New Algorithms for Generalized Network Flows

Edith Cohen* Nimrod Megiddo†

Revised; March 1993

Abstract

This paper[1] is concerned with generalized network flow problems. In a generalized network, each edge $e = (u, v)$ has a positive "flow multiplier" a_e associated with it. The interpretation is that if a flow of x_e enters the edge at node u, then a flow of $a_e x_e$ exits the edge at v.

The uncapacitated generalized transshipment problem (UGT) is defined on a generalized network where demands and supplies (real numbers) are associated with the vertices and costs (real numbers) are associated with the edges. The goal is to find a flow such that the excess or deficit at each vertex equals the desired value of the supply or demand, and the sum over the edges of the product of the cost and the flow is minimized. Adler and Cosares [1] reduced the restricted uncapacitated generalized transshipment problem, where only demand nodes are present, to a system of linear inequalities with two variables per inequality. The algorithms presented in [5] result in a faster algorithm for restricted UGT.

Generalized circulation is defined on a generalized network with demands at the nodes and capacity constraints on the edges (i.e., upper bounds on the amount of flow). The goal is to find a flow such that the excesses at the nodes are proportional to the demands and maximized. We present a new algorithm that solves the capacitated generalized flow problem by iteratively solving instances of UGT. The algorithm can be used to find an optimal flow or an approximation thereof When used to find a constant factor approximation, the algorithm is not only more efficient than previous algorithms but also strongly polynomial. It is believed to be the first strongly polynomial approximation algorithm for generalized circulation. The existence of such an approximation algorithm is interesting since it is not known whether the exact problem has a strongly polynomial algorithm.

*AT&T Bell Laboratories, Murray Hill, NJ. Research was done while the first author was attending Stanford University and IBM Almaden Research Center. Research partially supported by ONR-N00014-91-C-0026 and by NSF PYI Grant CCR-8858097, matching funds from AT&T and DEC.

†IBM Research, Almaden Research Center, San Jose, CA 95120-6099, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel. Research partially supported by ONR-N00014-91-C-0026.

[1]A preliminary version appeared in ISTCS'92 [4]

1. Introduction

A *generalized network* is a digraph $G = (V, E)$ given together with positive *flow multipliers* a_e $(e \in E)$ associated with the edges. The multiplier a_e $(e \in E)$ is interpreted as a gain factor (when $a_e > 1$) or a loss factor (when $a_e < 1$) of flow along the edge e; if x_e units of flow enter the edge e, then $a_e x_e$ units exit. Generalized network flows are also known in the literature as *flows with gains*. They can be used to model many situations that arise in financial analysis [8, 9, 12].

The uncapacitated generalized transshipment problem (UGT) is defined on a generalized network, where costs are given for the edges and supplies or demands are given for the nodes. The goal is to find a flow of minimum cost, which satisfies the supply and demand requirements. Adler and Cosares [1] gave an algorithm for solving restricted instances of UGT where there are many sources and no sinks. Their algorithm is based on solving the dual linear programming problem. In this case the dual problem has two variables per inequality (a TVPI system) and also has a special property, which we call *monotonicity*, namely, in each inequality there is at most one positive and at most one negative coefficient. Hence, the results of [5] imply better time bounds for restricted UGT.

In the generalized circulation problem (GC) we consider a generalized network where (nonnegative) demands are associated with the nodes and capacities are associated with the edges. The goal is to find a feasible flow which maximizes the fraction of the satisfied demand. Goldberg, Plotkin, and Tardos [9] presented an algorithm for the more general capacitated generalized transshipment problem without costs. Their algorithm is based on solving an instance of GC with a single supply node, the *source*, and performs additional computation of $O(mn)$ time. We present a scheme for solving generalized circulation problems by iteratively relaxing the capacity constraints. An iteration features (i) solving an instance of UGT on the same network with costs chosen with respect to the capacities, (ii) scaling the flow to a feasible one, and (iii) replacing the capacities by the residual capacities calculated relative to the latter flow.

Our scheme introduces a general method of approximating a solution to linear programming problems of the following form:

$$(P) \qquad \begin{array}{rl} \text{Maximize} & t \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = t\boldsymbol{b} \\ & \boldsymbol{U}\boldsymbol{x} \le \boldsymbol{d} \\ & \boldsymbol{x} \ge \boldsymbol{0} \ , \end{array}$$

where $\boldsymbol{A} \in R^{n\times m}$, $\boldsymbol{O} \le \boldsymbol{U} \in R^{\ell\times m}$, $\boldsymbol{b} \in R^n$, and $\boldsymbol{0} \le \boldsymbol{d} \in R^{\ell}$. The system $\boldsymbol{U}\boldsymbol{x} \le \boldsymbol{d}$ may be viewed as a set of generalized capacity constraints. Denote by t^* the maximum of (P). Suppose that for $\boldsymbol{c} \ge \boldsymbol{0}$ the following problem is relatively easy:

$$(E) \qquad \begin{array}{rl} \text{Minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \ . \end{array}$$

We will show that by solving a single instance of (E) (with a suitable $\boldsymbol{c}$) a feasible solution $(\boldsymbol{x}', t') \in R^{n+1} \times R$ of (P) can be found such that $t' \geq t^*/\ell$.

