A strongly polynomial algorithm for generalized flow maximization

[Extended Abstract] *

László A. Végh Department of Management
London School of Economics and Political Science
L.Vegh@lse.ac.uk

ABSTRACT
A strongly polynomial algorithm is given for the generalized flow maximization problem. It uses a new variant of the scaling technique, called continuous scaling. The main measure of progress is that within a strongly polynomial number of steps, an arc can be identified that must be tight in every dual optimal solution, and thus can be contracted.

Categories and Subject Descriptors
G.2.2 [Mathematics of Computing]: Discrete Mathematics—Graph Theory: Graph algorithms, Network problems; G.1.6 [Mathematics of Computing]: Numerical Analysis—Linear Programming; F.1.1 [Theory of Computation]: Computation by Abstract Devices—Models of Computation

General Terms
Algorithms, Theory

Keywords
generalized flows, strongly polynomial algorithms, linear programming, combinatorial algorithms

1. INTRODUCTION
The generalized flow model is a classical extension of network flows. Besides the capacity constraints, for every arc there is a gain factor \( \gamma > 0 \), such that flow amount gets multiplied by \( \gamma e \) while traversing the arc \( e \). We study the flow maximization problem, where the objective is to send the maximum amount of flow to a sink node \( t \). The model was already formulated by Kantorovich [17], as one of the first examples of linear programming; it has several applications in operations research [2, Chapter 15]. Gain factors can be used to model physical changes such as leakage or theft. Other common applications use the nodes to represent different types of entities, e.g. different currencies, and the gain factors correspond to the exchange rates.

The existence of a strongly polynomial algorithm for linear programming is a major open question from a theoretical perspective. This refers to an algorithm with the number of arithmetic operations polynomially bounded in the number of variables and constraints, and the size of the numbers during the computations polynomially bounded in the input size. The landmark result by Tardos [27] gives an algorithm with the running time dependent only on the size of numbers in the constraint matrix, but independent from the right-hand side and the objective vector. This gives strongly polynomial algorithms for several combinatorial problems such as minimum cost flows (see also Tardos [26]) and multicommodity flows. Instead of the sizes of numbers, one might impose restrictions on the structure of the constraint matrix. Hence a natural question arises whether there exists a strongly polynomial algorithm for linear programs (LPs) with at most two nonzero entries per column (that can be arbitrary numbers). This question is still open; as shown by Hochbaum [15], all such LPs can be polynomially transformed to instances of the minimum cost generalized flow problem. (Note also that every LP can be polynomially transformed to an equivalent one with at most three nonzero entries per column.)

Generalized flow maximization is an important special case of minimum cost generalized flows; it is probably the simplest natural class of LP where no strongly polynomial algorithm has been known. The existence of such an algorithm has been a well-studied and longstanding open problem (see e.g. [8, 3, 31, 23, 25]) A strongly polynomial algorithm for the corresponding dual feasibility problem was given by Megiddo [19], but this is not applicable to flow maximization. A strongly polynomial algorithm for some restricted classes of generalized flow problems was given by Adler and Cosares [1].

In this paper, we exhibit a strongly polynomial algorithm for generalized flow maximization. Let \( n \) denote the number of nodes and \( m \) the number of arcs in the network, and let \( B \) denote the largest integer used in the description of the input (see Section 2 for the precise problem setting). A strongly polynomial algorithm for the problem entails the following (see [14]): (i) it uses only elementary arithmetic operations (addition, subtraction, multiplication, division), and comparisons; (ii) the number of these operations is bounded by a polynomial of \( n \) and \( m \); (iii) if all numbers in the input are rational, then all numbers occurring in the computations are

*A full version of the paper is available on arxiv:1307.6809
rational numbers of encoding size polynomially bounded in $n, m$ and $B$. In this extended abstract we outline a simpler version of our algorithm that satisfies requirements (i) and (ii) only, using a model with real number computations, ignoring the encoding sizes. In order to satisfy (iii) as well, additional rounding steps have to be introduced; this is done in Section 7 of the full version.

Combinatorial approaches have been applied to generalized flows already in the sixties by Dantzig [4] and Jewell [16]. However, the first polynomial-time combinatorial algorithm was only given in 1991 by Goldberg, Plotkin and Tardos [8]. This was followed by a multitude of further combinatorial algorithms e.g. [3, 10, 12, 28, 6, 11, 13, 31, 23, 24, 30]; a central motivation of this line of research was to develop a strongly polynomial algorithm. The algorithms of Cohen and Megiddo [3], Wayne [31], and Restrepo and Williamson [24] present fully polynomial time approximation schemes, that is, for every $\varepsilon > 0$, they can find a solution within $\varepsilon$ from the optimum value in running time polynomial in $n, m$ and $\log(1/\varepsilon)$. This can be transformed to an optimal solution for a sufficiently small $\varepsilon$; however, this value does depend on $B$ and hence the overall running time will also depend on $\log B$. The current most efficient weakly polynomial combinatorial algorithms are the interior point approach of Kapoor an Vaidya [18] with running time $O(m^{5.5}n^{2.5}\log B)$, and the combinatorial algorithm by Radzik [23] with running time $O(m^2n^2\log B)$.

For a survey on combinatorial generalized flow algorithms, see Shigeno [25].

The generalized flow maximization problem exhibits deep structural similarities to the minimum cost circulation problem, as first pointed out by Truemper [29]. Most combinatorial algorithms for generalized flows, including both algorithms by Goldberg et al. [8], exploit this analogy and adapt existing efficient techniques from minimum cost circulations. For the latter problem, several strongly polynomial algorithms are known, the first given by Tardos [26]; others relevant to our discussion are those by Goldberg and Tarjan [9], and by Orlin [21]; see also [2, Chapters 9-11]. Whereas these algorithms serve as starting points for most generalized flow algorithms, the applicability of the techniques is by no means obvious, and different methods have to be combined. As a consequence, the strongly polynomial analysis cannot be carried over when adapting minimum cost circulation approaches to generalized flows, although weakly polynomial bounds can be shown.

To achieve a strongly polynomial guarantee, further new algorithmic ideas are required that are specific to the structure of generalized flows. The new ingredients of our algorithm are highlighted in Section 2.3.

Let us now outline the scaling method for minimum cost circulations, a motivation of our generalized flow algorithm. The first (weakly) polynomial time algorithm for minimum cost circulations was given by Edmonds and Karp [5], introducing the simple yet powerful idea of scaling (see also [2, Chapter 9.7]). The algorithm consists of $\Delta$-phases, with the value of $\Delta > 0$ decreasing by a factor of at least two between every two phases, yielding an optimal solution for sufficiently small $\Delta$. In the $\Delta$-phase, the flow is transported in units of $\Delta$ from nodes with excess to nodes with deficiency using shortest paths in the graph of arcs with residual capacity at least $\Delta$. Orlin [21], (see also [2, Chapters 10.6-7]) devised a strongly polynomial version of this algorithm.

