Stable Network Flow with Piece-wise Linear Constraints

Young-San Lin * Thành Nguyen †

Abstract

We consider a general stable flow problem in a directed and capacitated network, where each vertex has a strict preference list over the incoming and outgoing edges. A flow is stable if no group of vertices forming a path can mutually benefit by rerouting the flow. Motivated by applications in supply chain networks, we generalize the traditional Kirchhoff's law, requiring the outflow is equal to the inflow at every non-terminal node, to a monotone piecewise linear relationship between the inflows and the outflows. We show the existence of a stable flow using Scarf's Lemma, and provide a polynomial time algorithm to find such a stable flow. We further show that finding a minimum cost generalized stable network is NP-hard, while the problem is polynomial time solvable for the traditional stable flow satisfying Kirchhoff's law.

1 Introduction

In the classic stable marriage problem, men and women with individual preference orders of the opposite gender, are to be matched such that there are no man-woman pairs who are inclined to abandon their original partners and marry each other. Gale and Shapley [1962] showed the existence of such a stable matching by the deferred acceptance (DA) algorithm. Since then, the stable marriage problem and the DA algorithm have become the cornerstones of market design and have changed the organization of many centralized markets, including resident matching, school choice and kidney exchange. (See for example, Roth and Peranson [1999], Roth et al. [2005], Abdulkadiroglŭ and Sönmez [2003]).

Following the success of the stable marriage problem, stability in supply chain networks, a generalization of the stable marriage problem from a two-sided market to a network market, has been studied extensively. See for example, Ostrovsky [2008a], Fleiner [2010],

*Computer Science Department, Purdue University, e-mail: lin532@purdue.com
†Krannert School of Management, Purdue University, e-mail: nguye161@purdue.edu

Hatfield et al. [2015]. One natural version of the problem, first considered by Fleiner [2010], is modeled as a directed graph whose vertices and edges represent agents and bilateral contracts, respectively. The outgoing endpoint vertex of an edge is the seller of that contract while the incoming endpoint vertex is the buyer. Each edge of the network has a capacity, representing the maximum amount of goods that can be traded in the contract. It is further assumed that each vertex holds a preference list over the adjacent contracts. A flow is a collection of bilateral contracts satisfying Kirchhoff's law, that is, the sum of inflow contracts is the same as the sum of outflow contracts for every vertex, except a source and a sink vertex representing a producer and a consumer, respectively. A flow is stable if no group of agents forming a path can benefit more by cooperating among themselves.

Stable network flow has a variety of applications in physical and Internet traffic networks Haxell and Wilfong [2008] as well as supply chain and manufacturing networks Fleiner [2010]. However, because of the Kirchhoff's law, many applications do not fit this model. For example, the amount of goods that an intermediate firm receives if not equal to the amount the firm sells because during transportation the goods might get damaged and lost. In the case of manufacturing firms, the relationship between input and output varies depending on the products and the manufacturing technologies.[1]

Motivated by this, we extend Fleiner [2010] by considering monotone and piecewise linear relationship between inflows and outflows at every node of the network. While the generalized flow problems have been studied extensively from the "cardinal" optimization point of view, to the best of our knowledge, there is essentially no prior work studying the problems with ordinal preferences.

Our results show a sharp contrast between the generalized problem we consider and the the traditional stable flow. In particular, as shown in Fleiner [2010], the traditional stable flow problem can be reduced to the stable allocation problem, it is not true in our setting. We instead reduce this problem to a stable solution of Scarf's Lemma (Scarf [1967]). However, finding a stable solution in general framework of Scarf is PPAD-hard (Kintali [2008]). Our next contribution is to provide a polynomial time algorithm to find such a stable generalized flow. We obtain our main result by several reductions and new algorithms, generalizing the path augmenting method. We further study the optimization version of the problem and show that finding a minimum cost generalized stable network is NP-hard, while the problem is polynomial time solvable for the traditional stable flow satisfying Kirchhoff's law.

Our contributions are thus twofold. First, the generalized problem we introduce and

[1]See Ahuja et al. [1993] for other applications of general flow constraints.

our algorithm apply to a wide range of applications in supply chains networks. Second, our hardness result shows that generalizing Kirchhoff's law is not just a technical transformation between network flows, it significantly changes the problem's computational complexity.

The paper is organized as follow. After discussing related work, in section 2, we describe the setting and background definitions of this problem including how flow and stability are defined when agents utilize some special mappings to their inflow and outflow. Specifically, we introduce the concept of piecewise linear (PL) mapping, truncated linear (TL) mapping, and linear mapping.

In section 3, we show the existence of stable flow in a truncated linear network (TL-network) where inflow and outflow has a TL relation for all agents by a reduction to Scarf's Lemma (see Scarf [1967]). Later on, by reducing a piecewise linear network (PL-network) where inflow and outflow has a PL relation for all agents to another TL-network, we can show the existence of stable flow in PL-networks.

In section 4, we present a polynomial time path augmenting algorithm that finds a stable flow in an acyclic network where all agents apply linear functions, i.e. an acyclic linear network (AL-network). The main difference of our approach from Cseh and Matuschke [2013] is an augmented path may be a σ-cycle, a path from a source to a vertex that is visited twice. Each iteration in Cseh and Matuschke [2013] augments a path from source to sink or a cycle since when augmenting a cycle, the augmented flow from the source to cycle is always zero. This does not apply to our setting because flow conservation property is not guaranteed.

In section 5, we show a reduction from finding a stable flow in any TL-network to finding a stable flow in its equivalent AL-network. The main approach is to first map the TL-network instance to a Scarf's instance mentioned in section 3, then apply the path augmenting algorithm presented in section 4 on the equivalent AL-network to find the solution of the Scarf's instance. As a PL-network instance can be reduced to a TL-network instance, the entire problem for finding stable flow in PL-networks is enclosed.

In section 6, we consider an optimization variant of the stable flow problem where edges have associated cost. For the traditional problem, finding stable flow with minimum cost is polynomial time solvable. However, finding the minimum cost a stable flow in AL-networks, TL-networks, and PL-networks are all NP-hard.

Related Work

As discussed above, our paper is closely related to Fleiner [2010], which provides a reduction of the stable flow to the stable allocation problem considered in Baïou and Balinski

[2002]. The reduction, therefore, shows the existence of a stable flow. Cseh et al. [2013] provide a preflow-push variant of the Gale-Shapley algorithm to find a stable flow in pseudo-polynomial time. Shortly afterward, Cseh and Matuschke [2013] used a path augmenting variant of the Gale-Shapley algorithm to improve the running time to be polynomial. To the best of our knowledge, there have not been existence results nor polynomial time algorithms to find a stable solution for generalized stable flow problem considered in this paper.

There is an extensive optimization literature on generalized flow problem, that assume the total outflow is equal to a constant times the total inflow. (See for example, Kantorovich [1960], Ahuja et al. [1993] and recent work of Olver and Végh [2016]). This literature however mainly focused on the "cardinal" problem, that is, to find the maximum total flow. Our paper provides a first step to investigate "ordinal" preferences in the generalized flow problem.

Our paper is also related to a growing economic literature on network stability problem. Ostrovsky [2008b] is the first to introduce the concept of stability to supply chain networks, and followed by, among others, Hatfield et al. [2013, 2015] and Fleiner et al. [2016]. However, these papers focus on discrete choice function and integral stable solutions. In particular, in these models each edge in the network corresponds to a trade and an outcome is a set of trades. In our paper each trade has a continuous capacity, and the difficulties are the non convex constraints between the inflow and outflow for each vertex.

Recently, Che et al. [2015], Azevedo and Hatfield [2015] consider continuous model of matching in *two-sided markets*, and show the existence of a stable solution. However, these results are based on fixed-point theorem argument and polynomial time algorithms to find such a stable solution are thus not known.

2 Preliminaries

2.1 Stable Flow

We start with traditional stable flow introduced in Fleiner [2010]. A network is a quadruple (G, s, t, c), where $G = (V \cup \{s, t\}, E)$ is a simple digraph. s and t are the source and sink vertices and $c : E \to \mathbb{R}_+$ determines the capacity $c(e)$ for $e \in E$. s only has outgoing edges while t only has incoming edges. For each $v \in V$, there is a path from s to v and a path from v to t. The strict preference $\succ_v$ of a vertex $v \in V$ is defined over edges adjacent to v. Incoming edges and outgoing edges are ranked strictly and separately. $e_1 \succ_v e_2$ means v prefers e_1 to e_2.

A *flow* of network (G, s, t, c) is a function $f : E \to \mathbb{R}_{\geqslant 0}$ such that capcity condition $0 \leqslant f(e) \leqslant c(e)$ holds for each $e \in E$ and each vertex $v \in V$ satisfies Kirchhoff's Law: $\sum_{uv \in E} f(uv) = \sum_{vu \in E} f(vu)$, that is, the amount of inflow equals the amount of outflow.

Let f be a flow, a walk $W = (v_1, v_2, ..., v_k)$ is *blocking* f if all the followings hold:

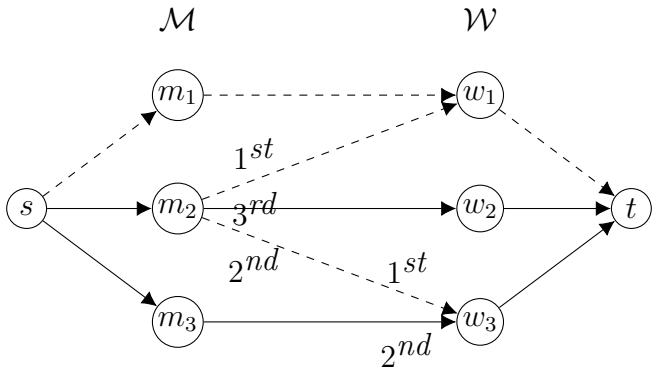
1. $v_i v_{i+1}$ is unsaturated for $i = 1, ..., k-1$.
2. $v_1 = s$ or there is an edge $v_1 u \in E$ such that $v_1 v_2 \succ_{v_1} v_1 u$ and $f(v_1 u) > 0$.
3. $v_k = t$ or there is an edge $u v_1 \in E$ such that $v_{k-1} v_k \succ_{v_k} u v_k$ and $f(u v_k) > 0$.

Network flow stability is a general model capturing the classic stable marriage problem as a sub case. To see this, consider the classic stable marriage problem with a set $\mathcal{M}$ of n men and a set $\mathcal{W}$ of n women with individual preference of the opposite gender. We also assume that each man or woman rather have a partner than be unmatched. In a matching M, we denote $M(m)$ as the partner of m for $m \in \mathcal{M}$ and $M(w)$ as the partner of w for $w \in \mathcal{W}$. A pair (m, w) is blocking M if all the followings hold:

1. m and w are not matched in M.
2. m is unmatched in M or m prefers w to $M(m)$.
3. w is unmatched in M or w prefers m to $M(w)$.

We can construct a network by attaching s to men vertices, men vertices to women vertices, and women vertices to t, by unit capacity edges. For preferences, each $m \in \mathcal{M}$ prefers w_1 to w_2 if and only if $mw_1 \succ_m mw_2$ and each $w \in \mathcal{W}$ prefers m_1 to m_2 if and only if $m_1 w \succ_w m_2 w$. Herein a blocking pair corresponds to a blocking walk in the network. Example 1 shows how network flow stability can be applied to the stable marriage problem.

Example 1



This network corresponds to a stable marriage matching instance. $m_2w_1 \succ_{m_2} m_2w_3 \succ_{m_2} m_2w_2$ *and* $m_2w_3 \succ_{w_3} m_3w_3$*. Although* $\mathcal{M}$ *and* $\mathcal{W}$ *form a complete bitartite graph, we highlight the edges with flow value 1 by solid edges while other edges have flow value 0. The dashed edges form blocking walks. The flow corresponds to the matching* $\{(m_2, w_2), (m_3, w_3)\}$*. The blocking pair* (m_1, w_1) *corresponds to the blocking walk* (s, m_1, w_1, t)*; the blocking pair* (m_2, w_1) *corresponds to the blocking walk* (m_2, w_1, t)*; while the blocking pair* (m_2, w_3) *corresponds to the blocking walk* (m_2, w_3)*.*

Network flow stability not only can be applied to stability in two sided markets, but also to stability in multi-level markets or supply chain networks. It is convenient to depict these settings as a digraph, where each capacitated edge $e \in E$ represents a contract with limited product quantity, and each vertex $v \in V$ represents an intermediate agent that holds its individual preference of incoming and outgoing contracts and strives to maximize the amount of flow through v. A blocking walk of a flow interprets a scenario in which a group of agents are willing to cooperate selfishly and benifit from rerouting some flow. A flow is stable if every agent is satisfied to its current offer such that no group of agents have the incentive to reroute the flow since they cannot benifit more.

2.2 Generalized Stable Flow

To define a general network flow, we use the same notation as the traditional stable flow problem, except we consider some special classes of mappings for vertices that we use to model the constraint between inflow and outflow. In our paper, a mapping is more general than a function because it maps an element of $\mathbb{R}_{\geqslant 0}$ to an element or an interval of $\mathbb{R}_{\geqslant 0}$. We are interested in linear, truncated linear and piecewise linear mappings. They are defined as follows.

