max flows with gains and pure min-cost flows". In: *SIAM Journal on Applied Mathematics* 32.2 (1977), pp. 450–456. URL: http://epubs.siam.org/doi/abs/10.1137/0132037.

[10] László A. Végh. "A strongly polynomial algorithm for generalized flow maximization". In: (2013), pp. 1–44. ISSN: 07378017. DOI: 10.1145/2591796.2591806. arXiv: 1307.6809. URL: http://arxiv.org/abs/1307.6809.