We refer to [21] and thereby the quick appearance of an abundant arc “would always appear in any of the above algorithms within a strongly polynomial number of steps.

Our contribution is a new type of scaling algorithm that suits better the dual structure of the generalized flow problem, and thereby the quick appearance of an abundant arc will be guaranteed. Whereas in all previous methods, the scaling factor $\Delta$ remains constant for a linear number of path augmentations, our continuous scaling method keeps it decreasing in every elementary iteration of the algorithm, even in those that lead to finding the next augmenting path.

The rest of this extended abstract is structured as follows. Section 2 first defines the problem setting, introduces relabelings, gives the characterization of optimality, and defines the notion of $\Delta$-feasibility. Section 2.3 then gives a more detailed account of the main algorithmic ideas. In Section 3 we first describe a simpler weakly polynomial version of our algorithm, whereas a sketch of the strongly polynomial algorithm is given in Section 4.

2. PRELIMINARIES

Let $G = (V, E)$ be a directed graph with a designated sink node $t \in V$. Let $V = |V|$, $m = |E|$, and for each node $i \in V$, let $d_i$ denote total number of arcs incident to $i$ (both entering and leaving). We will always assume $n \leq m$. We do not allow parallel arcs and hence we may use $ij$ to denote the arc from $i$ to $j$. This is for notational convenience only, and all result straightforwardly extend to the setting with parallel arcs. For an arc set $F \subseteq E$ and node set $S \subseteq V$, let $F[S] := \{ij \in F : i, j \in S\}$ denote the set of arcs in $F$ spanned in $S$. All paths and cycles in the paper will refer to directed paths and directed cycles.

The following is the standard formulation of the problem. Let us be given arc capacities $u : E \to \mathbb{R}_{>0}$ and gain factors

\[ \min \sum_{ij \in F} c_{ij} u_{ij} \]

subject to

\begin{align*}
\sum_{j \in S} f_{ij} &= \sum_{j \in S} f_{ji} & & \forall i \in V \\
\sum_{i \in S} f_{ij} &= x_i & & \forall j \in V \\
\sum_{i \in S} x_i &= 0 & & S \neq \emptyset \\
f_{ij} &\geq 0 & & \forall i, j \in V
\end{align*}

\[ f_{ij} \leq u_{ij} (1 + \gamma) & & \forall ij \in F, \forall \gamma \geq 0 \]

The $\tilde{O}(\cdot)$ notation hides a polylogarithmic factor.
\( \gamma: E \to \mathbb{R}_{>0} \)

\[
\begin{align*}
\max & \sum_{j,j' \in E} \gamma_{ji} f_{ji} - \sum_{j,j' \in E} f_{ij} \\
& \sum_{j,j' \in E} \gamma_{ji} f_{ji} - \sum_{j,j' \in E} f_{ij} \geq 0 \quad \forall i \in V - t \quad (P_u) \\
& 0 \leq f \leq u 
\end{align*}
\]

It is common in the literature to define the problem with equalities in the node constraints. The two forms are essentially equivalent, see e.g. \([25]\); moreover, the form with equality is often solved via a reduction to \((P_u)\). In this paper, we prefer to use yet another equivalent formulation, where all arc capacities are unbounded, but there are node demands instead. A problem given in the standard formulation can be easily transformed to an equivalent instance in demands instead. A problem given in the standard formulation is essentially equivalent, see e.g. \([25]\); moreover, the form with equalities in the node constraints. The two forms are essentially equivalent, see e.g. \([25]\); moreover, the form with equality is often solved via a reduction to \((P_u)\). In this paper, we prefer to use yet another equivalent formulation, where all arc capacities are unbounded, but there are node demands instead. A problem given in the standard formulation can be easily transformed to an equivalent instance in this form; the transformation is described in Section 8.1 of the full version. Given a node demand vector \(b: V - t \to \mathbb{R}\) and gain factors \(\gamma: E \to \mathbb{R}_{>0}\), the uncapacitated formulation is defined as

\[
\max \sum_{j,j' \in E} \gamma_{ji} f_{ji} - \sum_{j,j' \in E} f_{ij} \\
\sum_{j,j' \in E} \gamma_{ji} f_{ji} - \sum_{j,j' \in E} f_{ij} \geq b_i \quad \forall i \in V - t \quad (P)
\]

For a vector \(f \in \mathbb{R}^{|E|}\), let us define the excess of a node \(i \in V\) by

\[e_i(f) := \sum_{j \in E} \gamma_{ji} f_{ji} - \sum_{j \in E} f_{ij} - b_i.
\]

The node constraints in \((P)\) can be written as \(e_i(f) \geq 0\), and the objective is equivalent to maximizing \(e_i(f)\). When \(f\) is clear from the context, we will denote the excess simply by \(e_i := e_i(f)\). By a generalized flow we mean a feasible solution to \((P)\), that is, a nonnegative vector \(f \in \mathbb{R}^{|E|}\) with \(e_i(f) \geq 0\) for all \(i \in V - t\). For convenience, we define \(b_i = -\infty\), or some very small value, such that \(e_i(f) < 0\) must hold for every feasible \(f\). Let us define the surplus of \(f\) as

\[Ex(f) := \sum_{i \in V - t} e_i(f).
\]

It will be convenient to make the following assumptions; in Section 8.1 of the full version it is shown that any problem in the standard form can be transformed to an equivalent one in the uncapacitated form that also satisfies these assumptions.

There is an arc \(i \in E\) for every \(i \in V - t\); \((\star)\)

We are given an initial feasible solution \(\bar{f}\) to \((P)\); \((\star\star)\)

Note that for \((P_u)\), \(f \equiv 0\) is a feasible solution; \(\bar{f}\) in \((\star\star)\) will be the image of 0 under the transformation. Our main result is the following.

**Theorem 2.1.** There exists a strongly polynomial algorithm for the uncapacitated formulation \((P)\) with running time \(O(n^3m^2)\).

This in turn gives an \(O(m^5)\) time strongly polynomial algorithm for the standard formulation \((P_u)\).

### 2.1 Labelings and optimality conditions

Dual solutions to \((P)\) play a crucial role in the entire generalized flow literature. Let \(\lambda: V \to \mathbb{R}_+\) be a solution to the dual of \((P)\). Following Glover and Klingman \([7]\), the literature standard is not to consider the \(\lambda\) values but their inverses instead. With \(\mu_i := 1/\lambda_i\), we can write the dual of \((P)\) in the following form.

\[
\begin{align*}
\max & \sum_{i \in V} b_i \\
& \gamma_{ij} \mu_i \leq \mu_j \quad \forall ij \in E \\
& \mu_i > 0 \quad \forall i \in V - t \\
& \mu_i = 1 \\
& \mu \in (\mathbb{R}_+ \cup \{\infty\})^{|V|}
\end{align*}
\]

A feasible solution \(\mu\) to this program will be called a relabeling or labeling. An optimal labeling is an optimal solution to \((D)\). Under assumption \((\ast)\), all \(\mu_i\) values must be finite. A useful and well-known property is the following (see also Section 2.4).