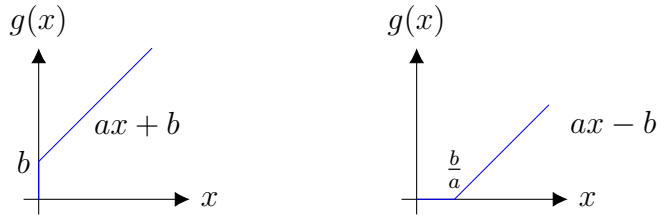
DEFINITION 2.1 *g is a* **linear mapping** *if $g(x) = ax$ for $a > 0$. g is a* **truncated linear mapping** *if it is one of the following:*

$$g(x) = \begin{cases} [0, b] & \text{if } x = 0, \\ ax + b & \text{otherwise.} \end{cases} \tag{1}$$

$$g(x) = \begin{cases} 0 & \text{if } x < \frac{b}{a}, \\ ax - b & \text{otherwise.} \end{cases} \tag{2}$$

where $a > 0$ and $b \geqslant 0$.

The following figure illustrates the two possible cases of a truncated linear mapping.

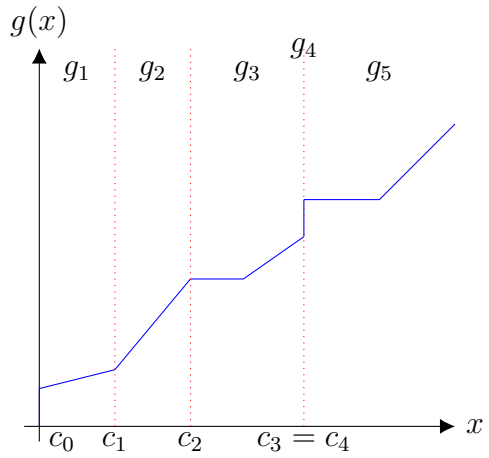


DEFINITION 2.2 *A* **piecewise linear mapping** *concatenates k segments of truncated linear mappings, namely:*

$$g(x) = \begin{cases} g_1(x) & \text{if } x \in [c_0, c_1] \\ \max\{g_1(c_1)\} + g_2(x - c_1) & \text{if } x \in [c_1, c_2], \\ \max\{g_1(c_1)\} + \max\{g_2(c_2 - c_1)\} + g_3(x - c_2) & \text{if } x \in [c_2, c_3], \\ \vdots & \\ \sum_{i=1}^{k-1} \max\{g_i(c_i - c_{i-1})\} + g_k(x - c_{k-1}) & \text{if } x \in [c_{k-1}, \infty). \end{cases} \tag{3}$$

where $c_0 = 0$, $c_0 \leqslant c_1 \leqslant ... \leqslant c_{k-1}$, and g_i is a truncated linear mapping for $i = 1, 2, ..., k$.

The following figure is an example of piecewise linear mapping.



Note that $g_2(x)$ is a linear mapping while $g_4(x)$ maps to only an interval of $\mathbb{R}_{\geqslant 0}$.

In a network (G, s, t, c), for each $v \in V$, v is associated with a mapping g_v where x is the inflow of v and $g_v(x)$ is the outflow of v. It is convenient to introduce the following definition assuming that each vertex in a network all apply some specific mappings.

DEFINITION 2.3 *If for all $v \in V$, g_v is a truncated linear mapping, then the network is a* **truncated linear network** *(TL-network). If for all $v \in V$, g_v is a piecewise linear mapping, then the network is a* **piecewise linear network** *(PL-network).*

In a market network, each vertex can be regarded as an agent given offers of incoming and outgoing contracts. They evaluate the quantity of desired outgoing contracts to be signed based on how many incoming contracts are accepted. Therefore, the feasibility of contract assignment can be defined as the following:

DEFINITION 2.4 *Given a flow f of a TL-network or PL-network, for each $v \in V$, let $f_{in}(v) = \sum_{uv \in E} f(uv)$ and $f_{out}(v) = \sum_{vu \in E} f(vu)$, f is* **feasible** *if:*

1. $0 \leqslant f(e) \leqslant c(e)$ *for each* $e \in E$.
2. *For each $v \in V$, $g_v(f_{in}(v)) = f_{out}(v)$ if $g_v(f_{in}(v))$ maps $f_{in}(v)$ to an element of $\mathbb{R}_{\geqslant 0}$; $g_v(f_{in}(v)) \in f_{out}(v)$ if $g_v(f_{in}(v))$ maps $f_{in}(v)$ to an interval of $\mathbb{R}_{\geqslant 0}$.*

The definition of flow stability is defined in the similar way to the traditional definition:

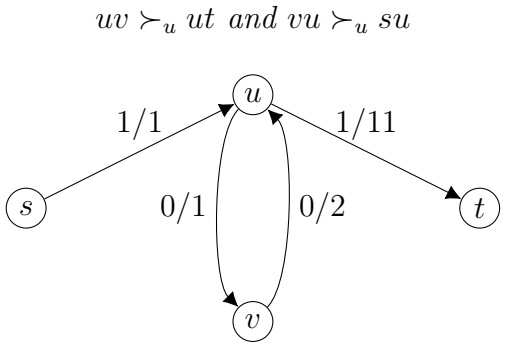
DEFINITION 2.5 *Given a flow f of a TL-network or PL-network, f is stable if it is feasible and there is no blocking walk in the network. f has a blocking walk $W = (v_1, v_2, ..., v_{k-1}, v_k)$ where $v_i v_{i+1} \in E$ for $i = 1, ..., k-1$ if all the followings hold:*

1. *There exists a vector $V_W = (r_1, r_2, ..., r_{k-1})$ such that:*
   (a) $r_i \geqslant 0$ *and there is at least one* $r_i > 0$ *for* $i = 1, ..., k-1$.
   (b) $r_i \leqslant c(v_i v_{i+1}) - f(v_i v_{i+1})$ *for* $i = 1, ..., k-1$.
   (c) *For* $i = 2, ..., k-1$, $f_{out}(v_i) + r_i = g_{v_i}(f_{in}(v_i) + r_{i-1})$ *if* g_{v_i} *maps* $f_{in}(v_i) + r_{i-1}$ *to an element of* $\mathbb{R}_{\geqslant 0}$; $f_{out}(v_i) + r_i \in g_{v_i}(f_{in}(v_i) + r_{i-1})$ *if* $g(f_{in}(v))$ *maps to an interval of* $\mathbb{R}_{\geqslant 0}$.
2. $v_1 = s$ *or there is an edge* $v_1 u \in E$ *such that* $v_1 v_2 \succ_{v_1} v_1 u$ *and* $f(v_1 u) > 0$.
3. $v_k = t$ *or there is an edge* $u v_1 \in E$ *such that* $v_{k-1} v_k \succ_{v_k} u v_k$ *and* $f(u v_k) > 0$.

All the conditions in Definition 2.5 are the same as the ones for the traditional stable flow, except the first condition has to be modified because the flow no longer satisfies Kirchhoff's Law. Point 1.(a) shows that in the blocking walk W, at least one agent v_i has the incentive to deliver positive flow. Point 1.(b) restricts the flow value that is intended

to be delivered along W by the remaining capacity, while point 1.(c) checks the feasibility. In other words, W is blocking f if vertices in W are better off by rerouting a feasible flow bounded by the remaining capacity. f is stable if no group of vertices can benifit from rerouting the flow.

EXAMPLE 2 *Consider the following TL-network. On each edge, the upper number represents the flow value and the lower number correspond to the capacity.*

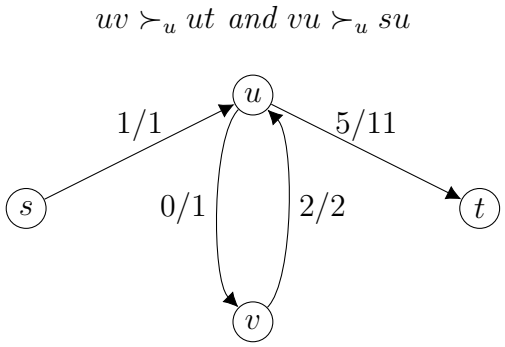


$$g_u(x) = \begin{cases} 0 & \text{if } x < 0.5, \\ 2x - 1 & \text{otherwise.} \end{cases}$$

$$g_v(x) = \begin{cases} [0, 2] & \text{if } x = 0, \\ 3x + 2 & \text{otherwise.} \end{cases}$$

u prefers any incoming or outgoing edges that have v as an endpoint. The flow assignment given in the picture is feasible but not stable. There are two blocking walks: $W_1 = (u, v, u)$ and $W_2 = (u, v, u, t)$. There exists a vector $V_{W_1} = (0, 2)$ that can be rerouted along W_1. There exists a vector $V_{W_2} = (0, 2, 4)$ that can be rerouted along W_2.

The following flow assignment is stable:



$$g_u(x) = \begin{cases} 0 & \text{if } x < 0.5, \\ 2x - 1 & \text{otherwise.} \end{cases}$$

$$g_v(x) = \begin{cases} [0, 2] & \text{if } x = 0, \\ 3x + 2 & \text{otherwise.} \end{cases}$$

We denote *TL-stable-flow* (TL-SF) a stable flow assignment in a TL-network, or the problem of finding a stable flow in a TL-network, and *PL-stable-flow* (PL-SF) a stable flow assignment in a PL-network, or the problem of finding stable flow in PL-networks.

3 Flow Stability of TL-networks and PL-networks

Scarf's Lemma originally appeared as a tool to prove the non-emptiness of the core in an n person game (Scarf [1967]). In this section, we show the existence of TL-SF by a reduction to Scarf's Lemma. The existence of PL-SF, on the other hand, is shown by a reduction to the existence of TL-SF.

3.1 Scarf's Lemma

DEFINITION 3.1 *Let A be an $m \times n$ nonnegative matrix with at least one positive entry in every column and row, $b \in \mathbb{R}^m_+$ be a positive vector, and $\mathcal{P} = \{x : x \geqslant 0, Ax \leqslant b\}$. For each row i of A, there is a strict ranking $\succ_i$ over the columns in $\{1 \leqslant j \leqslant n : A_{ij} > 0\}$. $k \succ_i j$ means row i prefers column k to column j.*

We say $x \in \mathcal{P}$ **dominates** *column j if there exists a row i such that:*

1. *$A_{ij} > 0$ and the constraint i binds, i.e. $(Ax)_i = b_i$.*
2. *$k \succ_i j$ for any other $k \neq j$ such that $A_{ik} > 0$ and $x_k > 0$.*

To simplify our notation, given an $x \in \mathcal{P}$, we also say row i *dominates* column j if the above mentioned conditions hold.

LEMMA 3.1 (SCARF'S LEMMA) *For any above mentioned A, b, and $\succ_i$, there exists an $x^* \in \mathcal{P}$ that dominates all columns of A.*

3.2 The Reductions

3.2.1 From TL-SF to Scarf's Lemma

Given a TL-network (G, s, t, c), where $G = (V \cup \{s, t\}, E)$, and $\succ_v$ for any $v \in V$, in order to show the existence of TL-SF, we employ Scarf's Lemma, i.e. construct the corresponding matrix A and vector b. We start with the edge capacity constraints:

1. For each $e \in E$, create column x_e for A and row e for A and b.
2. Set $A_{ex_e} = 1$ and $b_e = c(e)$.

Herein, x_e is a variable of vector x that represents the flow value of edge e. The aforementioned construction assures that the flow of edge e will not exceed its capacity $c(e)$. The column of A that corresponds to variable x_e is also labelled x_e. Similar notation applies in the later construction.

To ensure flow feasibility and stability, we create two rows for each vertex $v \in V$, one for the inflow and the other for the outflow. Besides, v is associated with a truncated linear mapping g_v, we add v as a subscript of the parameters in Definition 2.1. That is, g_v has parameters a_v and b_v involved. We start with the construction of A:

1. For each $v \in V$, create row v^{in} and v^{out}.
2. For each $v \in V$, create column x_v^{in} and column x_v^{out}.
3. For $e = uv \in E$, set $A_{v^{in}x_e} = a_v$.
4. For $e = vu \in E$, set $A_{v^{out}x_e} = 1$.
5. For each $v \in V$, $A_{v^{in}x_v^{in}} = A_{v^{in}x_v^{out}} = A_{v^{out}x_v^{in}} = A_{v^{out}x_v^{out}} = 1$.
6. Row v^{in} prefers column x_v^{in} the most and column x_v^{out} the least. Preference of $e = uv \in E$ remains the same as $\succ_v$.
7. Row v^{out} prefers column x_v^{out} the most and column x_v^{in} the least. Preference of $e = vu \in E$ remains the same as $\succ_v$.

By the setting of the preferences of row v^{in} and v^{out}, variables x_v^{in} and x_v^{out} enable us to capture the stability of a flow. The remaining is to select proper values for $b_{x_v^{in}}$ and $b_{x_v^{out}}$. To achieve this, let $M(v) = \max(\sum_{uv \in E} c(uv), \sum_{vu \in E} c(vu), b_v) + 1$ be a large value such that at least one of x_v^{in} and x_v^{out} has to be positive. The remaining construction of b is:

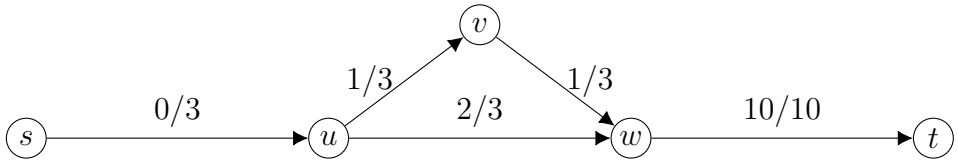
1. If g_v is in the form of equation 1, set $b_{v^{in}} = M(v)$ and $b_{v^{out}} = M(v) + b_v$.
2. If g_v is in the form of equation 2, set $b_{v^{in}} = M(v) + b_v$ and $b_{v^{out}} = M(v)$.