Consider the generalized circulation problem with the relaxed goal of computing a flow which satisfies a fraction of the demand which approximates within a constant factor the best attainable fraction. For the relaxed problem, the scheme described above yields a strongly polynomial time algorithm, which we also believe is the fastest known algorithm in a certain range of the input parameters. This scheme also yields an algorithm for obtaining an optimal solution, which is the fastest known for a certain range of the input parameters.

In Section 2 we define the UGT problem and review the algorithm of Adler and Cosares [1]. In Section 3 we introduce the approximation algorithm and apply it to the generalized circulation problem. In Section 4 we introduce bidirected generalized networks and discuss the UGT and generalized circulation problems on these networks. Section 5 contains concluding remarks.

Note that for instances of the problems mentioned above we need to consider cases where $m = \Omega(n^2)$. The algorithms presented here handle multiple edges within the stated time bounds.

2. The generalized transshipment problem

Given a graph $G = (V, E)$, for every $i \in V$ we denote by $\mathrm{in}(i)$ and $\mathrm{out}(i)$ the sets of edges that go into and out of i, respectively.

Problem 2.1 [Uncapacitated Generalized Transshipment (UGT)]
Given is a generalized network, consisting of a graph $G = (V, E)$ with flow multipliers a_e and edge-costs c_e $(e \in E)$, and supplies (or demands) b_i $(i \in V)$. Find a flow function $\boldsymbol{x} = (x_e)_{e \in E}$ to solve the following:

$$\begin{aligned} \text{Minimize} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & \sum_{e \in \mathrm{in}(i)} a_e x_e - \sum_{e \in \mathrm{out}(i)} x_e = b_i \quad (i \in V) \\ & x_e \geq 0 \quad (e \in E)\ . \end{aligned}$$

When $b_i < 0$ (resp., $b_i > 0$), we call i a *sink* (resp., *source*). The dual linear programming problem can be stated as follows. Find $\pi_1, \ldots, \pi_n$ to solve the following:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} b_i \pi_i \\ \text{subject to} \quad & \pi_i - a_e \pi_j \leq c_e \quad (e = (i, j) \in E)\ . \end{aligned}$$

Note that the set of constraints of the dual problem is monotone in the sense defined above.

In this section we consider restricted instances of UGT where either there are only sinks ($\boldsymbol{b} \geq \boldsymbol{0}$) or there are only sources ($\boldsymbol{b} \leq \boldsymbol{0}$). Adler and Cosares [1] proposed a scheme for solving a subclass of linear programming problems in standard form[2] where each variable appears in at most two equations. In particular, that scheme is applicable to restricted UGT instances. They showed that these instances can be solved by using one application of Megiddo's algorithm for TVPI systems [13]. An application of the faster algorithms for TVPI systems presented in [5] can be used instead. Hence, restricted UGT instances can be solved in $O\left(mn^2(\log m + \log^2 n)\right)$ time, deterministically, and in $O\left(n^3 \log n + mn(\log m \log^3 n + \log^5 n)\right)$ expected time when using randomization.

We now characterize the problems to which the scheme of [1] applies. Consider a linear programming problem in standard form

$$(SF) \qquad \begin{aligned} \text{Minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

where each column of $\boldsymbol{A} \in R^{n \times m}$ contains at most two non-zeros. Denote by $\boldsymbol{x}^* \in R^m$ an optimal solution of (SF). Note that the dual of (SF) amounts to maximizing an arbitrary objective function subject to a TVPI system. Consider the problems

$$(SF_i) \qquad \begin{aligned} \text{Minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = b_i \boldsymbol{e}^i \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

$(i = 1, \ldots, n)$ and[3] suppose $\boldsymbol{x}^{(i)}$ is an optimal solution of (SF_i), $i = 1, \ldots, n$. The scheme of [1] applies to (SF) if $\boldsymbol{x}^* = \sum_{i:b_i \neq 0} \boldsymbol{x}^{(i)}$.