**Proposition 2.2.** Given an optimal solution to \((D)\), an optimal solution to \((P)\) can be obtained in strongly polynomial time, and conversely, given an optimal solution to \((P)\), an optimal solution to \((D)\) can be obtained in strongly polynomial time.

In fact, our strongly polynomial algorithm will proceed via finding an optimal solution to \((D)\), and computing the primal optimal solution via a single maximum flow computation. Relabelings will be used in all parts of the algorithm and proofs. For a generalized flow \(f\) and a labeling \(\mu\), we define the relabeled flow \(f^\mu\) by \(f^\mu_{ij} := \frac{f_{ij}}{\mu_j}\) for all \(ij \in E\). This can be interpreted as changing the base unit of measure at the nodes (i.e. in the example of the currency exchange network, it corresponds to changing the unit from pounds to pennies). To get a problem setting equivalent to the original one, we have to relabel all other quantities accordingly. That is, we define relabeled gains, demands, excesses and surplus by

\[
\gamma_{ij}^\mu := \gamma_{ij} \frac{\mu_j}{\mu_i}, \quad b_{ij}^\mu := \frac{b_{ij}}{\mu_i}, \quad e_i^\mu := \frac{e_i}{\mu_i},
\]

and

\[Ex^\mu(f) := \sum_{i \in V - t} e_i^\mu,
\]

respectively. Another standard notion is the residual network \(G_f = (V, E_f)\) of a generalized flow \(f\), defined as

\[E_f := E \cup \{ij : ji \in E, f_{ji} > 0\}.
\]

Arcs in \(E\) are called forward arcs, while arcs in the second set are reverse arcs. For a forward arc \(ij\), let \(\gamma_{ij} < \infty\) denote the same as in the original graph. For a reverse arc \(ji\), let \(\gamma_{ji} := 1/\gamma_{ij}\). Also, we define \(f_{ji} := -\gamma_{ij} f_{ij}\) for every reverse arc \(ji \in E_f\). By increasing (decreasing) \(f_{ji}\) by \(\alpha\) on a reverse arc \(ji \in E_f\), we mean decreasing (increasing) \(f_{ij}\) by \(\alpha/\gamma_{ij}\).

The crucial notion of conservative labeling is motivated by primal-dual slackness. Let \(f\) be a generalized flow (that is, a feasible solution to \((P)\)), and let \(\mu: V \to \mathbb{R}_{>0}\). We say that \(\mu\) is a conservative labeling for \(f\), if \(\mu\) is a feasible solution to \((D)\) with the further requirement that \(\gamma_{ij}^\mu = 1\) whenever \(f_{ij} > 0\) for \(ij \in E\). The following characterization of optimality is a straightforward consequence of primal-dual slackness in linear programming.
Theorem 2.3. Assume (⋆) holds. A generalized flow $f$ is an optimal solution to $(P)$ if and only if there exists a conservative labeling $\mu$ such that that $\epsilon_i = 0$ for all $i \in V − t$.

Given a labeling $\mu$, we say that an arc $ij \in E_f$ is tight if $\gamma^\mu_{ij} = 1$. A directed path in $E_f$ is called tight if it consists of tight arcs.

2.2 $\Delta$-feasible labels

Let us now introduce a relaxation of conservativity crucial in the algorithm. This is a new notion, although similar concepts have been used in previous scaling algorithms [10, 30]. Section 2.3 explains the background and motivation of this notion. Given a labeling $\mu$, let us call arcs in $E$ with $\gamma^\mu_{ij} < 1$ non-tight, and denote their sets by

$$F^\mu := \{ij \in E : \gamma^\mu_{ij} < 1\}.$$  

For every $i \in V$, let

$$R_i := \sum_{j \in V : F^\mu} \gamma^\mu_{ji} f_{ji}$$

denote the total flow incoming on non-tight arcs; let $R^\mu_i := R_i / \mu_i = \sum_{j \in V : F^\mu} \gamma^\mu_{ji} f_{ji}$. For some $\Delta \geq 0$, let us define the $\Delta$-fat graph as

$$E_f^\mu(\Delta) = E \cup \{ij : ji \in E, f^\mu_{ji} > \Delta\}.$$  

We say that $\mu$ is a $\Delta$-conservative labeling for $f$, or that $(f, \mu)$ is a $\Delta$-feasible pair, if

- $\gamma^\mu_{ij} \leq 1$ holds for all $ij \in E_f^\mu(\Delta)$, and
- $\mu_i = 1$, and $\mu_i > 0$, $\epsilon_i \geq R_i$ for every $i \in V − t$.

Note that in particular, $\mu$ must be a feasible solution to $(D)$. The first condition is equivalent to requiring $f^\mu_{ij} \leq \Delta$ for every non-tight arc. Note that 0-conservativeness is identical to conservativeness: $E_f^\mu(\Delta) = E_f^\mu$, and therefore every arc carrying positive flow must be tight: the second condition simply gives $\epsilon_i \geq 0$ whenever $\mu_i > 0$. The next lemma can be seen as the converse of this observation.

Lemma 2.4. Let $(f, \mu)$ be a $\Delta$-feasible pair for some $\Delta > 0$. Let us define the generalized flow $f$ with $f_{ji} = 0$ if $ij \in F^\mu$ and $f_{ij} = f_{ji}$ otherwise. Then $\mu$ is a conservative labeling for $f$, and $Ex^\mu(f) \leq Ex^\mu(f) + |F^\mu| \Delta$.

Claim 2.5. In a $\Delta$-conservative labeling, $R^\mu_i < \delta_i \Delta$ holds for every $i \in V$.

2.3 Overview of the algorithms

We now informally describe some fundamental ideas of our algorithms CONTINUOUS SCALING and ENHANCED CONTINUOUS SCALING, and explain their relations to previous generalized flow algorithms.

Basic features and previous work

By sending $\alpha$ units of flow on a walk $P = w_1 w_2 \ldots w_t \subseteq E_f$, we mean increasing the flow on $w_1$ by $p_1 = \alpha$, and increasing the flow on $w_t$ by $p_t = \gamma_{w_{t−1} P_{t−1}}$ for all $1 < i < t$ (assuming that the residual capacities do not get violated). Note that if $P$ is a cycle then this will change $w_i$ on precisely one node $i$.