For vertex v, the difference between $b_{v^{out}}$ and $b_{v^{in}}$ is set to b_v. Depending on whether the inflow or outflow part is truncated, $b_{v^{out}}$ and $b_{v^{in}}$ are set to proper values accordingly.

Let x^* be the Scarf's solution that dominates all the columns of A. With a bit abuse of notation, we label columns of A by superscript or subscript of x. The superscript or subscript of x^* stands for the exact value of the Scarf's solution x^*. For each $e \in E$, we can set $f(e) = {x^*}_e$. x^* corresponds to a TL-SF by the setting of A and b. On the other hand, a TL-SF corresponds to a vector x^* that dominates all the columns of A. For more details, see Theorem 3.1.

Before showing the proof, to give an illustration of the construction, it may be useful to look over Example 3.

EXAMPLE 3 *Consider the following TL-network:*



$uv \succ_u uw$ $\qquad\qquad$ $uw \succ_w vw$

$$g_u(x) = \begin{cases} [0,4] & \text{if } x = 0, \\ x+4 & \text{otherwise.} \end{cases}, \; g_v(x) = \begin{cases} 0 & \text{if } x < 0.5, \\ 2x-1 & \text{otherwise.} \end{cases}, \; g_w(x) = \begin{cases} [0,1] & \text{if } x = 0, \\ 3x+1 & \text{otherwise.} \end{cases}$$

The corresponding Scarf's solution is the following where the blanks are zeros:

$$Ax^* = \begin{array}{c} \\ su \\ uv \\ vw \\ uw \\ wt \\ u^{in} \\ u^{out} \\ v^{in} \\ v^{out} \\ w^{in} \\ w^{out} \end{array} \begin{array}{c} \begin{array}{cccccccccccc} x_{su} & x_{uv} & x_{vw} & x_{uw} & x_{wt} & x_u^{in} & x_u^{out} & x_v^{in} & x_v^{out} & x_w^{in} & x_w^{out} \end{array} \\ \left(\begin{array}{cccccccccccc} 1 & & & & & & & & & & \\ & 1 & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & 1 & & & & & & & \\ & & & & 1 & & & & & & \\ \hline 1 & & & & & 1 & 1 & & & & \\ & 1 & & 1 & & 1 & 1 & & & & \\ \hline & 2 & & & & & & 1 & 1 & & \\ & & 1 & & & & & 1 & 1 & & \\ \hline & & 3 & 3 & & & & & & 1 & 1 \\ & & & & 1 & & & & & 1 & 1 \end{array}\right) \end{array} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 2 \\ 10 \\ 7 \\ 0 \\ 3 \\ 0 \\ 2 \\ 0 \end{pmatrix} \leqslant \begin{pmatrix} 3 \\ 3 \\ 3 \\ 3 \\ 10 \\ 7 \\ 11 \\ 5 \\ 4 \\ 11 \\ 12 \end{pmatrix} = \begin{pmatrix} b_{su} \\ b_{uv} \\ b_{vw} \\ b_{uw} \\ b_{wt} \\ b_{u^{in}} \\ b_{u^{out}} \\ b_{v^{in}} \\ b_{v^{out}} \\ b_{w^{in}} \\ b_{w^{out}} \end{pmatrix}$$

The following table is the rows ranking over columns from the highest to the lowest:

row	u^{in}	u^{out}	v^{in}
column	$x_u^{in}, x_{su}, x_u^{out}$	$x_u^{out}, x_{uv}, x_{uw}, x_u^{in}$	$x_v^{in}, x_{uv}, x_v^{out}$

row	v^{out}	w^{in}	w^{out}
column	$x_v^{out}, x_{vw}, x_v^{in}$	$x_w^{in}, x_{uw}, x_{vw}, x_w^{out}$	$x_w^{out}, x_{wt}, x_w^{in}$

By the aforementioned construction, $M(u) = 7$, $M(v) = 4$, *and* $M(w) = 11$. *Clearly, the flow capcity constraints are satisfied and only edge* wt *is saturated which corresponds to the only capacity binding constraint row* wt. *Flow feasibility is also satisfied. We can see*

that row u^{out} does not bind and $f_{out}(u) = 3 < 4$ does not reach the truncated threshold. For other vertices, the inflow and outflow constraints all bind.

For stability, since row su does not dominate column x_{su}, row u^{in} is the only choice to dominate column x_{su}. By the setting of $M(u)$, one of x^{*in}_u and x^{*out}_u must be positive, this forces $x^{*in}_u = 7$ and $x^{*out}_u = 0$. Since row u^{out} prefers column x^{in}_u the least, row v^{in} must dominate column x_{uv} and row w^{in} must dominate column x_{uw}. Therefore, $x^{*in}_v > 0$ and $x^{*out}_v = 0$, and $x^{*in}_w > 0$ and $x^{*out}_w = 0$. Column x_{vw} is less preferred than column x_{uw} and column x^{in}_w by row w^{in}. This ensures that x^* dominates all the columns of A.

THEOREM 3.1 *There exists a TL-SF in a TL-network.*

See Appendix A for the formal proof.

3.2.2 From PL-SF to TL-SF

Given a PL-network (G, s, t, c), where $G = (V \cup \{s, t\}, E)$, and $\succ_v$ for any $v \in V$, as the existence of TL-SF is shown in Theorem 3.1, we reduce PL-SF to TL-SF to show the existence of PL-SF.

For each $v \in V$, v is associated with a piecewise linear mapping g_v, we add v as a subscript of the parameters in Definition 2.2. That is, g_v has k_v segments of truncated linear mappings $g_{v,i}$ and $c_{v,i-1}$ as segment boundaries where $i = 1, ..., k_v$.

We create a sub-TL-network for each $v \in V$ as the following:

1. Split v into v_{in} and v_{out} where $g_{v_{in}}(x) = x$ and $g_{v_{out}}(x) = x$.

2. For each $uv \in E$ (for simplicity, suppose $s_{out} = s$ and $t_{in} = t$), connect u_{out} to v_{in} by keeping the same capacity and preference as uv, that is, capacity of $u_{out}v_{in}$ is the same as uv, and v_{in} has the same preference for incoming edges as v.

3. For each segment that applies $g_{v,i}$, create vertex v_i where $g_{v_i}(x) = g_{v,i}$ and edges $v_{in}v_i$ and v_iv_{out}.

4. For capacities of $v_{in}v_i$ and v_iv_{out}:

   (a) If $g_{v,i}$ follows equation 1, then set $c(v_{in}v_i) = c_{v,i} - c_{v,i-1}$ and $c(v_iv_{out}) = \max\{g_{v,i}(c_{v,i} - c_{v,i-1})\}$.

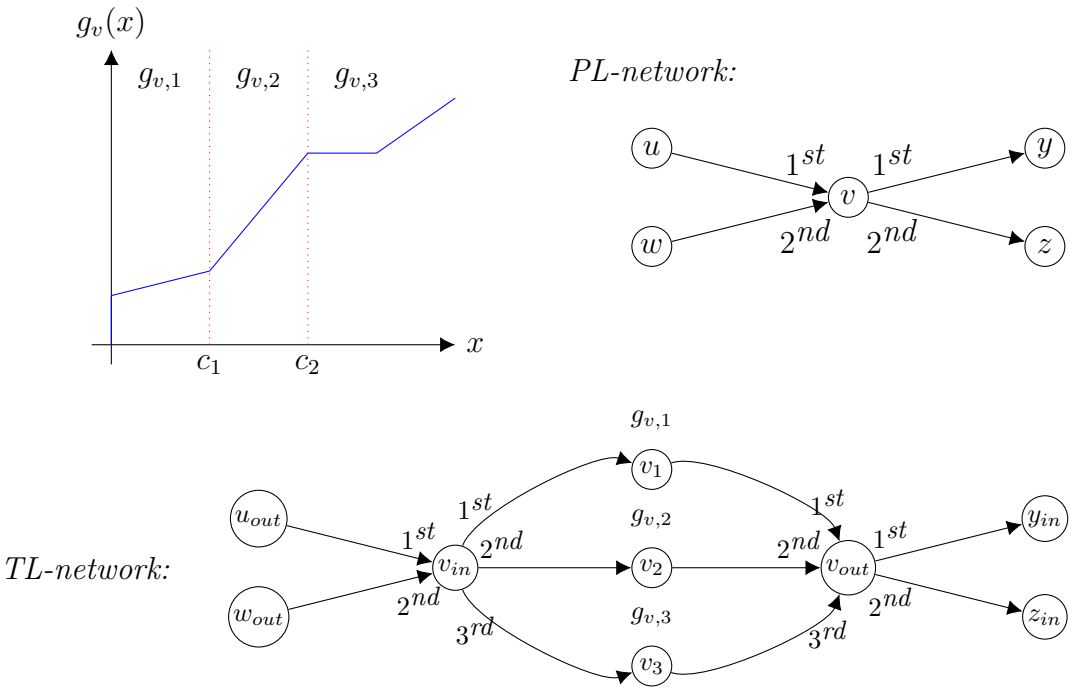
   (b) If $g_{v,i}$ follows equation 2, then set $c(v_{in}v_i) = c_{v,i} - c_{v,i-1}$ and $c(v_iv_{out}) = \max\{g_{v,i}(c_{v,i} - c_{v,i-1})\}$.

(c) By (a) or (b), if $c(v_{in}v_i) = 0$ then reset $c(v_{in}v_i) = \epsilon$; if $c(v_iv_{out}) = 0$ then reset $c(v_iv_{out}) = \epsilon$, for some small $\epsilon > 0$.

5. v_{in} prefers $v_{in}v_i$ and v_{out} prefers $v_{out}v_i$ with smaller i.

In the new TL-network, we split v into v_{in} and v_{out} to handle incoming and outgoing edges in the same behavior as v in the original PL-network, then create gadget vertices v_i to handle the truncated linear mapping segments and prioritize the segments with smaller i accordingly. Point 4.(c) deals with extreme cases to ensure that each edge has a positive capacity in the PL-network. To give an illustration of the construction, it may be useful to look over Example 4.

EXAMPLE 4



In this example, we focus on vertex $v \in V$. uv, wv, vy, and vz are all in E. We split each vertex and connect the out vertices to the in vertices, so $u_{out}v_{in}$, $w_{out}v_{in}$, $v_{out}y_{in}$, and $v_{out}z_{in}$ are in the new TL-network. In the original PL-network, $uv \succ_v wv$ and $vy \succ_v vz$, so $u_{out}v_{in} \succ_{v_{in}} w_{out}v_{in}$ and $v_{out}y_{in} \succ_{v_{out}} v_{out}z_{in}$ in the new TL-network. Besides, we set proper capacities and prioritize the preference for $v_{in}v_i$ and v_iv_{out} for $i = 1, 2, 3$ such that $f_{out}(v)$ meets the expected flow value $g_v(f_{in}(v))$.

By the construction in section 3.2.2 and Theorm 3.1, we obtain the following corollary:

COROLLARY 3.1 *There exists a PL-SF in a PL-network.*

4 A Path Augmenting Algorithm for AL-Networks

In section 3, we show the existence of TL-SF and PL-SF. In this section, we first introduce a path augmenting algorithm that finds a stable flow assignment for an *acyclic linear network* defined as the following:

DEFINITION 4.1 *In a network (G, s, t, c), if G does not have any cycle and for all $v \in V$, g_v is a linear mapping, then the network is an* **acyclic linear network** *(AL-network).*

We denote *AL-stable-flow* (AL-SF) a stable flow assignment in an AL-network, or the problem of finding a stable flow in an AL-network.

Starting from flow stability in AL-networks is useful because we eventually show a series of reductions from PL-SF and TL-SF to AL-SF in section 5. Thus, having a constructive algorithm for AL-SF enables us to constructively find TL-SF and PL-SF.

4.1 The Algorithm

Suppose we are given an AL-network (G, s, t, c), where $G = (V \cup \{s, t\}, E)$, and $\succ_v$ for any $v \in V$. In the algorithm, each vertex $v \in V \cup \{s\}$ is associated with a state and an edge. The states for vertices are $\{\texttt{PROPOSE}, \texttt{REJECT}, \texttt{DONE}\}$. If v is at the `PROPOSE` state, then the associated edge is the outgoing edge that v currently prefers the most. v reaches the `REJECT` state when v is running out of outgoing edges to propose to. The associated edge at this state is the incoming edge with positive flow value that v currently prefers the least. v reaches the `DONE` state when its most preferred edge is also rejected and there is no associated edge.

The DA algorithm works in a fashion of path augmentation. In each iteration, an augmenting path corresponds to a series of acceptance and rejections. In the augmenting path, vertices at the `PROPOSE` state proposes to their currently most preferred edge while vertices at the `REJECT` state rejects their currently least preferred edge. Vertices in the `DONE` state will not be involved in any future path augmentations.

Initially, for each $v \in V \cup \{s\}$, v.state = `PROPOSE`, v.edge is set to the most preferred outgoing edge of v. The preference of s can be arbitrary.

In each iteration, the auxiliary graph $H = (V_H, E_H)$ is constructed as the following:

1. $V_H = V$.
2. $E_H = \{v.\text{edge} : v \in V \cup \{s\} \text{ and } v.\text{state} = \texttt{PROPOSE}\} \cup \{\text{rev}(v.\text{edge}) : v \in V \cup \{s\} \text{ and } v.\text{state} = \texttt{REJECT}\}$ where $\text{rev}(uv) = vu$.