We now sketch the ideas used in the scheme of [1]. Let $\pi_i^{\min}$ (resp., $\pi_i^{\max}$) denote the minimum (resp., maximum) value of π_i subject to the TVPI system of constraints $\boldsymbol{A}^T \boldsymbol{\pi} \leq \boldsymbol{c}$. If $\pi_i^{\max} = \infty$ (resp., $\pi_i^{\min} = -\infty$), then (SF) is feasible only if $b_i \leq 0$ (resp., $b_i \geq 0$). If $b_i \neq 0$, a vector $\boldsymbol{x}^{(i)}$ as defined above can be constructed from a minimal subset of the dual constraints which implies (i) $\pi_i \leq \pi_i^{\max}$ if $b_i > 0$, or (ii) $\pi_i \geq \pi_i^{\min}$ if $b_i < 0$. The edges which correspond to such a minimal system comprise a generalized augmenting path [10] of flow into node i (i.e., a flow generating cycle and a path from a node on the cycle to node i). The vector $\boldsymbol{x}^{(i)}$ is obtained by adjusting the flow values on the edges while pushing flow along this augmenting path.

It is easy to see that the scheme of [1] applies to restricted UGT instances. Consider a UGT instance where $\boldsymbol{b} \geq \boldsymbol{0}$ (the arguments are similar for the case of $\boldsymbol{b} \leq \boldsymbol{0}$). The system of dual constraints is monotone, so by a single application of the algorithm of [5] we determine $\pi_i^{\max}$ for $i = 1, \ldots, n$ (see the sections on monotone systems in [5]). If $\pi_i^{\max} < \infty$, the algorithm also computes a minimal subset of dual constraints which implies that $\pi_i \leq \pi_i^{\max}$. The vector $\boldsymbol{x}^{(i)}$ can be constructed using this information. If $\pi_i^{\max} = \infty$ and $b_i > 0$, the UGT system is not feasible. Otherwise, $\boldsymbol{x}^* = \sum_{i:b_i > 0} \boldsymbol{x}^{(i)}$ is a solution. It is also apparent why the scheme of [1] is not applicable to unrestricted UGT instances, that is, when the vector $\boldsymbol{b}$

[2] In the standard form the constraints are in the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$.

[3] $\boldsymbol{e}^i$ is such that $e^i_j = 0$, $i \neq j$ and $e^i_i = 1$.

Vaidya, 89 [17]	$O(n^2 m^{1.5} \log(n\gamma))$
Kapoor and Vaidya, 88 [11]	$O(n^{2.5} m^{1.5} \log(n\gamma))$
Goldberg, Plotkin and Tardos, 88 [9]	$O(n^2 m^2 \log n \log^2 \gamma)$ $O(n^2 m(m + n \log n) \log n \log \gamma)$

Table 1: Some previous results on generalized circulation

is general. For unrestricted instances, the problem is not "separable" in the sense that the optimal solution for the right hand-side vector $\boldsymbol{b}$ is not necessarily a sum of the solutions of the simpler systems (SF_i) $(i = 1, \ldots, n)$.

3. Generalized circulation

Definition 3.1 Consider a generalized network $G = (V, E)$, where demands $b_i \geq 0$ $(i = 1, \ldots, n)$ are associated with the nodes and capacities $c_{ij} \geq 0$ (possibly $c_{ij} = \infty$) are associated with the edges.

i. A *generalized flow* is a nonnegative flow function $\boldsymbol{x} = (x_e)_{e \in E}$ for which there exists a scalar $\hat{t} = \hat{t}(\boldsymbol{x})$ such that

$$\sum_{e \in \text{in}(i)} a_e x_e - \sum_{e \in \text{out}(i)} x_e = \hat{t} b_i \quad (i = 1, \ldots, n)$$

i.e., the flow $\boldsymbol{x}$ satisfies a fraction $\hat{t}$ of the demand.

ii. A generalized flow is said to be *feasible* if

$$x_e \leq c_e \quad (e \in E) .$$

Problem 3.2 [Generalized Circulation]
Given is a generalized network with demands and capacities as above. Find a feasible generalized flow $\boldsymbol{x}^*$ that maximizes $\hat{t}(\boldsymbol{x}^*)$. Denote $t^* \equiv \hat{t}(\boldsymbol{x}^*)$.

We refer to t^* as the *optimal* value. A feasible generalized flow $\boldsymbol{x}$ is said to be *optimal* if $\hat{t}(\boldsymbol{x}) = t^*$, and $(1 - \epsilon)$-*optimal* if $\hat{t}(\boldsymbol{x}) \geq (1 - \epsilon)t^*$.

Vaidya [17] gave an $O(n^2 m^{1.5} \log(n\gamma))$ time algorithm for the problem, where γ is an upper bound on all the numerators and denominators of the capacities, multipliers, and costs. Vaidya's bound is based on a specialization of his currently fastest known general purpose linear programming algorithm and relies on the theoretically fast matrix multiplication algorithms. The previously fastest known algorithm, due to Kapoor and Vaidya [11], does not rely on fast matrix multiplication, and has a bound worse by a factor of $\sqrt{n}$. A recent result by Murray [14] is based on a different specialization of Vaidya's linear programming algorithm to generalized flow. Murray's generalized circulation algorithm matches the bound of [17] and does not rely on fast matrix multiplication. The algorithms of [11, 14, 17] are applicable to the more general min-cost generalized flow problem. Different algorithms, of a somewhat more combinatorial nature, were given by Goldberg, Plotkin and Tardos [9]. These results are summarized in Table 1.