Given a generalized flow $f$, a cycle $C$ in the residual graph $E_f$ is called flow generating, if $\gamma(C) = \prod_{c \in C} \gamma_c > 1$. If there exists a flow generating cycle, then some positive amount of flow can be sent around it to create positive excess in an arbitrary node $i$ incident to $C$.

The notion of conservative labellings is closely related to flow generating cycles. Notice that for an arbitrary labeling $\mu$, $\gamma(C) = \gamma^\mu(C)$. Therefore, if $\mu$ is a conservative labeling, then there cannot be any flow generating cycle in $E_f$. It is also easy to verify the converse: if there are no flow generating cycles, then there exists a conservative labeling.

The MAXIMUM-MEAN-GAIN CYCLE-CANCELING procedure, introduced in [8], can be used to eliminate all flow generating cycles efficiently. The subroutine proceeds by choosing a cycle $C \subseteq E_f$ maximizing $\gamma(C) 1/C$, and from an arbitrary node $i$ incident to $C$, sending the maximum possible amount of flow around $C$ admitted by the capacity constraints, thereby increasing the excess $\epsilon_i$. It terminates once there are no more flow generating cycles left in $E_f$. This is a natural analogue of the minimum mean cycle cancelation algorithm of Goldberg and Tarjan [9] for minimum cost circulations. Radzik [22] (see also Shigeno [25]) gave a strongly polynomial running time bound $O(m n^2 \log n)$ for the MAXIMUM-MEAN-GAIN CYCLE-CANCELING algorithm.

Our algorithm also starts with performing this algorithm, with the input being the initial solution $f$ provided by (⋆). Hence one can obtain a feasible solution $f$ along with a conservative labeling $\mu$ in strongly polynomial time.

Such an $f$ can be transformed to an optimal solution using Onaga’s algorithm [20]: while there exists a node $i \in V − t$ with $\epsilon_i > 0$, find a highest gain augmenting path from $i$ to $t$, that is, a path $P$ in the residual graph $E_f$ with the product of the gains maximum. Send the maximum amount of flow on this augmenting path enabled by the capacity constraints. A conservative labeling can be used to identify such paths: we can transform a conservative labeling to a canonical labeling (see [8]), where every node $i$ is connected to the sink via a tight path. Such a canonical labeling can be found via a Dijkstra-type algorithm, increasing the labels of certain nodes. The correctness of Onaga’s algorithm follows by the observation that sending flow on a tight path maintains the conservativeness of the labeling, hence no new flow generating cycles may appear.

Unfortunately, Onaga’s algorithm may run in exponentially many steps, and might not even terminate if the input is irrational. The FAT-PATH algorithm [8] introduces a scaling technique to overcome this difficulty. The algorithm maintains a scaling factor $\Delta$ that decreases geometrically. In the $\Delta$-phase, flow is sent on a highest gain “$\Delta$-fat” augmenting path, that is, a highest gain path among those that have sufficient capacity to send $\Delta$ units of flow to the sink. However, this might create new flow generating cycles, that have to be cancelled by calling the cycle-canceling subroutine at the beginning of every phase.

Our notion of $\Delta$-feasible pairs in Section 2.2 is motivated by the idea of $\Delta$-fat paths: note that every arc in the $\Delta$-fat graph $E_f(\Delta)$ has sufficient capacity to send $\Delta$ units of relabeled flow. A main step in our algorithm will be sending $\Delta$ units of relabeled flow on a tight path in $E_f(\Delta)$ from a node with “high” excess to the sink $t$ or another node with “low” excess. This is in contrast to FAT-PATH and most other algorithms, where these augmenting paths always terminate in the sink $t$. We allow termination in other nodes as well in order to guarantee that the conditions $\epsilon_i \geq R_i$ are maintained during the algorithm. The purpose of these conditions is
to make sure that we always stay “close” to a conservative labeling; recall Lemma 2.4 asserting that if \((f, \mu)\) is a \(\Delta\)-feasible pair, then if we set the flow values to 0 on every non-tight arc, the resulting \(\tilde{f}\) is a feasible solution to \((P)\) not containing any flow generating cycles. That is the reason why we need to call the cycle-canceling algorithm only once, at the initialization, in contrast to Fat-Path.

Similar ideas have been already used previously. The algorithm of Goldfarb, Jin and Orlin [10] also uses a single initial cycle-canceling and then performs highest-gain augmentations in a scaling framework, combined with a clever bookkeeping on the arcs. The algorithm in [30] does not perform any cycle cancellations and uses a homononymous notion of \(\Delta\)-conservativeness that is closely related to ours; however, it uses a different problem setup (called “symmetric formulation”), and includes a condition stronger than \(e_i \geq R_i\).

The way to the strongly polynomial bound

The basic principle of our strongly polynomial algorithm is motivated by Orlin’s strongly polynomial algorithm for minimum cost circulations ([21], see also [2, Chapters 10.6–11]). The true purpose of the algorithm will be to compute a dual optimal solution to \((D)\). Provided a dual optimal solution, we can compute a primal optimal solution to \((P)\) by a single maximum flow computation on the network of tight arcs.

The main measure of progress will be identifying an arc \(ij \in E\) that must be tight in every dual optimal solution. Such an arc can be contracted, and an optimal dual solution to the contracted instance can be easily extended to an optimal dual solution on the original instance. The algorithm can be simply restarted from scratch in the contracted instance. The algorithm ENHANCED CONTINUOUS SCALING in the full version is somewhat more complicated and keeps the previous primal solution to achieve better running time bounds by a global analysis of all contraction phases.

We use a scaling-type algorithm to identify such arcs tight in every dual optimal solution. Our algorithm always maintains a scaling parameter \(\Delta\), and a \(\Delta\)-feasible pair \((f, \mu)\) such that \(Ex^+(f) \leq 16m\Delta\). Using standard flow decomposition techniques, it can be shown that an arc \(ij\) with \(f^+_{ij} \geq 17m\Delta\) must be positive in some optimal solution \(f^+\) to \((P)\). Then by primal-dual slackness it follows that this arc is tight in every dual optimal solution. Arcs with \(f^+_{ij} \geq 17m\Delta\) will be called abundant.

A simple calculation shows that once \(|b^i_t| \geq 20mn\Delta\) for a node \(i \in V - t\), there must be an abundant arc leaving or entering \(i\). Hence our goal is to design an algorithm where such a node appears within a strongly polynomial number of iterations.

A basic step in the scaling approaches (e.g., [8, 10, 30]) is sending \(\Delta\) units of relabeled flow on a tight path; we shall call this a path augmentation. In all previous approaches, the scaling factor \(\Delta\) remained fixed for a number of path augmentations, and reduces by a substantial amount (by at least a factor of two) for the next \(\Delta\)-phase. Our main idea is what we call continuous scaling: the boundaries between \(\Delta\)-phases are dissolved, and the scaling factor decreases continuously, even during the iterations that lead to finding the next path for augmentation. We now give a high-level overview.