For each $uv \in E_H$, the residual capacity $c_f(uv)$ based on the current flow f is:

$$c_f(uv) = \begin{cases} c(uv) - f(uv) & \text{if } u.\text{state} = \texttt{PROPOSE} \text{ and } u.\text{edge} = uv \\ f(vu) & \text{if } u.\text{state} = \texttt{REJECT} \text{ and } u.\text{edge} = uv \end{cases}$$

Before showing how the algorithm works, we introduce a special path to augment, a *σ-cycle*, a path that meets a cycle.

DEFINITION 4.2 *A* **σ-cycle** *is a path $P = (s, v_1, v_2, ..., v_k)$ where all vertices are distinct except $v_k = s$ or $v_k = v_j$ for some $1 \leqslant j < k$.*

The algorithm iteratively augments the flow f in H. In each round, one can always augment an s-t-path or a σ-cycle P such that $c_f(e) = 0$ for some e in P. In the traditional stable flow problem, we augment either an s-t-path or a cycle. For AL-networks, we may have to augment a σ-cycle because flow conservation no longer holds.

After the augmentation, traverse backward from the augmenting path P and update the associated edge or the state of a vertex u where uv is a saturated edge in H after the augmentation. If u just finished proposing to v, then move on to the next preferred edge. If u is running out of vertices to propose to, then move to the `REJECT` state, set the associated edge to the least preferred edge wu that is currently accepted, and update the vertices less preferred than wu that are going to propose to u. If u is running out of vertices to reject, then u has rejected all the edges, set u.state to `DONE`. The algorithm stops when s.state = `REJECT`. Namely, there are no edges for s to propose to.

Algorithm 1 is a path augmenting DA algorithm for AL-networks. To see how it works, it may be useful to look up Example 5.

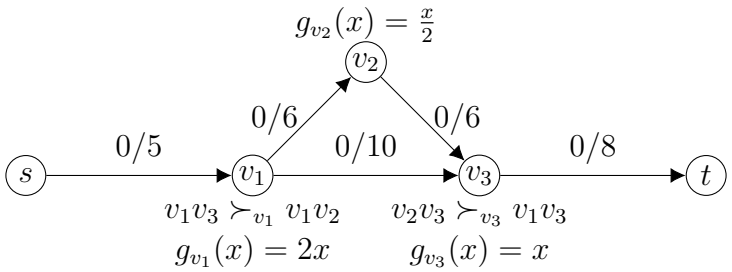
Algorithm 1 : Path Augmenting DA Algorithm for AL-networks

1: Initialize the state and associated edge for $v \in V \cup \{s\}$ and set f as zero flow.
2: **while** s.state = `PROPOSE` **do** ▷ H and f are global variables
3: Let $P = (v_0, v_1, ..., v_k)$ $(s = v_0)$ be an s-t-path or a σ-cycle in H
4: Augment f by P such that the capacity of at least one edge in H drops to 0
5: **for** $i = k$ to 1 **do**
6: **if** $c_f(v_{i-1}v_i) = 0$ and v_{i-1} was not updated in this iteration **then** `Update`(v_{i-1})
7: **if** H has a new augmentable s-t-path or σ-cycle **then break**
8: **return** f

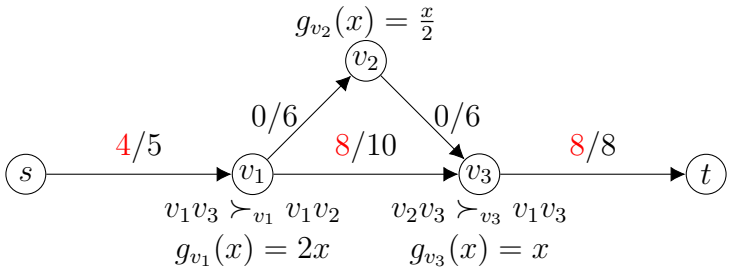
Procedure 1 : `Update`(u)

1: **if** u.state $=$ `PROPOSE` **then**
2: u.edge $\leftarrow$ u's next preferred edge uv with "$uv \succ_v v$.edge and v.state $=$ `REJECT`" or "v.state $=$ `PROPOSE`"
3: **if** no such v **then**
4: u.state $=$ `REJECT`
5: **if** $u = s$ **then**
6: **return**
7: **if** u.state $=$ `REJECT` **then**
8: u.edge $\leftarrow$ u's next least preferred edge vu with $f(vu) > 0$
9: **if** no such v **then**
10: u.state $=$ `DONE`
11: u.edge $\leftarrow \emptyset$
12: **for** each $w \in V \cup \{s\}$ where w.state $=$ `PROPOSE` and $wu = w$.edge **do**
13: **if** u.edge $\succ_u wu$ or u.edge $= wu$ or u.state $=$ `DONE` **then**
14: `Update`(w)

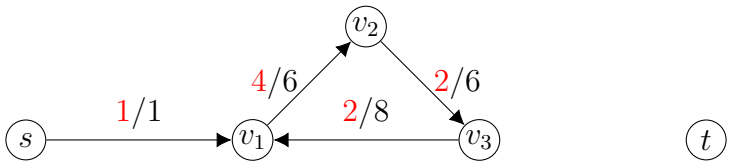
Example 5



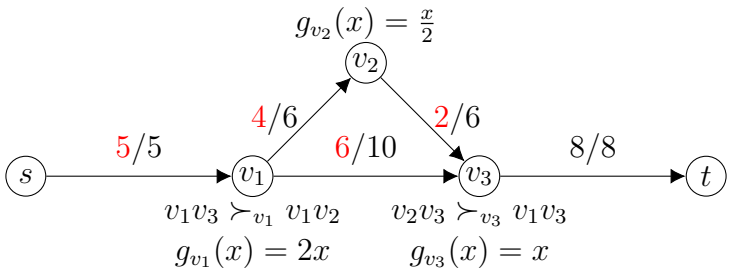
In this example, we start with f equals to zero flow and augment the first s-t-path (s, v_1, v_3, t) because v_1v_3 is the current most preferred edge of v_1.



After the augmentation, v_3t is saturated. v_3 is running out of edges to propose to, so it reaches the **REJECT** *state. v_1v_3 is the least preferred edge currently accepted by v_3 so v_3 is going to reject v_1v_3. v_1 can no longer propose to v_1v_3 because $v_2v_3 \succ_{v_3} v_1v_3$. v_1 moves to the next preferred edge v_1v_2. The following is the auxiliary graph H.*



Next augment σ-cycle (s, v_1, v_2, v_3, v_1) with value $(1, 4, 2, 2)$ (see section 4.2.2 for how this value is obtained) such that the outflow of v_1 is twice of the inflow of v_1 as the following:



After the augmentation, sv_1 is saturated and s is running out of edges to propose to. Therefore, s reaches the **REJECT** *state and the algorithm terminates.*

4.2 Details of Path Augmentation

Before describing how an s-t-path or a σ-cycle $P = (v_0, v_1, ..., v_k)$ $(v_0 = s)$ is augmented with vector $(\Delta_0, \Delta_1, ..., \Delta_{k-1})$ where edge $v_{i-1}v_i$ is augmented with value Δ_i for $1 \leqslant i \leqslant k$, since the augmented value for each edge along P is always proportional, it will be useful to keep a *rate* field v.rate for each $v \in V \cup \{s\}$. Each vertex v has a state, an associated edge, and a rate field. v_{i-1}.rate is a scaling factor of Δ_{i-1}.

4.2.1 Augmenting an s-t-path

While augmenting an s-t-path $P = (v_0, v_1, ..., v_k)$ $(v_0 = s)$, $v_k = t$ and no vertices in P are visited twice. Starting from v_0, set v_0.rate $= 1$ and traverse through P to get the rate field of each vertex. Eventually, augment flow along P by $(v_0.\text{rate}, ..., v_{k-1}.\text{rate})$ calculated by Procedure 3 times some quantity Δ decided by a *bottleneck edge* $v_{i-1}v_i$ as in Procedure 2.

Herein, each vertex $v \in V$ is associate with a linear mapping $g_v(x) = a_v x$. While augmenting an s-t-path, we consider the following three cases:

Push flow : If v_{i-1} and v_i are at `PROPOSE` state, then $v_i.\text{rate} = a_{v_i} \times v_{i-1}.\text{rate}$.

Redirect flow : If v_{i-1} and v_i are at different state, then $v_i.\text{rate} = a_{v_i}^{-1} \times v_{i-1}.\text{rate}$.

Remove flow : If v_{i-1} and v_i are at `REJECT` state, then $v_i.\text{rate} = a_{v_i}^{-1} \times v_{i-1}.\text{rate}$.

In the push flow case, v_{i-1} proposes to v_i while v_i proposes to another vertex, so Δ_i must be a_{v_i} times Δ_{i-1}. Since Δ_i will be determined eventually depending on the quantity Δ, we keep track of $v_i.\text{rate}$ instead, that is, we set $v_i.\text{rate} = a_{v_i} \times v_{i-1}.\text{rate}$.

In the redirect flow case, one of the states of v_{i-1} and v_i is `PROPOSE` and the other is `REJECT`. v_i is either proposing the same flow value as rejected from v_{i-1} or rejecting the same flow value as proposed by v_{i-1}.

In the remove flow case, v_{i-1} rejects $v_{i-1}v_i$ while v_i rejects v_iv_{i+1}, so we will update Δ_i and Δ_{i-1} in a reverse manner, that is, set $v_i.\text{rate} = a_{v_i}^{-1} \times v_{i-1}.\text{rate}$.

The multipliers for the rate fields depend on the states of v_{i-1} and v_i, and base on these two states, we can define the `status` function as in Procedure 4 that decides the multipliers of the rate fields.

Procedure 2 : `Augment_`s-t-`path`(P) — $P = (v_0, ..., v_k)$ where $v_0 = s$ and $v_k = t$

1: $(v_0.\text{rate}, ..., v_{k-1}.\text{rate}) \leftarrow$ `get_rate`(P)
2: $\Delta \leftarrow \arg\min_{1 \leqslant i \leqslant k} \frac{c_f(v_{i-1}v_i)}{v_{i-1}.\text{rate}}$
3: $(\Delta_0, \Delta_1, ..., \Delta_{k-1}) \leftarrow \Delta \times (v_0.\text{rate}, ..., v_{k-1}.\text{rate})$
4: Augment f by along P by $(\Delta_0, \Delta_1, ..., \Delta_{k-1})$

Procedure 3 : `get_rate`(P) — $P = (v_0, ..., v_k)$

1: $r_0 \leftarrow 1$
2: **for** $i = 1$ to k **do**
3: $\quad r_i \leftarrow$ `status`$(v_{i-1}v_i) \times r_{i-1}$
4: **return** $(r_0, ..., r_{k-1})$

Procedure 4 : `status`$(v_{i-1}v_i)$ — $v_{i-1}v_i$ is an edge in H

1: **if** $v_{i-1}.\text{state} = v_i.\text{state} =$ `PROPOSE` **then return** a_{v_i}
2: **else if** $v_{i-1}.\text{state} \neq v_i.\text{state}$ **then return** 1
3: **else if** $v_{i-1}.\text{state} = v_i.\text{state} =$ `REJECT` **then return** $a_{v_i}^{-1}$

4.2.2 Augmenting a σ-cycle

While augmenting a σ-cycle $P = (v_0, v_1, ..., v_k)$ $(v_0 = s)$, $v_k = v_j$ for some $0 \leqslant j < k$. For $i \neq j$ and $i \neq k$ (we will mostly use j instead of k in later argument), the behavior of edges adjacent to v_i is similar to the s-t-path augmentation. However, for v_j, the situation is more complicated. v_j has two incoming edges $v_{j-1}v_j$ and $v_{k-1}v_j$ in H. $v_{j-1}v_j$ is the last edge of the path part of σ-cycle P while $v_{k-1}v_j$ is the last edge of the cycle part.

Regardless of the path part, we can first treat the cycle part $(v_j, ..., v_k)$ as a path and get v_j.rate and v_{k-1}.rate like the s-t-path augmentation. We can obtain v_{j-1}.rate by Procedure 6 depending on v_{j-1}.state, v_{k-1}.state, and v_j.state by equation 4, 5, and 6.

1. If v_{j-1}.state $= v_{k-1}$.state then:

$$v_j.\text{rate} = \texttt{status}(v_{j-1}v_j) \times (v_{j-1}.\text{rate} + v_{k-1}.\text{rate})$$

$$v_{j-1}.\text{rate} = \frac{v_j.\text{rate}}{\texttt{status}(v_{j-1}v_j)} - v_{k-1}.\text{rate} \tag{4}$$

2. If v_{j-1}.state $\neq v_{k-1}$.state and v_j.state $= v_{j-1}$.state, then:

$$v_j.\text{rate} - v_{k-1}.\text{rate} = \texttt{status}(v_{j-1}v_j) \times v_{j-1}.\text{rate}$$

$$v_{j-1}.\text{rate} = \frac{v_j.\text{rate} - v_{k-1}.\text{rate}}{\texttt{status}(v_{j-1}v_j)} \tag{5}$$

3. If v_{j-1}.state $\neq v_{k-1}$.state and v_j.state $= v_{k-1}$.state, then:

$$v_j.\text{rate} = \texttt{status}(v_{k-1}v_j) \times (v_{k-1}.\text{rate} - v_{j-1}.\text{rate})$$

$$v_{j-1}.\text{rate} = v_{k-1}.\text{rate} - \frac{v_j.\text{rate}}{\texttt{status}(v_{k-1}v_j)} \tag{6}$$

In the first case, v_{j-1} and v_{k-1} are at the same state, so the augmented flow from v_j depends on the total augmented flow of $v_{j-1}v_j$ and $v_{k-1}v_k$.

For the second case, since v_j and v_{k-1} are at different state, the augmented flow from v_{j-1} depends on the flow difference between the flow from v_j and the flow from v_{k-1}.