3.1. A generalized circulation algorithm

The algorithms discussed above are designed to find an optimal flow. We present an algorithm for Problem 3.2 which is based on iteratively obtaining closer and closer approximations of the optimal solution. In each iteration, the algorithm computes a $(1/m)$-optimal flow and then computes the residual network. The subsequent iteration is applied to the residual network. The residual network is constructed by updating the capacities in a similar way as for regular network flows: Consider a network with capacities c_e $(e \in E)$ and denote by f_e $(e \in E)$ the value of the current flow on an edge e. Consider a pair of edges $e = (u, v)$ and $e' = (v, u)$ where $a_e a_{e'} = 1$. Note that we can always assume that either $f_e = 0$ or $f_{e'} = 0$. Suppose that $f_{e'} = 0$. The residual capacity of e is $c_e - f_e$ and the residual capacity of e' is $c_{e'} + a_e f_e$. A solution of Problem 3.2 on the original network can be obtained by adding the current flow values f_e $(e \in E)$ to the flow values on the corresponding edges of a solution of Problem 3.2 for the residual network. Thus, after k iterations, the algorithm produces a flow which is $(1 - (1 - 1/m)^k)$-optimal. Hence, a $(1 - \epsilon)$-optimal flow can be computed in $O(m \log \epsilon^{-1})$ iterations. Note that $O(m)$ iterations suffice for any constant ϵ. The optimal value can be found within $O(m|t^*|)$ iterations, where $|t^*| \leq m \log(n\gamma)$ is a bound on the number of bits in the optimal solution and γ is the maximum value of any numerator or denominator of a capacity or a multiplier in the network.

What remains is to show how we perform each iteration. In Subsection 3.2 we introduce an approximation method that allows us to find a $(1/m)$-optimal feasible generalized flow by solving a single UGT instance on the same generalized network, where the capacity constraints are relaxed and costs are introduced. We will see that each iteration of our algorithm amounts to solving an instance of the restricted UGT problem.

The resulting deterministic and randomized bounds for computing a $(1 - \epsilon)$-optimal generalized circulation are summarized in Table 2. We believe our algorithm is more practical than that of [17]. When the algorithm is used to find an approximate solution, we achieve strongly polynomial time bounds which are also strictly better than those of [9, 11], and better than those of [17] in a certain range of the parameters (e.g., when the size of the binary encoding of capacities and multipliers is large).

Note that when $\epsilon = 1/q(m, n)$, for some polynomial q, a $(1 - \epsilon)$-optimal flow can be found in strongly polynomial time. It is still not known whether a strongly polynomial time algorithm exists for finding an optimal solution. This question is of a particular interest because generalized circulation is one of the simplest classes of linear programming problems for which no strongly polynomial algorithm is yet known [9, 16].

We note that there does not seem to be an obvious way of obtaining a strongly polynomial bound to the approximate problem by adding appropriate stopping rules to previous

Computing a $(1-\epsilon)$-optimal flow:

expected time	$O\left(m \log \epsilon^{-1}\left(n^3 \log n + mn(\log m \log^3 n + \log^5 n)\right)\right)$
deterministic	$O\left(m^2 n^2 \log \epsilon^{-1}(\log m + \log^2 n)\right)$

Computing the optimal solution:

expected time	$\tilde{O}\left(\lvert t^*\rvert(mn^3 + m^2 n)\right)$
deterministic	$\tilde{O}(\lvert t^*\rvert m^2 n^2)$

Table 2: Bounds for generalized circulation

algorithms. Primal-dual interior point algorithms work in iterations where each (strongly-polynomial) iteration reduces by a constant factor the duality gap. This implies that the gap between the value of the current solution and the optimal value tends to zero geomterically. The initial gap may, however, be very large with respect to the optimal value and there is no known way to obtain a sufficiently small initial gap. One of the combinatorial algorithms presented by Goldberg, Plotkin, and Tardos [9], the fat-path algorithm, is an iterative approximation scheme such that each iteration reduces by a factor of $1/n$ the gap between the excess at the source of the current pseudo-flow and the optimal value. Each iteration, however, is not strongly polynomial (more specifically, the cancellation of flow-generating cycles is not performed in strongly-polynomial time). Moreover, the initialization is such that the value of the gap is γ^2, and hence, a strongly polynomial number of iterations may not be sufficient to obtain a solution within a constant factor of the optimal value.