We shall have a set \(T_0\) with nodes of “high” relabelled excess; another set \(N\) will be the set of nodes with “low” relabelled excess, always including the sink \(t\). We will look for tight paths connecting a node in \(T_0\) to one in \(N\); we will send \(\Delta\) units of relabeled flow along such a path. In an intermediate elementary step, we let \(T\) to denote the set of nodes reachable from \(T_0\) on a tight path; if it does not intersect \(N\), then we increase the labels \(\mu_i\) for all \(i \in T\) by the same factor \(\alpha\) hoping that a new tight arc appears between \(T\) and \(V \setminus T\), and thus \(T\) can be extended. We simultaneously decrease the value of \(\Delta\) by the same factor \(\alpha\). Thus the relabeled excess of nodes in \(V \setminus T\) increases relative to \(\Delta\). This might lead to certain changes in the sets \(T_0\) and \(N\); hence an elementary step does not necessarily terminate when a new tight arc appears, and the value of \(\alpha\) must be carefully chosen.

This framework is undoubtedly more complicated than the traditional scaling algorithms. The main reason for this approach is the phenomenon one might call “inflation” in the previous scaling-type algorithms. There it might happen that the relabeling steps used for identifying the next augmenting paths increase some labels by very high amounts, and thus the relabeled flow remains small compared to \(\Delta\) on every arc of the network - therefore a new abundant arc can never be identified. It could even be the case that most \(\Delta\)-scaling phases do not perform any path augmentations at all, but only label updates: the relabeled excess at every node becomes smaller than \(\Delta\) during the relabeling steps.

The advantage of changing \(\Delta\) continuously in our algorithm is that the ratios \(|b^i_t|/\Delta\) are nondecreasing for every \(i \in V - t\) during the entire algorithm. In the above described situation, these ratios are unchanged for \(i \in T\) and increase for \(i \in V \setminus T\). As remarked above, there must be an abundant arc incident to \(i\) once \(|b^i_t| \geq 20mn\Delta\).

We first present a simpler version of this algorithm, CONTINUOUS SCALING, proving only a weakly polynomial running time bound. Whereas the ratios \(|b^i_t|/\Delta\) are nondecreasing, we are not able to prove that one of them eventually reaches the level \(20mn\) in a strongly polynomial number of steps. This is since the set \(V \setminus T\) where the ratio increases might always consist only of nodes where \(|b^i_t|/\Delta\) is very small. The algorithm ENHANCED CONTINUOUS SCALING therefore introduces one additional subroutine, called FILTRATION. In case \(|b^i_t| < \Delta/n\) for every \(i \in (V \setminus T) - t\), we “tidy-up” the flow inside \(V \setminus T\), by performing a maximum flow computation here. This drastically reduces all relabeled excesses in \(V \setminus T\), and thereby guarantees that most iterations of the algorithm will have to increase certain \(|b^i_t|/\Delta\) values that are already at least \(1/n\).

In summary, the strongly polynomiality of our algorithm is based on the following three main new ideas.

- The definition of \(\Delta\)-feasible pairs, in particular, the condition on maintaining a security reserve \(R_i\). It is a cleaner and more efficient framework than similar ones in [10] and [30]; we believe this is the “real” condition a scaling type algorithm has to maintain.
- Continuous scaling, that guarantees that the ratios \(|b^i_t|/\Delta\) are nondecreasing during the algorithm. This is achieved by doing the exact opposite of [8, 10, 30].

\(^2\)However, to the extent of the author’s knowledge, no actual examples are known for these phenomena in any of the algorithms.
that use the natural analogue of the scaling technique for minimum cost circulations.

- The FILTRATION subroutine that intervenes in the algorithm whenever the nodes on a certain, relatively isolated part of the network have “unreasonably high” excesses as compared to the small node demands in this part.

2.4 The maximum flow subroutine

Standard maximum flow computation (see e.g. [2, Chapters 6-7]) will be a crucial subroutine in our algorithm. First and foremost, if we have an optimal labeling, then we show that an optimal solution to (P) can be obtained by computing a maximum flow. We now describe the subroutine Tight-Flow(S,μ), to perform such computations. In the weakly polynomial algorithm (Section 3), it will be used twice: at the initialization and at the termination of the algorithm, with S = V in both cases. However, it will also be the key part of the subroutine FILTRATION in the strongly polynomial algorithm (Section 4), also applied for subsets S ⊆ V.

The input of Tight-Flow(S,μ) is a node set S ⊆ V with t ∈ S, and a labeling μ, that is a feasible solution to (D) when restricted to S. The subroutine returns a generalized flow f from s to t on the network (S ∪ {s},E') with capacities ℓ and u.

Let us now turn to the case (ii) where the maximum flow problem is feasible, and returns a vector f'. Then f' is a feasible solution to (P) on S, and

\[ e_i^\mu(f') \leq n \max_{j \in S - i} \{ |b_j'| \} \quad \forall i \in S. \]

3. The CONTINUOUS SCALING ALGORITHM

The algorithm CONTINUOUS SCALING is shown on Figure 1. It starts with the subroutine INITIALIZE, that returns an initial flow f, along with a Δ = Δ-conservative labeling μ such that e_i^\mu < (d_i + 2)Δ holds for every i ∈ V. This subroutine takes the elementary step Elementary Step(T); Tight-Flow(V,μ);

Figure 1: Description of the weakly polynomial algorithm

is based on the MAXIMUM-MEAN-GAIN CYCLE-CANCELING algorithm as in [8, 22], a standard method also used e.g. in [12] the same way. The termination condition depends on the parameter B bounding the input description as described in the full version. After the while loops, the exact optimum solution can be obtained via the subrouting Tight-Flow(V,μ) described in the previous section. This is essentially the standard termination method used in all previous generalized flow algorithms.

The main part of the algorithm (the while loop) consists of iterations. The value of the scaling parameter Δ is monotone decreasing and all μ_i values are monotone increasing during the algorithm. In every iteration, a Δ-feasible pair (f,μ) is maintained.

The set N always denotes the set of nodes with e_i^\mu < (d_i + 1)Δ, and T_0 will consist of a certain set of nodes (but not all) with e_i^\mu ≥ (d_i + 2)Δ. The set T will denote a set of nodes that can be reached from T_0 on a tight path in the Δ-fat graph E'_T(Δ). Both T_0 and T are initialized empty. Note that t ∈ N as we chose b_t such that e_t < 0 always holds.