In the third case, since v_{j-1} and v_{k-1} are at different state, the augmented flow from v_{j-1} depends on the flow difference between the flow from v_{k-1} and the flow from v_{j-1}.

For the path part, similar to Procedure 3 in the s-t-path augmentation, find the rate proportion of vertices $v_0, ..., v_{j-1}$ as in s-t-path augmentation and scale $(v_0.\text{rate}, ..., v_{j-2}.\text{rate})$ accordingly to the previously caluculated v_{j-1}.rate. We only check this when $v_j \neq v_0$.

Eventually, the bottleneck edges can belong to either the cycle part $(v_j, ..., v_k)$ or the path part $(v_0, ..., v_j)$. Augment flow along P proportionally to $(v_0.\text{rate}, ..., v_{k-1}.\text{rate})$ times some quantity Δ decided by some bottleneck edges $v_{i-1}v_i$ as in Procedure 5. Note that $v_{j-1}.\text{rate}$ can be a non-positive value. If this happens, then the bottleneck edges belong to the cycle part. Besides, if $v_j = v_0 = s$, then the σ-cycle is just a cycle.

Procedure 5 : `Augment_`σ`-cycle`(P) $\quad P = (v_0, ..., v_j, ..., v_k)$ where $v_0 = s$ and $v_k = v_j$

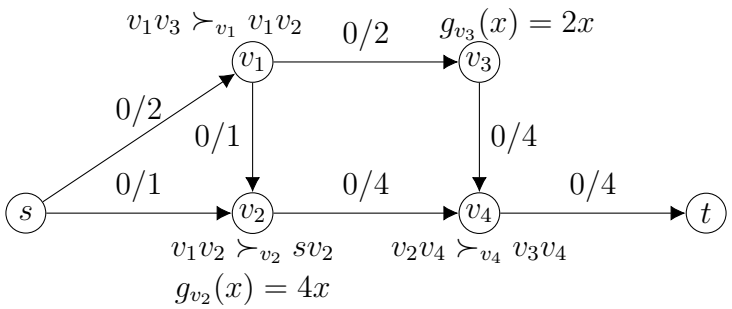
$(v_j.\text{rate}, ..., v_{k-1}.\text{rate}) \leftarrow$ `get_rate`$((v_j, ..., v_k))$
if $v_j \neq v_0$ **then**
    $v_{j-1}.\text{rate} \leftarrow \sigma$`_rate`$(v_{j-1}, v_j, v_{k-1})$
    $(r_0, ..., r_{j-1}) \leftarrow$ `get_rate`$((v_0, ..., v_j))$
    $(v_0.\text{rate}, ..., v_{j-2}.\text{rate}) \leftarrow \frac{v_{j-1}.\text{rate}}{r_{j-1}} \times (r_0, ..., r_{j-1})$
if $v_{j-1}.\text{rate} \leqslant 0$ **then**
    $\Delta \leftarrow \arg\min_{j+1 \leqslant i \leqslant k} \frac{c_f(v_{i-1}v_i)}{v_{i-1}.\text{rate}}$
else
    $\Delta \leftarrow \arg\min_{1 \leqslant i \leqslant k} \frac{c_f(v_{i-1}v_i)}{v_{i-1}.\text{rate}}$
$(\Delta_0, \Delta_1, ..., \Delta_{k-1}) \leftarrow \Delta \times (v_0.\text{rate}, ..., v_{k-1}.\text{rate})$
Augment f by along P by $(\Delta_0, \Delta_1, ..., \Delta_{k-1})$

Procedure 6 : σ`_rate`(v_{j-1}, v_j, v_{k-1}) $\quad v_{j-1}v_j$ and $v_{k-1}v_j$ are edges in H

if If $v_{j-1}.\text{state} = v_{k-1}.\text{state}$ **then**
    return $\frac{v_j.\text{rate}}{\texttt{status}(v_{j-1}v_j)} - v_{k-1}.\text{rate}$
else if $v_j.\text{state} = v_{j-1}.\text{state}$ **then**
    return $\frac{v_j.\text{rate} - v_{k-1}.\text{rate}}{\texttt{status}(v_{j-1}v_j)}$
else if $v_j.\text{state} = v_{k-1}.\text{state}$ **then**
    return $v_{k-1}.\text{rate} - \frac{v_j.\text{rate}}{\texttt{status}(v_{k-1}v_j)}$

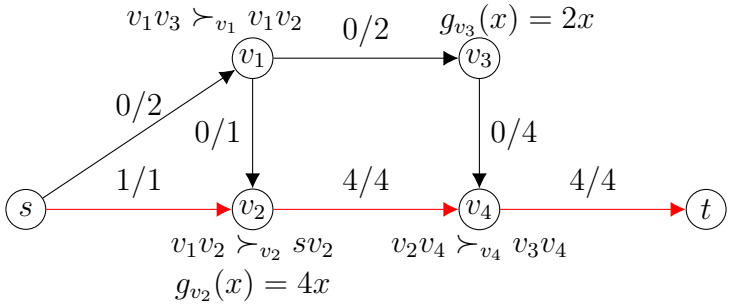
Example 6 and Example 7 below illustratre Algorithm 1.

Example 6

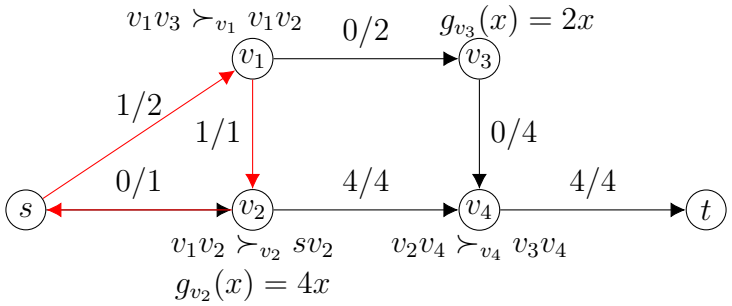


In the given AL-network, inflow is equal to outflow for both v_1 and v_4. The preference of s is arbitrary, we will work on different orders: starting with sv_1 or starting with sv_2.

Starting with sv_2, we augment (s, v_2, v_4, t) and get the following:

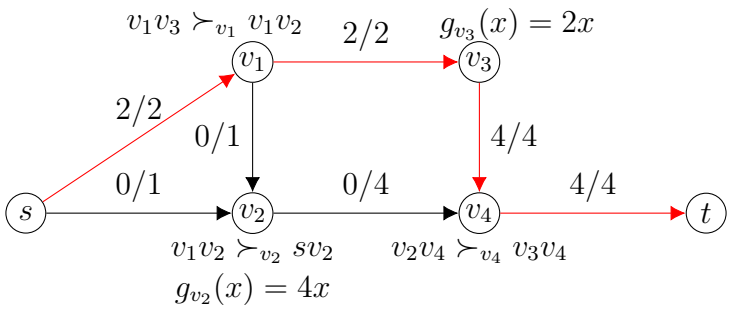


According to Algorithm 1, we **Update**(v_4) *because v_4t is saturated. v_4.state becomes* **REJECT** *and v_4.edge is v_2v_4. v_3.edge $= v_3v_4$ and $v_2v_4 \succ_{v_4} v_3v_4$, this triggers* **Update**(v_3) *and v_3.state becomes* **DONE**. *As a consequence, the update of v_3 also triggers* **Update**(v_1) *and v_1.edge becomes v_1v_2. Then we* **Update**(v_2) *because v_2v_4 is saturated. v_2.state becomes* **REJECT** *and v_2.edge is sv_2. s can no longer propose to sv_2, so it moves on to sv_1 and augments (s, v_1, v_2, s) with $(1, 1, 1)$ according to Procedure 5.*

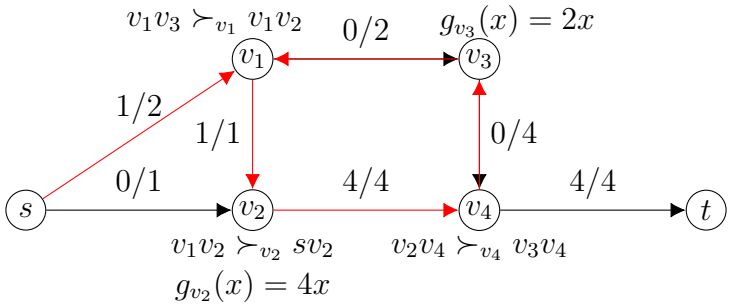


v_1 is running out of edges to propose so it reaches the **REJECT** *state. This triggers* **Update**(s) *and s also reaches the* **REJECT** *state. Algorithm 1 terminates and the result is a stable flow.*

If s starts with sv_1, it will first augment (s, v_1, v_3, v_4, t):

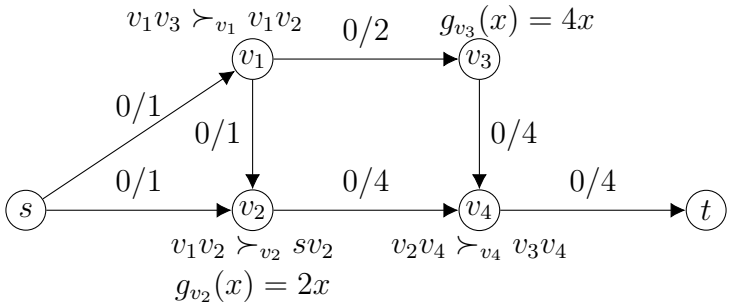


We **Update**(v_4) *because* v_4t *is saturated.* v_4*.state becomes* **REJECT** *and* v_4*.edge is* v_3v_4*.* v_3v_4 *is rejected by* v_4*, so we* **Update**(v_3) *and* v_3*.state becomes* **REJECT** *and* v_3*.edge is* v_1v_3*.* v_1v_3 *is rejected by* v_3*, so* v_1 *moves on to* v_1v_2*. At this point, we have a* σ*-cycle* $(s, v_1, v_2, v_4, v_3, v_1)$ *to augment according to line 7 of Algorithm 1.* v_3*.state =* **REJECT***,* v_1*.state =* **PROPOSE***, and* s*.state =* **PROPOSE***, so according to Procedure 5 and equation 5, we augment* $(-1, 1, 4, 4, 2)$*:*

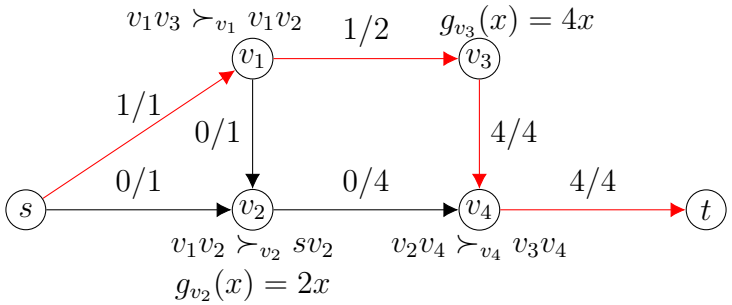


After the augmentation, v_3 *rejects all the flow from* v_1v_3 *so it reaches the* **DONE** *state.* v_4 *rejects all the flow from* v_3v_4*, so* v_4*.edge becomes* v_2v_4*.* v_2*.state becomes* **REJECT** *because* v_2 *is running out of edges to propose to.* v_1 *is also running out of edges to propose to, so* v_1*.state also becomes* **REJECT***. This triggers* s *to move on to* sv_2*. However,* v_2 *will reject* s *as* v_2 *prefers* v_1v_2*. Thus,* s*.state becomes* **REJECT** *so Algorithm 1 terminates and this result is a stable flow.*

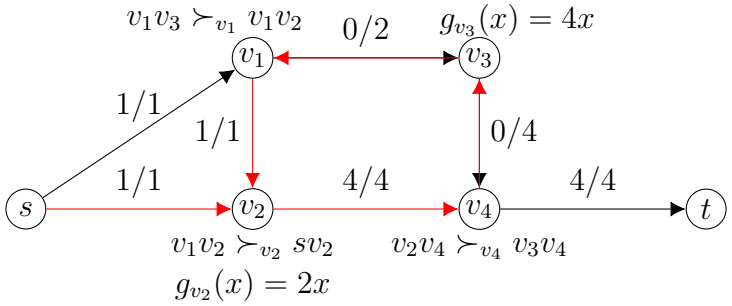
EXAMPLE 7



Let us consider the graph similar to Example 6, except v_2 and v_3 swap their linear mappings, s starts with sv_1, and the capacity of sv_1 is brought down to 1. The first path to augment is (s, v_1, v_3, v_4, t).



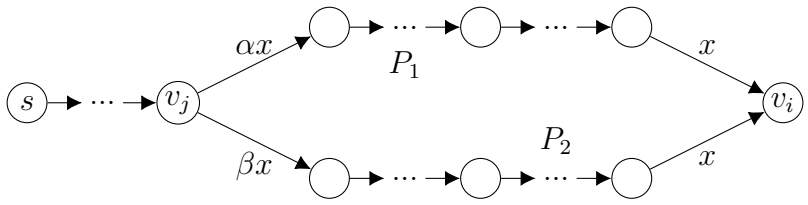
By similar reasoning in Example 4, we have an σ-cycle $(s, v_1, v_2, v_4, v_3, v_1)$ to augment. According to Procedure 5, the augmenting rate is $(\frac{1}{2}, 1, 2, 2, \frac{1}{2})$ for the edges of $(s, v_1, v_2, v_4, v_3, v_1)$. However, sv_1 is saturated and s.rate is positive, so this path is not augmentable. According to line 7 in Algorithm 1, we have to move on and `Update`(s). *The next σ-cycle to augment is (s, v_2, v_4, v_3, v_1). According to Procedure 5, we augment $(1, 4, 4, 1, 1)$ and the result is a stable flow.*



Remark. AL-SF is a special case of the traditional stable flow problem studied in Cseh and Matuschke [2013]. Note that in equation 4, 5, and 6, the `status` and rate values are always 1. While augmenting a σ-cycle, either $v_j = v_0$ or v_{j-1}.rate $= 0$. This implies the augmentation in each iteration of Algorithm 1, as a consequence of flow conservation property, is either on an s-t-path or a cycle. Besides, the path augmenting algorithm in Cseh and Matuschke [2013] is different from Algorithm 1. Instead of updating v_{i-1} one at a time from the tail of the path where $v_{i-1}v_i$ is saturated, the approach of Cseh and Matuschke [2013] is to update all such v_{i-1}'s and move to the next iteration. This works for the traditional stable flow problem because it does not matter where the path starting from s enters the cycle. However, it matters for AL-networks.