3.2. Obtaining an approximation

Consider linear programming problems of the following form:

$$\begin{aligned} \text{Maximize}\ & t \\ \text{subject to}\ & \boldsymbol{A}\boldsymbol{x} = t\boldsymbol{b} \\ & \boldsymbol{U}\boldsymbol{x} \leq \boldsymbol{d} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

where $\boldsymbol{A} \in R^{n\times m}$, $\boldsymbol{O} \leq \boldsymbol{U} \in R^{\ell\times m}$, $\boldsymbol{b} \in R^n$, and $\boldsymbol{0} \leq \boldsymbol{d} \in R^{\ell}$. We refer to the constraints $\boldsymbol{U}\boldsymbol{x} \leq \boldsymbol{d}$ as *generalized capacity constraints.* A vector $\boldsymbol{x} \geq \boldsymbol{0}$, such that $\boldsymbol{A}\boldsymbol{x} \propto \boldsymbol{b}$ ($\boldsymbol{A}\boldsymbol{x}$ is proportional to $\boldsymbol{b}$) and $\boldsymbol{U}\boldsymbol{x} \leq \boldsymbol{d}$ is called *feasible.* For a feasible vector $\boldsymbol{x}$, denote by $\hat{t}(\boldsymbol{x})$ the scalar $\hat{t}$ such that $\boldsymbol{A}\boldsymbol{x} = \hat{t}\boldsymbol{b}$. Denote by $(\boldsymbol{x}^*, t^*)$ an optimal solution (so that $t^* = \hat{t}(\boldsymbol{x}^*)$). A feasible vector $\boldsymbol{x}$ is $(1-\epsilon)$-*optimal* if $\hat{t}(\boldsymbol{x}) \geq (1-\epsilon)t^*$.

Suppose that for $\boldsymbol{0} \leq \boldsymbol{c} \in R^m$ it is relatively easy to compute a vector $\boldsymbol{x} \geq \boldsymbol{0}$ which minimizes $\boldsymbol{c}^T\boldsymbol{x}$ subject to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. We refer to problems of this form as *uncapacitated* instances, whereas an instance of the original problem is referred to as a *capacitated* one. Note that when the capacitated problem is an instance of generalized circulation, $\boldsymbol{U}$ is a diagonal matrix with at most $m = |E|$ rows. The corresponding uncapacitated problem is an instance of UGT on the same network, where only demand nodes are present.

We present an algorithm for constructing a $(1/\ell)$-optimal vector $\boldsymbol{y}$. The algorithm amounts to solving a single instance of the uncapacitated problem. Let $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{d})$, i.e., $\boldsymbol{D}$ is a diagonal matrix with the coordinates of $\boldsymbol{d}$ in its diagonal. Let $\boldsymbol{p}^T = \boldsymbol{e}^T \boldsymbol{D}^{-1} \boldsymbol{U}$, where $\boldsymbol{e}$ is a vector of 1's, and consider the linear cost function $\boldsymbol{p}^T \boldsymbol{x}$. Note that $\boldsymbol{p}^T = \sum_{i=1}^{\ell} \boldsymbol{U}_{i\bullet}/d_i$. It is easy to verify that the following properties hold:

i. If $\boldsymbol{x}$ is feasible then $\boldsymbol{p}^T \boldsymbol{x} \leq \ell$.

ii. If $\boldsymbol{A}\boldsymbol{x} \propto \boldsymbol{b}$, $\boldsymbol{x} \geq \boldsymbol{0}$, and $\boldsymbol{p}^T \boldsymbol{x} \leq 1$, then $\boldsymbol{x}$ is feasible.

iii. If $\boldsymbol{p}^T \boldsymbol{x}^* > 0$, then $\boldsymbol{p}^T \boldsymbol{x}^* \geq 1$ (since for some i, $\boldsymbol{U}_{i\bullet} \boldsymbol{x}^* = d_i$).

Consider a vector $\boldsymbol{y} \geq \boldsymbol{0}$ such that $\boldsymbol{A}\boldsymbol{y} \propto \boldsymbol{b}$ and $\boldsymbol{p}^T \boldsymbol{y} = 1$ which maximizes $\hat{t}(\boldsymbol{y})$. Note that the vector $\ell \boldsymbol{y}$ maximizes $\hat{t}(\boldsymbol{x})$ among all vectors $\boldsymbol{x} \geq \boldsymbol{0}$ such that $\boldsymbol{A}\boldsymbol{x} \propto \boldsymbol{b}$ and $\boldsymbol{p}^T \boldsymbol{x} \leq \ell$. In particular, $\hat{t}(\ell \boldsymbol{y}) \geq t^*$, and hence, $\hat{t}(\boldsymbol{y}) \geq t^*/\ell$. Such a vector $\boldsymbol{y}$ can be obtained by normalizing a vector $\boldsymbol{x} \geq \boldsymbol{0}$ which minimizes $\boldsymbol{p}^T \boldsymbol{x}$ subject to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Also, $\boldsymbol{y}$ is feasible and therefore provides the desired approximation. A formal description of the algorithm follows.