Every iteration first checks whether N ∩ T = ∅. If yes, then nodes p ∈ T_0 and q ∈ N are picked connected by a tight path P in the Δ-fat graph. Δ units of relabeled flow is sent from p to q on P: that is, f_i^\mu is increased by Δμ_i for every ij ∈ P (if ij was a reverse arc, this means decreasing f_j^\mu by Δμ_j). The only e_i values that change are e_p and e_q. If the new value is e_i^\mu < (d_p + 2)Δ, then p is removed from T_0. The iteration finishes in this case by resetting T = T_0 (irrespective of whether p was removed or not).

Let us now turn to the case N ∩ T = ∅. If there is a node j ∈ V \ T connected by a tight arc in E'_T(Δ) to T, then we extend T by j, and the iteration terminates. Otherwise, the subroutine ELEMENTARY STEP(T) is called, described next.

3.1 The Elementary step subroutine

Let (f,μ) be a Δ-feasible pair for Δ > 0. Let T ⊆ V be a (possibly empty) set of nodes with e_i^\mu ≤ 4(d_i + 2)Δ for
Subroutine Elementary Step \((T)\)
\[α_1 \leftarrow \min \left\{ \frac{1}{μ_0} : ij \in E[T \setminus \emptyset] \right\};\]
\[α_2 \leftarrow \min \{α_1, α_2\};\]
\[Δ' \leftarrow \frac{Δ}{2};\]
for \(i \in T\) do \(μ_i' \leftarrow α_2 μ_i;\)
for \(i \in V \setminus T\) do \(μ_i' \leftarrow μ_i;\)
for \(j \in E\) do
\[\begin{align*}
&\text{if } ij \in F[Δ'] \cup E[V \setminus T], \text{ then } f_{ij}' \leftarrow \frac{f_{ij}}{μ_i'}; \\
&T_0' \leftarrow T_0 \cup \{i : i \in V \setminus T, \; e_i^0 = 4(d_i + 2)Δ\}; \\
&T \leftarrow T \cup T_0; \\
&\text{if } \exists j \in T_0 \setminus \{i : i \in V \setminus T, \; e_i' < (d_i + 2)Δ\} \text{ then } T_0 \leftarrow T_0 \setminus \{i : i \in V \setminus T, \; e_i' < (d_i + 2)Δ\};
\end{align*}\]
\(T \leftarrow T_0;\)

Figure 2: The Elementary Step subroutine

every \(i \in V\), with strict inequality whenever \(i \in V \setminus T\). The subroutine (Figure 2) performs the following modifications for some \(α > 1\). The \(μ_i\) values are multiplied by \(α\) for \(i \in T\), and left unchanged for \(i \in V \setminus T\). The new value of the scaling parameter is set to \(Δ' := Δ/α\). Finally, the flow on non-tight arcs \(ij \in F[Δ']\) and on all arcs \(ij \in E[V \setminus T]\) is divided by \(α\).

The value of \(α\) is chosen maximal such that the modified pair \((f', μ')\) is \(Δ'\)-feasible, and further the modified excess \(e_i'(f') \leq 4(d_i + 2)Δμ_i\) holds for every \(i \in V\). To guarantee the latter property, we need the following definitions for every \(i \in V\).

Let
\[\begin{align*}
F_1(i) &:= \delta^i \cap F[V \setminus T], \\
r_1(i) &:= \sum_{i \in F_1(i)} f_{ij}, \\
F_2(i) &:= \delta^i \cap F_1(i), \\
r_2(i) &:= \sum_{i \in F_2(i)} f_{ij}, \\
F_3(i) &:= \delta^i \cap (F[V \setminus T] \cup E[V \setminus T, T]), \\
r_3(i) &:= \sum_{i \in F_3(i)} f_{ij}, \\
F_4(i) &:= \delta^i \cap F_3(i), \\
r_4(i) &:= \sum_{i \in F_4(i)} f_{ij}.
\end{align*}\]

Note that \(F_1(i)\) and \(F_2(i)\) denote the set of those incoming and outgoing arcs where we wish to decrease the flow by a factor \(α\). Let us define
\[δ_i := \frac{4(d_i + 2)Δμ_i + r_3(i) - r_1(i)}{r_2(i) - r_4(i) - b_i}.\]

If the denominator is 0 then \(δ_i := \infty\) is set. One can show that the denominator is always nonnegative and the numerator is positive.

The subroutine (Figure 2) chooses the largest \(α\) that is smaller than all \(δ_i\) values for \(i \in V \setminus T\), and also α \(≤ 1/τ_0\) for all arcs \(ij \in E\) leaving the set \(T\), and performs the above described modifications. Nodes \(i\) with \(e_i' = 4(d_i + 2)Δ\) (that is, \(α = δ_i\)) are added to both \(T_0\) and \(T\). Finally, if \(e_i'\) drops below \((d_i + 2)Δ\) for some \(i \in T_0\), then all such nodes \(i\) will be removed from \(T_0\), and \(T\) is reset to \(T = T_0\).

3.2 Analysis

We shall prove the following running time bound:

**Theorem 3.1.** The algorithm Continuous Scaling can be implemented to find an optimal solution for the uncapacitated formulation \((P)\) in running time \(\max(θ/m + n \log n \log B), O(m^2n \log^3 n)\).

We can adapt the algorithm to work directly for the standard formulation \((P_0)\); one can obtain a running time bound \(O(m^2n \log n \log B)\), matching the one by Goldfarb et al. [12].

In what follows, we give a high level overview of the analysis. Let \(Δ'(τ)\) denote the value of the scaling factor at the beginning of the \(τ\)-th iteration; clearly, \(Δ'(1) ≥ Δ'(2) ≥ ... ≥ Δ'(τ)\). Let \(τ(\tau)\) denote the set \(T\) at the beginning of iteration \(τ\).

Let us classify the iterations into three categories. An iteration is shrinking, if \(T(τ) \setminus T(τ + 1) = \emptyset\). This happens whenever a path augmentation is performed, or if the subroutine Elementary Step is performed, and for some \(τ \in T_0\), the value of \(e_i'\) is decreased below \((d_i + 2)Δ\). The iteration \(θ\) is expanding, if \(T(θ) ⊆ T(θ + 1)\). This can either happen if the iteration only consists of extending \(T\) by adding new a node reachable by a tight arc in the \(Δ\)-fat graph, or if \(T_0\) is extended in Elementary Step, and no node from \(T_0\) is removed. An iteration that is neither shrinking nor expanding is called neutral. Note that in a neutral iteration we must perform Elementary Step, and further we must have \(T(θ) = T(θ + 1)\). It is easy to verify that the iteration following the neutral iteration \(θ\) must be either expanding or shrinking. The key lemma of the analysis is the following.

**Lemma 3.2.** For the starting value \(Δ'(1) = Δ\) and arbitrary integer \(τ ≥ 1\), we have
\[τ ≤ 26mn \log_2 \frac{Δ}{(τ + 1)}\]

Further, the total number of shrinking iterations among the first \(τ\) is at most
\[13m \log_2 \frac{Δ}{(τ + 1)}.\]

The proof relies on the following lemma:

**Lemma 3.3.** During the first \(τ\) iterations, a node \(i\) may enter the set \(T_0\) altogether at most \(log_2 \frac{Δ}{τ + 1}\) times.