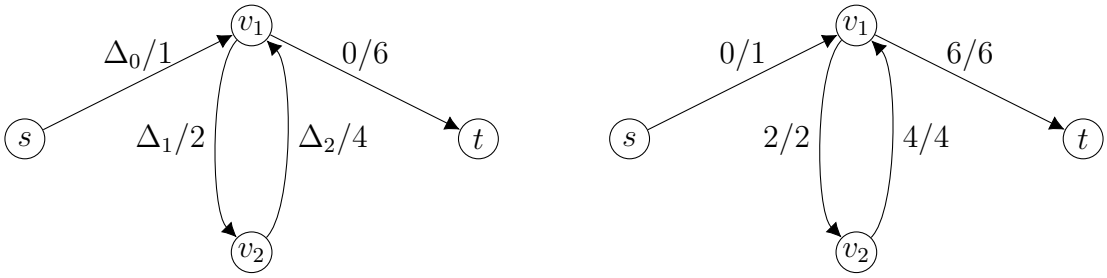
Remark. While augmenting a σ-cycle $(s, v_1, ..., v_j, ..., v_k)$ where $v_k = v_j$, the augmented value for the path part $(s, v_1, ..., v_j)$ is negative when v_j finds a "more efficient" path to some v_i where $j < i < k$ and v_i is a redirecting vertex that accepts and rejects the same amount of flow. Such a v_i always exists in AL-networks.

Suppose previously v_j sends a flow αx by path P_1 to v_i and v_i receives flow x. If there is another path P_2 that v_j proposes to later on with flow βx where $0 < \beta < \alpha$ and v_i receives flow x and prefers P_2, then P_2 is more efficient for v_j. This will force v_i to receive flow from P_2 and remove flow from P_1. Since $\beta < \alpha$, the flow from s to v_j is forced to decrease.



However, this is not the case when we consider a cyclic graph where agents apply linear mappings. Consider the following example:

EXAMPLE 8 *In this graph, the outflow is twice of inflow for v_1 and v_2. For the preference, $v_2v_1 \succ_{v_1} sv_1$ and $v_1v_2 \succ_{v_1} v_1t$. Algorithm 1 does not work properly for this graph. The first σ-cycle found is (s, v_1, v_2, v_1). We would like to augment $(\Delta_0, \Delta_1, \Delta_2)$ along this σ-cycle as the left graph shows. The following must be satisfied: $\Delta_1 = 2(\Delta_0 + \Delta_2)$ and $\Delta_2 = 2\Delta_1$. This implies $\Delta_0 = -\frac{3}{2}\Delta_1$ and flows cannot be negative, so it is impossible to augment this σ-cycle such that one edge is saturated. Besides, it is possible that there is no outflow from s or there is no inflow to t in a stable flow. The right graph is a stable assignment.*



Algorithm 1 works properly for AL-networks because for each σ-cycle there is always a redirecting vertex in the cycle part. This is not true for cyclic networks.

Remark. While augmenting a σ-cycle $P = (v_0, ..., v_j, ..., v_k)$, v_{j-1}.state, v_j.state, and v_{k-1}.state are either `PROPOSE` or `REJECT`. Therefore, there are eight cases to consider. These

cases are all covered by equation 4, 5, and 6, listed in the following table.

v_{j-1}.state	v_j.state	v_{k-1}.state	equation	status of v_j
PROPOSE	PROPOSE	PROPOSE	4	push flow
REJECT	REJECT	REJECT	4	remove flow
PROPOSE	REJECT	PROPOSE	4	redirect flow
REJECT	PROPOSE	REJECT	4	redirect flow
PROPOSE	PROPOSE	REJECT	5	push flow (redirect flow if v_{j-1}.rate $= 0$)
REJECT	REJECT	PROPOSE	5	remove flow (redirect flow if v_{j-1}.rate $= 0$)
REJECT	PROPOSE	PROPOSE	6	push flow
PROPOSE	REJECT	REJECT	6	remove flow

4.3 Analysis

We start from proving the correctness of Algorithm 1. To ensure that Algorithm 1 can be executed properly in each iteration, we prove the following lemma:

LEMMA 4.1 *At the beginning of any iteration in Algorithm 1 line 2, H always contains an s-t-path or a σ-cycle.*

Proof. Consider any $v \in V$, v has an outgoing edge if v is at PROPOSE or REJECT state. When v.state = DONE, v rejected its last incoming edge or v got rejected without any incoming flow. There are no incoming edges of v in E_H when v.state = DONE. v does not have incoming and outgoing edges if and only if v.state = DONE.

Before finding an augmenting path, s.state = PROPOSE so s has an outgoing edge in H. s always reaches vertices that is in either the PROPOSE or REJECT state and each such vertex has an outgoing edge. One can always traverse from s until a vertex is visited twice which forms a σ-cycle or until t is reached. ■

THEOREM 4.1 *Algorithm 1 computes an AL-SF in polynomial time.*

Proof. Suppose there is a blocking walk $W = (v_1, ..., v_k)$ in G. Edges in this blocking walk must be unsaturated becuase all the vertices apply linear mapping and all entries in V_W must be positive in point 1 of Definition 2.5. From the second condition in Definition 2.5, either $v_1 = s$ or there is an edge v_1u such that $v_1v_2 \succ_{v_1} v_1u$ and $f(v_1u) > 0$.

If $v_1 \neq s$, v_1v_2 is unsaturated, $v_1v_2 \succ_{v_1} v_1u$, and $f(v_1u) > 0$ either because v_2.state = REJECT and v_2.edge $= v_1v_2$ or v_1.edge was updated once v_2.state changed to REJECT. In the

former case, v_1v_2 was accepted by v_2 with some positive flow but $f(v_1v_2)$ decreased later on. In the later case, v_1 did not get the chance to propose more to v_1v_2 and v_2.edge $= vv_2$ where $v = v_1$ or $vv_2 \succ_{v_2} v_1v_2$. If $v_1 = s$, the termination of Algorithm 1 implies that v_2.state $=$ `REJECT`. In either case, v_2.state $=$ `REJECT` and v_2.edge $= vv_2$ where $v = v_1$ or $vv_2 \succ_{v_2} v_1v_2$.

From v_2.state $=$ `REJECT` and v_2v_3 is unsaturated, v_2 proposed to all available vertices and either v_2 has proposed to v_3 earlier and $f(v_2v_3)$ decreased later on or v_2.edge was updated once v_3.state changed to `REJECT` and v_2 did not get the chance to make v_2v_3 saturated. In either case v_3.state $=$ `REJECT` and v_3.edge $= vv_3$ where $v = v_2$ or $vv_3 \succ_{v_3} v_2v_3$.

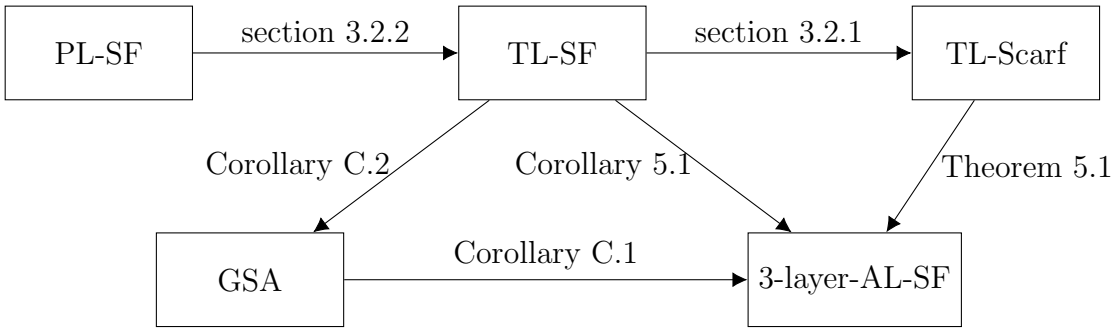
By continuing analogous arguments, v_k.state $=$ `REJECT` and v_k.edge $= vv_k$ where $v = v_{k-1}$ or $vv_k \succ_{v_k} v_{k-1}v_k$. From the third condition of Definition 2.5, either $v_k = t$ or there is an edge wv_k such that $v_{k-1}v_k \succ_{v_k} wv_k$ and $f(wv_k) > 0$. For the former case, t never rejects vertices or forces vertices to update so this cannot happen. For the later case, v_k rejected w earlier, so we have $f(wv_k) = 0$ since v_k.edge $= v_{k-1}v_k$ and $v_{k-1}v_k \succ_{v_k} wv_k$. However, this contradicts to our assumption $f(wv_k) > 0$.

For time complexity, line 6 and 7 in Algorithm 1 will be executed at most $O(|E|)$ times because the total number of `Update` is bounded by the number of edges. This is because for each `Update`, the corresponding edge has to be saturated by either fully proposed or rejected. Augmenting a path in line 4 or checking whether a path is augmentable in line 7 takes $O(|V|)$ time. Therefore the running time of Algorithm 1 is $O(|E||V|)$. ■

5 Reductions of Stable Flow Problems

As mentioned in the beginning of section 4, we show a reduction from PL-SF and TL-SF to AL-SF, which is solvable by Algorithm 1. We denote a Scarf's instance that corresponds to a TL-SF instance *TL-Scarf*. The intermediate step is to convert a TL-Scarf instance to an AL-SF instance. In particular, the eventual AL-network consists of three layers of vertices. We denote the problem of finding stable flow in such networks *3-layer-AL-SF*. In Appendix C, we further show a reduction from TL-SF to the *generalized stable allocation* (GSA) problem introduced in Dean and Swar [2009], and a reduction from GSA to 3-layer-AL-SF.

The following flowchart is an overview of the reductions between different problems.



5.1 From TL-SF to 3-layer-AL-SF

This reduction consists of two steps, from TL-SF to TL-Scarf shown in section 3.2.1 and from TL-Scarf to 3-layer-AL-SF. Herein, we focus on the later part.

We will use the same notation as in section 3.2.1 for the TL-Scarf instance with matrix A and vector b. Beside the TL-Scarf instance, we are also given the corresponding TL-network (G, s, t, c). To construct the three-layer AL-network (G', s', t', c') where $G' = (V' \cup \{s', t'\}, E')$, for the vertex part V':

1. Create s', s_{out}, t', and t_{in}.
2. For each row v^{out}, create vertex v_{out}.
3. For each row v^{in}, create vertex v_{in}.
4. For each column x_e, create vertex m_e.
5. For each column x_v^{out}, create vertex m_v^{out}.
6. For each column x_v^{in}, create vertex m_v^{in}.

Starting from source s', the first layer consists of the outgoing vertices s_{out} and v_{out}, the second layer consists of the middle vertices m_e, m_v^{out}, and m_v^{in}, and the third layer consists of the incoming vertices t_{in} and v_{in}, which evetually merge together and end at sink t'. For the edge part E' and capacities c':

1. Set $c'(s's_{out}) = \sum_{su \in E} b_{su} + 1$, and set $c'(s'v_{out}) = b_{v^{out}}$.
2. Set $c'(t_{in}t') = \sum_{ut \in E} b_{ut} + 1$, and set $c'(v_{in}t') = b_{v^1}$.
3. For each $e = uv \in E$, if $v \neq t$, set $c'(u_{out}m_e) = b_e$ and $c'(m_e v_{in}) = A_{v^{in}x_e} b_e$; if $v = t$, set $c'(u_{out}m_e) = c'(m_e t_{in}) = b_e$.

4. For each $v \in V$, set $c'(v_{out}m_v^{in}) = c'(v_{out}m_v^{out}) = c'(m_v^{in}v_{in}) = c'(m_v^{out}v_{in}) = \max\{b_{v^{in}}, b_{v^{out}}\}$.

Point 1 sets the edge capacities between s' and the first layer vertices while point 2 sets the edge capacities between the third layer vertices and t'. Point 3 and point 4 set the edge capacities from the first to the second layer and from the second to the third layer. The capacities are set in a way such that any vertex v_{out} in the first layer must send some positive flow to either $v_{out}m_v^{in}$ or $v_{out}m_v^{out}$. Similarly, any vertex v_{in} in the third layer must receive some positive flow from either $m_v^{in}v_{in}$ or $m_v^{out}v_{in}$. Therefore, the setting of the linear mapping for each vertex is:

1. The outflow of s_{out}, t_{in}, v_{in}, v_{out}, m_v^1, m_v^2, and m_e where $e = vt$ are the same as their inflow.

2. The outflow of m_e where $e = vw$ and $w \neq t$ is $A_{w^1x_e}$ times of its inflow.

Eventually, base on the setting of the TL-Scarf instance, we set the preference of vertices as the following:

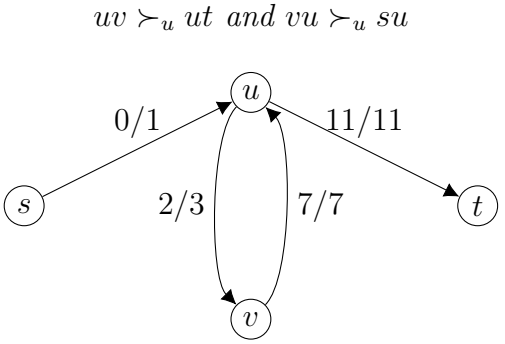
1. For each $v \in V$, v_{out} prefers $v_{out}m_v^{out}$ the most and $v_{out}m_v^{in}$ the least, the preference of $v_{out}m_e$ for some $e = vu$ is the same as in row v^{out}.