Algorithm 3.3 [Compute a $(1/\ell)$-optimal vector]

i. Solve the following instance of the uncapacitated problem:

$$\begin{aligned} \text{Minimize } & \boldsymbol{p}^T \boldsymbol{x} \\ \text{subject to } & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \,. \end{aligned}$$

If it is infeasible, then conclude that $\boldsymbol{x} = \boldsymbol{0}$ is the only feasible vector of the capacitated instance. Otherwise, let $\boldsymbol{x}$ be the solution.

ii. If $\boldsymbol{p}^T \boldsymbol{x} = 0$ then for every $r \geq 0$, the vector $r\boldsymbol{x}$ is feasible, and hence the capacitated problem is unbounded.

iii. Otherwise, when $\boldsymbol{p}^T \boldsymbol{x} \neq 0$, compute the largest number r (which must exist in this case) such that $r\boldsymbol{U}\boldsymbol{x} \leq \boldsymbol{d}$. Conclude that $r\boldsymbol{x}$ is $(1/\ell)$-optimal.

Correctness. Consider the vector $\boldsymbol{x}$ computed in step i of the algorithm. Note that $\hat{t}(\boldsymbol{x}) = 1$. Hence, $\hat{t}(r\boldsymbol{x}) = r\hat{t}(\boldsymbol{x}) = r$.

Proposition 3.4 $\boldsymbol{p}^T \boldsymbol{x} = 0$ *if and only if the capacitated problem is unbounded.*

Proof: Suppose first that $\boldsymbol{p}^T \boldsymbol{x} = 0$. Note that for all $r \geq 0$, $r\boldsymbol{p}^T \boldsymbol{x} = 0$, and hence, $r\boldsymbol{x}$ is feasible. Also, for all $r \geq 0$, $\hat{t}(r\boldsymbol{x}) = r$. Hence, the problem is unbounded. Second, suppose the problem is unbounded. There exists a vector $\boldsymbol{x}$ such that $r\boldsymbol{x}$ is feasible for all $r \geq 0$. It follows that $\boldsymbol{p}^T(r\boldsymbol{x}) = r\boldsymbol{p}^T \boldsymbol{x} \leq \ell$ for all $r \geq 0$. Hence, $\boldsymbol{p}^T \boldsymbol{x} = 0$. ∎

The following proposition concludes the correctness proof.

Proposition 3.5 *If t^* is finite, then $r \geq t^*/\ell$.*

Proof: For $k \geq 0$, denote

$$\begin{aligned} R(k) &= \max\{\hat{t}(\boldsymbol{y}) \mid \boldsymbol{p}^T\boldsymbol{y} \leq k, \boldsymbol{A}\boldsymbol{y} \propto \boldsymbol{b}, \boldsymbol{y} \geq \boldsymbol{0}\} \text{ , and} \\ R^*(k) &= \max\{\hat{t}(\boldsymbol{y}) \mid \boldsymbol{p}^T\boldsymbol{y} \leq k, \boldsymbol{y} \text{ is feasible}\} \ . \end{aligned}$$

Obviously, (i) R and R^* are increasing functions, (ii) $R \geq R^*$, (iii) for every $a > 0$, $R(ak) = aR(k)$, and (iv) $t^* \leq R^*(\ell)$.

For the vector $\boldsymbol{x}$ (computed in step i), $R(\boldsymbol{p}^T\boldsymbol{x}) = 1$, and hence $R(r\boldsymbol{p}^T\boldsymbol{x}) = r$. Since $\boldsymbol{p}^T\boldsymbol{x} > 0$, it follows that $r\boldsymbol{U}_{i\bullet}\boldsymbol{x} = d_i$ for some $1 \leq i \leq \ell$. Hence, $r\boldsymbol{p}^T\boldsymbol{x} \geq 1$. It follows that

$$t^* \leq R^*(\ell) \leq R(\ell) \leq R(\ell r\boldsymbol{p}^T\boldsymbol{x}) = \ell r R(\boldsymbol{p}^T\boldsymbol{x}) = \ell r \ .$$

∎

4. Bidirected generalized networks

In the previous sections we discussed generalized networks where the flow multipliers are positive numbers. We refer to the edges in these networks as head-tail edges. A head-tail edge produces a nonnegative amount of flow at the tail end of the edge and a proportional nonpositive amount at the head end. In bidirected generalized networks we allow two additional types of edges: head-head and tail-tail. The properties of edges of these types are shown in Figure 1. Note that a tail-tail edge can be viewed as a head-tail edge with a negative multiplier. Bidirected generalized networks are more general than bidirected networks (see [12]). In bidirected networks the multipliers associated with the edges are always unity. Bidirected networks were first considered by Edmonds [7] who related them to non-bipartite matching theory. In this section we apply the methods discussed in previous sections to flow problems on bidirected generalized networks.