**Proof (Sketch).** Consider a node \(i\) that leaves \(T_0\) in iteration \(τ\) and re-enters \(T_0\) in iteration \(τ'\). We show that \(Δ'(τ') ≤ 2 Δ'(τ)/2\). This can be proved by analyzing the changes of the quantity \(e_\iota/Δμ_i\). At the end of iteration \(τ\) it is smaller than \((d_i + 2)Δ\), whereas in iteration \(τ'\) it is \(4d_i + 8\); hence it must increase at least by a factor \(4\) during these iterations. The key of the proof is to track the change of \(e_\iota/Δμ_i\) when the subroutine Elementary Step is performed with value \(a\). If \(i \in T\) then \(e_\iota/Δμ_i\) may only decrease. If \(i \in V \setminus T\), then \(e_\iota/Δμ_i\) may increase, but the new value cannot exceed \(a^2 \max(e_\iota/Δμ_i, d_i)\).

Using this, one can derive Lemma 3.2 by analyzing the potential
\[\Psi := \sum_{i ∈ T_0} \frac{e_i}{Δμ_i} - (d_i + 1).\]

The value of \(Ψ\) cannot increase during the subroutine Elementary Step, and decreases by one in every shrinking...
4. **THE STRONGLY POLYNOMIAL ALGORITHM**

Given a Δ-feasible pair \((f, \mu)\), we say that an arc \(pq \in E\) is abundant, if \(f_{pq}^0 \geq 17m\Delta\). The importance of abundant arcs is that they must be tight in all dual optimal solutions. This is a corollary of the following theorem.

**Theorem 4.1.** Let \((f, \mu)\) be a Δ-feasible pair. Then there exists an optimal solution \(f^*\) such that

\[ ||f^* - f^0||_\infty \leq \text{Ex}_\infty(f) + (|F^0| + 1)\Delta.\]

The proof uses a standard flow decomposition method; it can also be derived from Lemma 5 in Radzik [23]. For the flow \(f^*\) in an iteration with scaling factor \(\Delta\), we have \(\text{Ex}_\infty(f) \leq \sum_{pq \in E} (d_i + 2\Delta) \leq (8m + 8n - 8)\Delta \leq (16m - 8)\Delta\). Further, \(|F^0| \leq m\). This gives the following corollary; the last part follows by primal-dual slackness conditions.

**Corollary 4.2.** Let \((f, \mu)\) be the Δ-feasible pair during the algorithm. If for an arc \(pq \in E\), \(f_{pq}^0 \geq 17m\Delta\), then \(f_{pq}^0 > 0\) for some optimal solution \(f^*\) to \((P)\). Consequently, \(\gamma_{pqf}^0 = \mu_q^0\) for every optimal solution \(\mu^*\) to \((D)\).

The following simple claim gives a sufficient condition on the existence of abundant arcs.

**Claim 4.3.** Assume that for some node \(i \in V - t\), \(|b_i^t| \geq 20m\Delta\) holds in a certain iteration of the algorithm. Then there exists an incoming or outgoing abundant arc incident to \(i\).

Once we identify an abundant arc \(pq\) in the Enhanced Continuous Scaling algorithm, we will be able to reduce our problem by contracting \(pq\). Consider the problem instance \((V, E, t, b, \gamma)\). The contraction of the arc \(pq\) returns a problem instance \((V', E', t', \gamma')\) with \(t' := t\). We only include the description for the case \(p \neq t\). Let \(V' = V \setminus \{p\}\), and add an arc \(ij \in E'\) if \(ij \in E\) and \(i, j \neq p\). For every arc \(ip \in E\), add an arc \(iq \in E'\), and for every arc \(pi \in E\), \(i \neq q\), add an arc \(qi \in E'\). Set the gain factors as \(\gamma_{ij}' := \gamma_{ij}\) if \(i, j \neq p\), \(\gamma_{ip}' := \gamma_{ip}/\gamma_{pq}\) and \(\gamma_{qi}' := \gamma_{pi}/\gamma_{pq}\). Let us set \(b_i' := b_i\) if \(i \neq q\), and \(b_q' := b_q + \gamma_{pq}b_p\). If parallel arcs are created, keep only one that maximizes the \(\gamma\) value. In the case \(p = t\), a similar construction is performed with \(V' = V \setminus \{q\}\). If conditions \((*)\) and \((***)\) hold for the original instance, they will also hold for the contracted instance.

We have now an overview of the algorithm Enhanced Continuous Scaling. The while loop proceeds very similarly to Continuous Scaling, with the addition of the special subroutine Filtration, described in Section 4.1; also termination will be different. Our goal is to find a node \(i \in V - t\) with \(|b_i^t| \geq 20m\Delta\) as in Claim 4.3. There must be an abundant arc incident to such a node that we can contract and restart the algorithm in the smaller graph. (Instead of restarting from scratch, the implementation in the full version uses the flow \(f\) before the contraction to achieve better running time bounds.)

Consider the set

\[ D := \left\{ i \in V - t : |b_i^t| \geq \frac{\Delta}{n} \right\}. \] (3)

Our aim is to guarantee that most iterations when \(\Delta\) is multiplied by \(\alpha\) will multiply \(|b_i^t|/\Delta\) by \(\alpha\) for some \(i \in D\). This will ensure that \(|b_i^t| \geq 20m\Delta\) happens within \(O(mn \log n)\) number of steps. Recall that in the subroutine ELEMENTARY STEP\((T)\), the \(|b_i^t|/\Delta\) ratio is multiplied by \(\alpha\) for all nodes \(i \in V \setminus T\) and remains unchanged for \(i \in T\). To guarantee the above property, modify the while loop of CONTINUOUS SCALING as follows. If \((V \setminus T) \cap D \neq \emptyset\), ELEMENTARY STEP\((T)\) is performed identically. If \((V \setminus T) \cap D = \emptyset\), then before ELEMENTARY STEP\((T)\), the special subroutine Filtration\((V \setminus T)\) is executed, described as follows.

4.1 The Filtration subroutine

This subroutine is executed before ELEMENTARY STEP\((T)\) if \(|b_i^t| < \Delta/(16m)\) holds for all \(i \in (V \setminus T) - t\).

**Filtration\((V \setminus T)\)** (Figure 3) performs the subroutine Tight Flow\((V \setminus T, \mu)\), as described in Section 2.4. This replaces \(f\) by an entirely new flow \(f^*\) on the arcs in \(E(V \setminus T)\). We further set \(f_{ij} = 0\) on all arcs entering \(T\), and keep the original \(f\) value on all other arcs (that is, arcs in \(E(T) \cup E(T, V \setminus T)\)).