2. For each $v \in V$, v_{in} prefers $m_v^{in}v_{in}$ the most and $m_v^{out}v_{in}$ the least, the preference of m_ev_{in} for some $e = uv$ is the same as in row v^{in}.

3. The preference of s_{out} and t_{in} is arbitrary.

Suppose we have a 3-layer-AL-SF f' of this network, we show that by setting ${x^*}_e = f'(u_{out}m_e)$ for $e = uv$, ${x^*}_v^{in} = f'(v_{out}m_v^{in})$, and ${x^*}_v^{out} = f'(v_{out}m_v^{out})$, we have a solution for TL-Scarf. Conversely, the solution of TL-Scarf also corresponds to a stable flow. Before showing the proof, it may be useful to look over Example 9.

EXAMPLE 9 *Consider the following cyclic TL-network with its stable flow assignment:*



$$g_u(x) = \begin{cases} 0 & \text{if } x < 0.5, \\ 2x - 1 & \text{otherwise.} \end{cases}$$

$$g_v(x) = \begin{cases} [0, 1] & \text{if } x = 0, \\ 3x + 1 & \text{otherwise.} \end{cases}$$

By the construction in section 3.2.1, we have the following TL-Scarf:

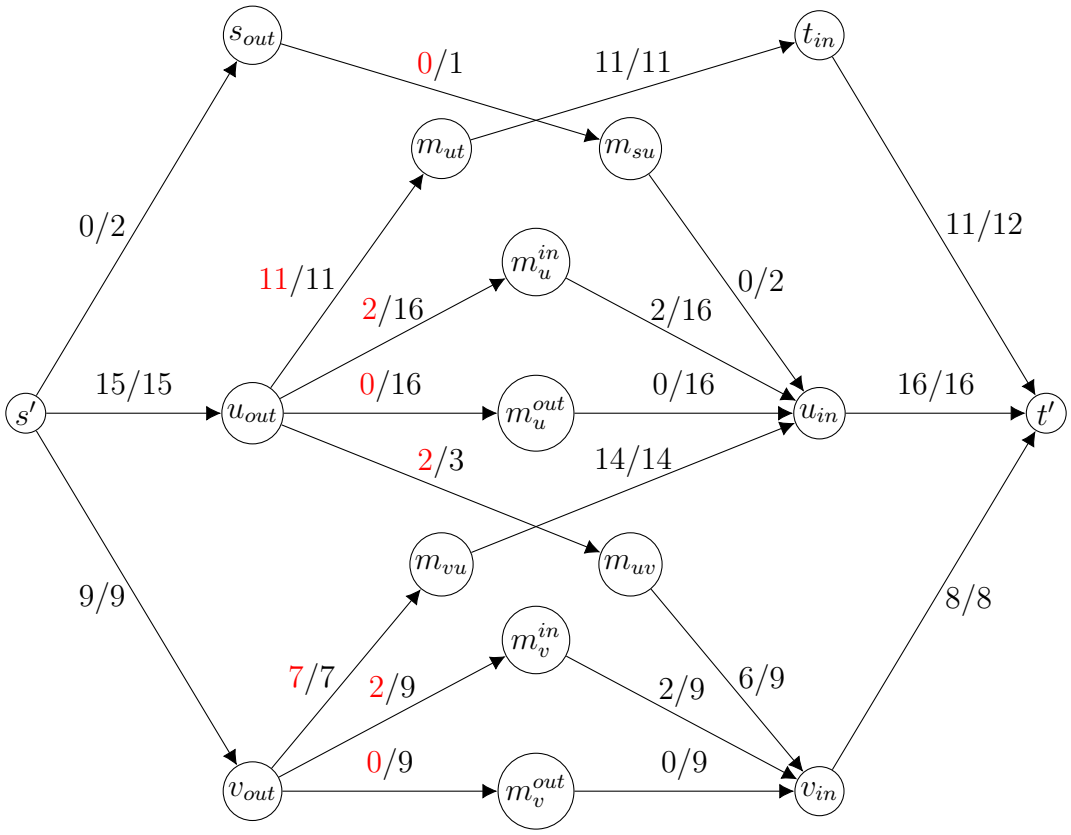
$$
Ax^* = \begin{array}{c} \\ su \\ uv \\ vu \\ ut \\ u^{in} \\ u^{out} \\ v^{in} \\ v^{out} \end{array}
\begin{array}{c}
\begin{array}{cccccccc} x_{su} & x_{uv} & x_{vu} & x_{ut} & x_u^{in} & x_u^{out} & x_v^{in} & x_v^{out} \end{array} \\
\left(\begin{array}{cccc|cc|cc}
1 & & & & & & & \\
 & 1 & & & & & & \\
 & & 1 & & & & & \\
 & & & 1 & & & & \\
\hline
2 & & 2 & & 1 & 1 & & \\
 & 1 & & 1 & 1 & 1 & & \\
\hline
 & 3 & & & & & 1 & 1 \\
 & & 1 & & & & 1 & 1
\end{array}\right)
\end{array}
\begin{pmatrix} 0 \\ 2 \\ 7 \\ 11 \\ 2 \\ 0 \\ 2 \\ 0 \end{pmatrix}
= \begin{pmatrix} 0 \\ 2 \\ 7 \\ 11 \\ 16 \\ 15 \\ 8 \\ 9 \end{pmatrix}
\leqslant \begin{pmatrix} 1 \\ 3 \\ 7 \\ 11 \\ 16 \\ 15 \\ 8 \\ 9 \end{pmatrix}
= \begin{pmatrix} b_{su} \\ b_{uv} \\ b_{vu} \\ b_{ut} \\ b_{u^{in}} \\ b_{u^{out}} \\ b_{v^{in}} \\ b_{v^{out}} \end{pmatrix}
$$

Where the preference of each row over columns from the highest to the lowest is:

row	u^{in}	u^{out}	v^{in}	v^{out}
column	$x_u^{in}, x_{vu} x_{su}, x_u^{out}$	$x_u^{out}, x_{uv}, x_{ut}, x_u^{in}$	$x_v^{in}, x_{uv}, x_v^{out}$	$x_v^{out}, x_{vu}, x_v^{in}$

By the aforementioned construction, we obtain the corresponding three-layer AL-network and the preference of each vertex as the following:

vertex	*preference*
u_{out}	$u_{out} m_u^{out} \succ_{u_{out}} u_{out} m_{uv} \succ_{u_{out}} u_{out} m_{ut} \succ_{u_{out}} u_{out} m_u^{in}$
u_{in}	$m_u^{in} u_{in} \succ_{u_{in}} m_{vu} u_{in} \succ_{u_{in}} m_{su} u_{in} \succ_{u_{in}} m_u^{out} u_{in}$
v_{out}	$v_{out} m_v^{out} \succ_{v_{out}} v_{out} m_{vu} \succ_{v_{out}} v_{out} m_v^{in}$
v_{in}	$m_v^{in} v_{in} \succ_{v_{in}} m_{uv} v_{in} \succ_{v_{in}} m_v^{out} v_{in}$



The linear mapping of each vertex v in the three-layer AL-network is:

v	m_{su}	m_{vu}	m_{uv}	*others*
$g_v(x)$	$2x$	$2x$	$3x$	x

Note that for vertices not listed in the table, the inflow is equal to the outflow.

We can see that the three-layer AL-network captures the properties of the TL-Scarf. The capacities of edges adjacent to s' and t' are set to large value such that $s's_{out}$ and $t_{in}t$ cannot be saturated, and for u in the original TL-network, there must be some flow passing m_u^{in} or m_u^{out}, similar arguement holds for v. The preference of each vertex simply follows the row preference in TL-Scarf. The linear mapping of each vertex is set in a way that follows the coefficient given in TL-Scarf.

The three-layer AL-network is a special case of AL-network. Therefore, flow stability can be found by using Algorithm 1.

THEOREM 5.1

f' is an AL-SF of the above mentioned three-layer AL-network if and only if x^* is a solution of TL-Scarf.

Proof. TL-Scarf $\rightarrow$ 3-layer-AL-SF:

Suppose x^* is a solution of TL-Scarf, then x^* dominates every column of A. We set f' as the following:

1. For $e = uv \in E$, if $v \neq t$ set $f'(u_{out}m_e) = {x^*}_e$ and $f'(m_e v_{in}) = A_{v^{in}x_e}{x^*}_e$; if $v = t$ set $f'(u_{out}m_e) = f'(m_e v_{in}) = {x^*}_e$.

2. For each $v \in V$, set $f'(v_{out}m_v^{in}) = f'(m_v^{in}v_{in}) = {x^*}_v^{in}$ and $f'(v_{out}m_v^{out}) = f'(m_v^{out}v_{in}) = {x^*}_v^{out}$.

3. Set $f'(s'v_{out}) = \sum_{e=vu\in E} A_{v^{out}x_e}{x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} = \sum_{e=vu\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out}$.

4. Set $f'(v_{in}t') = \sum_{e=uv\in E} A_{v^{in}x_e}{x^*}_e + {x^*}_v^{in} + {x^*}_v^{out}$.

5. Set $f'(s's_{out}) = \sum_{e=sv\in E} A_{v^{out}x_e}{x^*}_e = \sum_{e=sv\in E} {x^*}_e$.

6. Set $f'(t_{in}t') = \sum_{e=vt\in E} {x^*}_e$.

We can see that feasibility is satisfied in the three-layer AL-network. The remaining is to show that f' is stable. Assume there is a blocking walk W in this network. Observing the structure of this three-layer network, each vertex m_e, m_v^{in}, or m_v^{out} in the second layer only have one incoming and one outgoing edge. This indicates they cannot be the starting or ending vertex of the blocking walk. Besides, W cannot start from s' and end at a vertex in the first layer since each vertex in that layer has only one incoming edge. Similarly, W cannot start from a vertex in the third layer and end at t' as each vertex in the third layer has only one outgoing edge. The followings are the remaining cases of W. We show a proof by contradiction for each case.

1. W starts from s' and ends at a vertex v_{in} in the third layer:

   This means there exists a vertex u_{out} in the first layer such that row u^{out} does not bind since $s'u_{out}$ is not saturated. For $e = uv \in E$, $m_e v_{in}$ and $u_{out}m_e$ are not saturated and v_{in} prefers $m_e v_{in}$ to some other nonzero incoming edges. This indicates column x_e is dominated by neither row e nor row v^{in}. Column x_e is not dominated by x^*, a contradiction.

2. W starts from a vertex u_{out} in the first layer and ends at t':

   This means there exists a vertex v_{in} in the third layer such that row v^{in} does not bind since $v_{in}t'$ is not saturated. For $e = uv \in E$, $m_e v_{in}$ and $u_{out}m_e$ are not saturated and u_{out} prefers $u_{out}m_e$ to some other nonzero outgoing edges. This indicates column x_e

is dominated by neither row e nor row u^{out}. Column x_e is not dominated by x^*, a contradiction.

3. W starts from a vertex u_{out} in the first layer and ends at a vertex v_{in} in the third layer:

   This means for $e = uv \in E$, $m_e v_{in}$ and $u_{out} m_e$ are not saturated so column x_e is not dominated by row e. u_{out} prefers $u_{out} m_e$ to some other nonzero outgoing edges. This indicates column x_e is not dominated by row u^{out}. v_{in} prefers $m_e v_{in}$ to some other nonzero incoming edges. This indicates column x_e is not dominated by row v^{in}. Column x_e is not dominated by x^*, a contradiction.

4. W starts from s' to t':

   This means there are some rows v^{in} and u^{out} that do not bind and for $e = uv \in E$, $m_e v_{in}$ and $u_{out} m_e$ are not saturated which indicates x_e is not dominated by row e. Column x_e is not dominated by x^*, a contradiction.

3-layer-AL-SF $\rightarrow$ TL-Scarf:

Suppose f' is an AL-SF of the three-layer AL-network. For $e = uv \in E$, if $m_e v_{in}$ and $u_{out} m_e$ are saturated, then ${x^*}_e = c'(u_{out} m_e) = b_e$. Column x_e is dominated by row e. Otherwise, the following two cases cannot happen at the same time or there is a blocking walk (u_{out}, m_e, v_{in}).

1. $u_{out} = s_{out}$ or u_{out} prefers $u_{out} m_e$ to some other nonzero outgoing edges.

2. $v_{in} = t_{in}$ or v_{in} prefers $m_e v_{in}$ to some other nonzero incoming edges.

If the first case happens, v_{in} must prefer all nonzero incoming edges to $m_e v_{in}$ and $v_{in} t'$ must be saturated or the walk $(u_{out}, m_e, v_{in}, t')$ is blocking. This forces $f'(m_v^{in} v_{in}) = c'(v_{in} t') - \sum_{e'=wv_{in} \in E'} f(e')$ which corresponds to row v^{in} binds and dominates column x_e.