4.1. UGT on bidirected networks

We discuss the application of the scheme of Adler and Cosares for solving the UGT problem on bidirected networks. A bidirected UGT problem has the form:

$$\begin{aligned} \text{Minimize } & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to } & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \ , \end{aligned}$$

where $\boldsymbol{b} \in R^n$, $\boldsymbol{c} \in R^m$, and the matrix $\boldsymbol{A} \in R^{n\times m}$ has at most two non-zero entries in each column. Note that head-tail edges correspond to columns where the two entries have

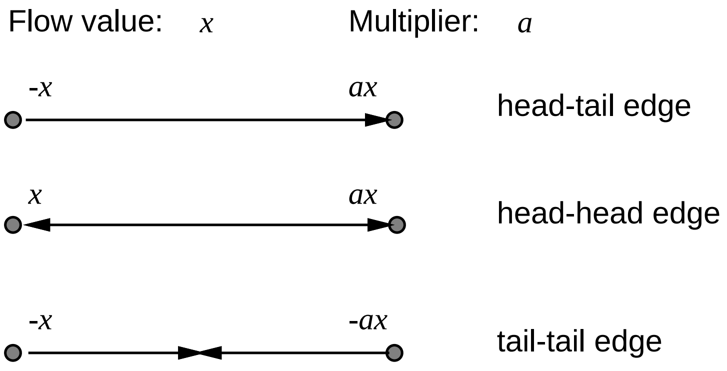

Figure 1: Edge types of a bidirected generalized network

opposite signs, head-head edges correspond to columns with two positive entries, and tail-tail edges correspond to columns with two negative entries. The dual of this problem is:

$$\begin{aligned} \text{Maximize} \quad & \boldsymbol{b}^T \boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{A}^T \boldsymbol{y} \leq \boldsymbol{c} \ , \end{aligned}$$

where the constraints have the TVPI property. Recall that when only head-tail edges are present, the dual has monotone constraints. When both head-tail and tail-tail edges are allowed, $\boldsymbol{A}^T$ is called a *pre-Leontief* matrix (see [6]), and it is known that in this case there exists a vector which maximizes all variables simultaneously. The scheme of Adler and Cosares applies when $\boldsymbol{b} \geq \mathbf{0}$. Similarly, when both head-tail and head-head edges are present, $-\boldsymbol{A}^T$ is a pre-Leontief matrix, and hence, there exists a vector which minimizes all variables simultaneously. The scheme of Adler and Cosares applies here when $\boldsymbol{b} \leq \mathbf{0}$. When all three types of edges are present, the dual constitutes a general TVPI system. The algorithm of [5] can be used to find a vector which maximizes or minimizes a single variable. The scheme of Adler and Cosares applies here to cases where $\boldsymbol{b} = \pm \boldsymbol{e}^i$. The UGT instances for which the scheme of Adler and Cosares applies are listed in Table 3. These instances can be solved by a single application of the algorithms of [5].

4.2. Generalized circulation on bidirected networks

We consider applying the approximation algorithm of Section 3 to generalized circulation problem where the underlying network is bidirected. Recall that the approximation algorithm iteratively computes a feasible flow and in the following iteration considers the residual graph. Since tail-tail edges give rise to head-head edges in the residual graph (and vice versa), we only consider networks where all three edge types are present. Note that when all three edge types are present the Adler-Cosares scheme applies to UGT instances where there is a single source or a single sink (that is, $\boldsymbol{b} = \boldsymbol{e}_i$ or $\boldsymbol{b} = -\boldsymbol{e}_i$ for some i $(1 \leq i \leq n)$.

Allowed edge types	*structure of the dual* $\boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{c}$	– *supply/demand vectors* – *TVPI needed*
head-tail	monotone TVPI	$\boldsymbol{b} \leq \boldsymbol{0}$ or $\boldsymbol{b} \geq \boldsymbol{0}$ monotone (maximize $\pm\boldsymbol{e}^T\boldsymbol{y}$)
head-tail, tail-tail	$\boldsymbol{A}^T$ is pre-Leontief	$\boldsymbol{b} \geq \boldsymbol{0}$ monotone (maximize $\boldsymbol{e}^T\boldsymbol{y}$)
head-tail, head-head	$-\boldsymbol{A}^T$ is pre-Leontief	$\boldsymbol{b} \leq \boldsymbol{0}$ monotone (minimize $\boldsymbol{e}^T\boldsymbol{y}$)
all types	general TVPI	$b_j = \pm 1$, $b_i = 0$ $(i \neq j)$ general (maximize/minimize y_j)

Table 3: Solving UGT on bidirected generalized networks

It follows that the approximation scheme presented in Section 3 can be used to solve bidirected generalized circulation instances where $\boldsymbol{b} = \pm\boldsymbol{e}_i$ for some i $(1 \leq i \leq n)$.