This might decrease \(e_i^t\) values below \((d_i + 2)\Delta\) for some \(i \in T_0\). In this case, we remove all such nodes from \(T_0\), reset \(T = T_0\), and jump to the next iteration without performing ELEMENTARY STEP\((T)\). Similarly, if \(e_i^t < (d_i + 1)\Delta\) for any \(i \in T\), that is, \(i\) is added to the set \(N \cap T\), then we do not perform ELEMENTARY STEP\((T)\) in this iteration.

**Figure 3: The Filtration subroutine**

4.2 The Enhanced Continuous Scaling Algorithm

We are ready to describe our strongly polynomial algorithm, shown on Figure 4. The algorithm consists of iterations similar to CONTINUOUS SCALING, with the addition of the above described Filtration subroutine.

The termination criterion is not on the value of \(\Delta\), but on the size of the graph: we terminate once it is reduced to a single node. The main progress is done when an abundant arc \(pq\) appears: we contract it and restart on the smaller graph. At termination, the subroutine Expand-to-Original finds an optimal primal and dual solution in the original graph. This is done by first expanding all contracted arcs \(pq\) in the
Enhanced Continuous Scaling

**Algorithm** Enhanced Continuous Scaling

**Input:** \((V, E, t, b, \gamma)\)

**Initialize:**

\(T_0 \leftarrow \emptyset; \ T \leftarrow \emptyset;\)

\(k \leftarrow 0;\)

**While** \(|V| > 0\) **do**

\(N \leftarrow \{i \in V; \ e_i^T < (d_i + 1)\Delta\};\)

if \(N \cap T \neq \emptyset\) **then**

- **pick** \(p \in T_0, q \in N\) connected by a tight path \(P\) in \(E_0^T(\Delta)\);
- **send** \(\Delta\) units of relabeled flow from \(p\) to \(q\) along \(P\);
- **if** \(e_p^T < (d_p + 2)\Delta\) **then** 
  \(T_0 \leftarrow T_0 \setminus \{e_p\}; \)

**else**

- **if** \(\exists ij \in E_0^T(\Delta); \ \gamma_{ij}^T = 1, i \in T, j \in V \setminus T\) **then** 
  \(T \leftarrow T \cup \{j\};\)

- **else** **if** \(\forall i \in (V \setminus T) - t: |b_i^T| < \frac{\Delta}{e}\) **then** 
  **Filtration** \((V \setminus T)\); 
  **Elementary step** \((T)\);
- **if** there exist \(pq \in E; f_{pq}^T \geq 17m\Delta\) **do** 
  **Contract** \((pq)\);
- **if** \(|V| = 1\) **then** **terminate**.

**Enh. Cont. Scal.** \((V^t, E^t, t^t, b^t, \gamma^t)\)

**Expand to Original**

**Tight-flow** \((V, \mu)\)

---

Figure 4: Description of the strongly polynomial algorithm

---

reverse order of contraction. Hence we obtain a dual optimal solution \(\mu\) in the original. Finally, the subroutine **Tight-flow** \((V, \mu)\) obtains a primal optimal solution, as guaranteed by Theorem 2.6(i).

**Theorem 4.4.** *The algorithm Enhanced Continuous Scaling finds an optimal solution for the uncapacitated formulation (P) in running time \(O(n^2m^2)\) elementary arithmetic operations and comparisons.*

The running time bound given in the theorem is for a slightly more complicated variant described in the full version, when instead of restarting the entire algorithm after contraction, we use the contracted image of the actual flow \(f\). The same analysis for the current form would give a running time bound worse by a factor \(\log n: O(n^2m^2 \log n)\).

To get a truly strongly polynomial algorithm, we also need to guarantee that the size of the numbers during the computations remain polynomially bounded. A modified version of the algorithm incorporating additional rounding steps is given in the full version.

The analysis builds on the analysis of the weakly polynomial algorithm in Section 3. The key new element is tracking nodes in the set \(D\) of nodes with large \(b_i^T\) values (see (3)). If \((V \setminus T) - t\) is nonempty when **Elementary step** is performed with a factor \(\alpha\), then one of the \(b_i^T\) in \(\Delta\) will be multiplied by \(\alpha\). Otherwise, **Filtration** “cleans up” the flow on \((V \setminus T) - t\). According to Theorem 2.6(ii), after this subroutine we will have

\[
e_i^T(f) \leq n \max_{j \in (V \setminus T) - t} |b_j^T| \quad \forall i \in V \setminus T.
\]

We perform **Elementary step** after this subroutine. Then either we pick a large enough \(\alpha\) such that some node in \((V \setminus T) - t\) enters \(D\), or in the next step the entire \((V \setminus T)\) will be contained in \(N\), and will be reachable from \(T\). Consequently, a path augmentation will be performed within two steps (one more iteration may be needed to extend \(T\)). In the first case \(D\) is extended; since a node cannot leave \(D\), this can only happen \(n - 1\) times altogether. In the second case one of the subsequent two iterations will be shrinking, which also guarantees progress.

---

5. CONCLUSION

We have given a strongly polynomial algorithm for the generalized flow maximization problem. A natural next question is to address the minimum cost generalized flows. As noted in the Introduction, this problem is equivalent to solving LPs with two nonzero entries per column (see Hochbaum [15]).

In contrast to the vast literature on the flow maximization problem, there is only one weakly polynomial combinatorial algorithm known for this setting, the one by Wayne [31]. This setting is more challenging since the dual structure cannot be characterized via the convenient relabelling framework, and thereby most tools for minimum cost circulations, including the scaling approach also used in this paper, become difficult if not impossible to apply.

**Acknowledgment**

The author is grateful to Joseph Cheriyan and Ian Post for several suggestions that helped to improve the presentation, and for pointing out a bug in the previous version.

6. REFERENCES

[1] I. Adler and S. Cosares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39(6):955–960, 1991.

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

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

[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] F. Glover and D. Klingman. On the equivalence of some generalized network problems to pure network problems. *Mathematical Programming*, 4(1):269–278, 1973.

[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] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimizations*. Springer-Verlag, 1993.

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

[16] W. S. Jewell. Optimal flow through networks. *Operations Research*, 10:476–499, 1962.

[17] 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.

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

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

[20] K. Onaga. Optimum flows in general communication networks. *Journal of the Franklin Institute*, 283(4):308–327, 1967.

[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] M. Shigeno. A survey of combinatorial maximum flow algorithms on a network with gains. *Journal of the Operations Research Society of Japan*, 47:244–264, 2004.

[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] K. Truemper. On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, 32(2):450–456, 1977.

[30] L. A. Végh. Concave generalized flows with applications to market equilibria. *Mathematics of Operations Research*, 2014. (to appear).

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