5. Concluding remarks

In this paper we presented algorithms for the uncapacitated generalized transshipment (UGT) problem and the generalized circulation (GC) problem. We also considered the UGT and GC problems on bidirected generalized networks. To solve UGT, we combined a scheme by Adler and Cosares [1], which reduces the restricted UGT problem where either only demand nodes or only supply nodes are present to solving the dual linear programming problem, with the algorithms given in [5]. The combination yielded better time bounds for restricted UGT instances.

In order to utilize the UGT algorithms for solving the capacitated GC problem, we introduced an iterative approximation algorithm. In each iteration, we consider a UGT instance, with costs which "capture" the relaxed capacities. The solution of this UGT instance yields an approximate solution for the GC instance. The next iteration considers the residual graph.

We now comment on the parallel running times of the algorithms mentioned above. We denote $f(n) = \tilde{O}(g(n))$ if there is a k such that $f(n) = O(g(n)(\log n)^k)$. The parallel complexity of the algorithms of [5] is $\tilde{O}(n)$ where the deterministic algorithm uses $O(mn)$ processors and the randomized one uses $O(m + n^2)$ processors. The algorithms for the restricted UGT instances have the same complexity. The approximation algorithms for GC run in $\tilde{O}(mn \log \epsilon^{-1})$ time with processor bounds of $O(mn)$ for the deterministic algorithm and $O(m + n^2)$ for the randomized one.

The existence of a strongly polynomial algorithm for the unrestricted UGT problem (where many sources and sinks are allowed) remains an open question. The LP dual of the UGT problem has the form of optimizing a general objective function subject to a monotone system (the feasibility version of the latter problem does have a strongly polynomial algorithm but the question is obviously open for the optimization version). The following demonstrates the difficulty of the problem by showing it is as hard as finding a strongly polynomial algorithm for the capacitated generalized transshipment problem (similar to [15]). An instance of capacitated generalized transshipment can be reduced in linear time to an instance of UGT with $3m$ edges and $n + 2m$ nodes. The reduction is as follows. Consider an instance of generalized circulation on a network $G = (V, E)$ where $d_v \in R$ is the supply or demand at v ($v \in V$), and a_e, u_e, and c_e are the multiplier, capacity, and cost, respectively, associated with the edge e ($e \in E$). The corresponding instance of the UGT problem $G' = (V \cup W \cup W', E')$ preserves the supplies and demands at the nodes in V. Each edge $e \in E$ has two corresponding nodes $w_e \in W$ and $w'_e \in W'$, and three edges in E' which form an undirected path. The node w_e has a demand u_e and the node w'_e has a supply $-u_e$. Suppose $e = (v_1, v_2)$ is an edge in G. The corresponding edges in G' are (i) (v_1, w) with multiplier 1 and cost c_e, (ii) (w', w) with multiplier 1 and cost 0, and (iii) (w', v_2) with multiplier a_e and cost 0.

References.

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

[2] E. Cohen. *Combinatorial Algorithms for Optimization Problems*. PhD thesis, Department of Computer Science, Stanford University, Stanford, Ca., 1991.

[3] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 145–155. ACM, 1991.

[4] E. Cohen and N. Megiddo. New algorithms for generalized network flows. In D. Dolev, Z. Galil, and M. Rodeh, editors, *Proc. of the 1st Israeli Symposium on the Theory of Computing and Systems*, pages 103–114. Springer-Verlag, Lecture Notes in Computer Science Vol. 601, 1992.

[5] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.*, 1993. To appear.

[6] R. W. Cottle and A. F. Veinott Jr. Polyhedral sets having a least element. *Math. Prog.*, 3:238–249, 1972.

[7] J. Edmonds. An introduction to matchings. In *Engineering Summer Conference*. The Univ. of Michigan, Ann Arbor, 1967. Mimeographed notes.

[8] F. Glover, J. Hultz, D. Klingman, and J. Stunz. Generalized networks: a fundamental computer-based planning tool. *Management Science*, 24(12), August 1978.

[9] A. V. Goldberg, S. K. Plotkin, and É. Tardos. Combinatorial algorithms for the generalized circulation problem. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 432–443. IEEE, 1988.

[10] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons, New York, 1984.

[11] K. Kapoor and P. M. Vaidya. Speeding up Karmarkar's algorithm for multicommodity flows. *Math. Prog.*, 1992. To appear.

[12] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Reinhart, and Winston, New York, 1976.

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

[14] S. Murray. An interior point conjugate gradient approach to the generalized flow problem with costs and the multicommodity flow problem dual. Manuscript, 1991.

[15] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 377–387. ACM, 1988.

[16] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Oper. Res.*, 34:250–256, 1986.

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