If the second case happens, u_{out} must prefer all nonzero outgoing edges to $u_{out} m_e$ and $s' u_{out}$ must be saturated or the walk $(s', u_{out}, m_e, v_{in})$ is blocking. This forces $f(u_{out} m_v^{out}) = c'(s' u_{out}) - \sum_{e'=u_{out} w \in E'} f(e')$ which corresponds to row u^{out} binds and dominates column x_e.

The remaining is to show how f' makes column x_v^{in} and x_v^{out} dominated for each $v \in V$. If both $s' v_{out}$ and $v_{in} t'$ are saturated, then both row v^{in} and v^{out} bind. We know that both walks $(v_{out}, m_v^{in}, v_{in})$ and $(v_{out}, m_v^{out}, v_{in})$ are not blocking. Therefore, both column x_v^{in} and x_v^{out} are dominated. It is not possible that both $s' v_{out}$ and $v_{in} t'$ are not saturated, otherwise

$(s', v_{out}, m_v^{in}, v_{in}, t')$ or $(s', v_{out}, m_v^{out}, v_{in}, t')$ is a blocking walk. The remaining case is exactly one of $s'v_{out}$ and $v_{in}t'$ is saturated.

If $s'v_{out}$ is not saturated, then by the setting of $c'(m_v^{in}v_{in})$ and $c'(v_{out}m_v^{in})$, $m_v^{in}v_{in}$ and $v_{out}m_v^{out}$ cannot be saturated, this forces all the incoming edges of v_{in} except $m_v^{in}v_{in}$ have to be zero. Otherwise, $(s, v_{out}, m_v^{in}, v_{in})$ is a blocking walk since v_{in} prefers $m_v^1 v_{in}$ the most. This also forces $v_{in}t'$ saturated otherwise $(s, v_{out}, m_v^{in}, v_{in}, t)$ is blocking. Consequently, $f(v_{out}m_v^{in}) = f(m_v^{in}v_{in}) = c'(v_{in}t')$ corresponds to row v^{in} binds and dominates column x_v^{in} and x_v^{out}. This can only happen when $c'(s'v_{out}) > c'(v_{in}t')$, i.e. $b_{v^{out}} > b_{v^{in}}$. Similarly, if $v_{in}t'$ is not saturated, then $f(v_{out}m_v^{in}) = f(m_v^{in}v_{in}) = c'(s'v_{out})$ corresponds to row v^{out} binds and dominates column x_v^{in} and x_v^{out}. This can only happen when $c'(s'v_{out}) < c'(v_{in}t')$, i.e. $b_{v^{out}} < b_{v^{in}}$. ■

A TL-Scarf instance is actually defined by a hidden TL-network. Therefore, we can just shorten the two-stage reduction from TL-SF to TL-Scarf and from TL-Scarf to 3-layer-AL-SF into a reduction from TL-SF to 3-layer-AL-SF.

COROLLARY 5.1 *For a TL-network, there is an equivalent three-layer AL-network. That is, an AL-SF of the three-layer AL-network corresponds to TM-SF of the original TL-network.*

See Appendix B for the proof.

5.2 Analysis

We analyze the time complexity for finding both PL-SF and TL-SF.

COROLLARY 5.2 *PL-SF and TL-SF can be found in polynomial time.*

Proof. Let us start with TL-SF. We can simply apply Corollary 5.1 and get an equivalent three-layer AL-network then find AL-SF by Algorithm 1. Suppose we are given a TL-network (G, s, t, c) where $G = (V \cup \{s, t\}, E)$, the equivalent three-layer AL-network has $O(|V| + |E|)$ vertices, $O(|V|)$ for the first and the third layer and $O(|E|)$ for the second layer, and $O(|E|)$ edges. However, since there is only one incoming edge and one outgoing edge for each vertex in the second layer, the length of the path to augment in each iteration is $O(|V|)$. The number of augmentations is $O(|E|)$. Algorithm 1 takes $O(|E||V|)$ time.

Suppose we are given a PL-network with graph (G, s, t, c) where $G = (V \cup \{s, t\}, E)$, first reduce it to a TL-network then reduce the TL-network to a three-layer AL-network.

The final equivalent AL-network will have $O(|E| + K)$ vertices and $O(|E| + K)$ edges, where $K = \sum_{v \in V} k_v$. Recall that k_v is the number of segments for each vertex's piecewise linear mapping, so K is the total number of segments. By the special sturcture of the new AL-network, the length of path to augment in each iteration is $O(|V|)$ since edges from different segments of the same vertex cannot be augmented at the same time, and the number of augmentations is $O(|E| + K)$. Therefore, Algorithm 1 takes $O((|E| + K)|V|)$ time. Note that the information of each linear piecewise segment is a part of the input, so K is polynomial in the size of the input. ■

Remark. Using dynamic tree implementation, we can actually design a faster algorithm by the first approach. We apply the reduction in Corollary 5.1 and focus on augmenting paths in the final three-layer AL-network and modifying dynamic trees. While augmenting a σ-cycle, we will need an auxiliary vertex as a representative of the vertex that is visited twice. Besides, an efficient update of the `rate` field of vertices in the path to augment is also needed. As a path is represented by a tree structure while using dynamic trees, all of the above mentioned can be done in $O(\log |V|)$ time where the length of the path to augment is $O(|V|)$. Therefore, this also yields $O(|E| \log |V|)$ bound for finding TL-SF and $O((|E| + K) \log |V|)$ bound for finding PL-SF.

Corollary 5.3 *TL-SF can be found in $O(|E| \log |V|)$ time and PL-SF can be found in $O((|E| + K) \log |V|)$ time.*

6 The Optimal Stable Flow Problem

Given a network (G, s, t, c), suppose while assigning flow values, we care about not only stability but also the cost. For each edge $e \in E$, $p : E \to \mathbb{R}_{\geqslant 0}$ is the *price per flow*. Given a feasible flow assignment f, the cost of this edge is $f(e)p(e)$. The total cost of the flow assignment is $\sum_{e \in E} f(e)p(e)$. The decision version of the stable flow optimization problem is defined as the following:

Definition 6.1 (The Optimal Stable Flow Problem) *Given a network (G, s, t, c), the price per flow function p, and the budget $B \in \mathbb{R}_{\geqslant 0}$, determine whether there exists a stable flow such that the total cost is at most B.*

We show that the traditional optimal stable flow problem is polynomial time solvable, while the optimal AL-SF, TL-SF, and PL-SF are all NP-complete.

6.1 The Optimal Stable Flow Problem

For this problem, we assume that inflow is equal to outflow for each vertex. Before proving optimal stable flow problem is polynomial time solvable, we introduce the stable allocation problem. This is because the proof is based on the reduction from the stable flow problem to the stable allocation problem shown in Fleiner [2010].

THEOREM 6.1 *The optimal stable flow problem is polynomial time solvable.*

Proof. We reduce this problem to the optimal stable allocation problem. Suppose we are given (G, s, t, c), $G = (V \cup \{s, t\}, E)$, $\succ_v$ for each $v \in V$, and the cost function p, construct the optimal stable allocation instance as the following:

1. Let $I = \{i_s\} \cup \{i_v | v \in V\}$ and $J = \{j_t\} \cup \{j_v | v \in V\}$.
2. Let $C = \{i_w j_v | wv \in E\} \cup \{i_v j_v | v \in V\} \cup \{j_v i_v | v \in V\}$.
3. Set $u(i_w j_v) = c(wv)$.
4. Let $M(v) = \max\{\sum_{wv \in E} c(wv), \sum_{vw \in E} c(vw)\} + 1$ for each $v \in V \cup \{s, t\}$.
5. Set $q(i_v) = M(v)$ for each $i_v \in I$ and $q(j_v) = M(v)$ for each $j_v \in J$.
6. Set $u(i_v j_v) = u(j_v i_v) = \max\{\sum_{wv \in E} c(wv), \sum_{vw \in E} c(vw)\} + 1$ for each $v \in V \cup \{s, t\}$.
7. For each $w \in V$, i_w prefers $i_w j_w$ the most and $j_w i_w$ the least, the preference of i_w for $i_w j_v$ where $wv \in V$ is the same as $\succ_w$.
8. For each $v \in V$, j_v prefers $j_v i_v$ the most and $i_v j_v$ the least, the preference of j_v for $i_w j_v$ where $wv \in E$ is the same as $\succ_v$.
9. Let $h(i_w j_v) = c(wv)$ for each $wv \in E$. The costs of the rest of the edges in C are 0.

By solving the optimal stable allocation instance and setting $f(wv) = x(i_w j_v)$, we obtain the stable flow with the minimum cost. See the details in Fleiner [2010] for to proof of stability. Clearly, the costs of the stable flow and the stable allocation are the same. The optimal stable allocation problem is polynomial time solvable according to Dean and Munshi [2010], so the reduction indicates that the optimal stable flow problem can also be solved in polynomial time. ■

6.2 Optimality of Generalized Stable Flow

To show that the optimal PL-SF, TL-SF, and AL-SF are all NP-complete, we start with a problem that is NP-hard, the *optimal generalized stable allocation* (optimal GSA) problem.

6.2.1 The Generalized Stable Allocation Problem

The original stable allocation problem was stated in Baïou and Balinski [2002]. Here we consider a more general case, the *generalized stable allocation* (GSA) problem. With a bit abuse of notation, GSA may stand for the problem itself or an allocation that is stable. The problem setting includes two finite disjoint sets I and J of jobs and machines respectively and an edge set C (parallel edges are allowed) that forms a bipartite graph $G = (I \cup J, C)$. The function $u : C \to \mathbb{R}_+$ maps an egde to its capacity. Each job $i \in I$ has its own preference $\succ_i$ on edges in C that have i as an endpoint, and each machine $j \in J$ has its own preference $\succ_j$ on edges in C that have j as an endpoint. Each edge has a multiplier $\mu(ij) > 0$ that tells us one unit of job i uses up $\mu(ij)$ units of capacity when assigned to machine j.

Definition 6.2 *Let $x(ij)$ denote the quantity that job i is assigned to machine j, an* **assignment** *x is* **feasible** *if all of the followings hold:*

$$\sum_{j \in J} x(ij) \leqslant 1 \qquad \forall i \in I \tag{7}$$

$$\sum_{i \in I} \mu(ij) x(ij) \leqslant 1 \qquad \forall j \in J \tag{8}$$

$$0 \leqslant x(ij) \leqslant u(ij) \qquad \forall ij \in C \tag{9}$$

If the equality holds in equation 7, then job i is *x-saturated*. Similarly, if the equality holds in equation 8, then machine j is x-saturated. Edge ij is saturated if $x(ij) = u(ij)$ in equation 9. Now we can define stability:

Definition 6.3 *An assignment x is* **stable** *if it is feasible and for each $ij \in C$, at least one of the followings hold:*

1. *ij is saturated.*
2. *i is x-saturated and $ij' \succ_i ij$ holds for any other $j' \neq j$ such that $x(ij') > 0$.*
3. *j is x-saturated and $i'j \succ_j ij$ holds for any other $i' \neq i$ such that $x(i'j) > 0$.*

In short, an assignment is stable if there is no blocking edge. A blocking edge is an unsaturated pair where the involved job and machine both mutually prefer this edge to some of their other currently assigned edges.

Remark. Our setting is different from the one in Dean and Swar [2009] but they are equivalent. In their setting, adding a dummy job and a dummy machine always guarantees that all jobs and all machines except the dummy machine are x-saturated in a feasible assignment.

The optimal GSA problem is the same as GSA except each edge $ij \in C$ has a cost $h(ij)$ where $h : C \to \mathbb{R}_{\geqslant 0}$. The total cost of an assignment x is $\sum_{ij \in C} h(ij)x(ij)$. We consider not only stability but also whether the total cost is less than a given budget H where $H \in \mathbb{R}_{\geqslant 0}$. Dean and Swar [2009]) show a reduction of the optimal GSA problem to the subset sum problem to obtain the following result.

THEOREM 6.2 (DEAN AND SWAR [2009])) *Optimal GSA problem is NP-complete.*

We show that PL-SF, TL-SF, and AL-SF are all NP-complete by first presenting a polynomial time certificate and second reducing optimal GSA to these problems.

THEOREM 6.3 *Optimal PL-SF, TL-SF, and AL-SF are all NP-complete.*

Proof. Given a flow assignment, it is straightforward to check if a flow is feasible and the budget is exceeded or not in polynomial time.

For stability, we cannot check if each walk of a network is a blocking walk or not since there may be a lot of walks in a network. For a PL-network, we apply the reduction in section 3.2.1 to get an equivalent TL-network, while for a TL-network or AL-network, we apply the reduction in Corollary 5.1 to obtain an equivalent three-layer AL-network. In the final three-layer AL-network (G', s', t', c'), $G' = (V' \cup \{s', t'\}, E')$, we can just check if there are blocking walks either from s' to t', from s' to vertices in the third layer, from vertices in the first layer to vertices in the third layer, or from vertices in the first layer to t'. This takes only polynomial time.

Therefore, we can check if a flow is feasible and stable, and the cost is below budget in polynomial time. Optimal PL-SF, TL-SF, and AL-SF are all in NP.

For NP-hardness, as an AL-network is a special case of TL-network and PL-network, it suffices to reduce optimal GSA to optimal AL-SF. We apply the reduction in Corollary C.1 and additionally set $p(v_i v_{ij}) = h(ij)$ for each $ij \in C$ while the cost of other edges are set to zero. Clearly, the costs of the stable flow and the stable assignment are the same.

Therefore, we have a stable assignment below budget if and only if we have a stable flow below budget. ■

7 Conclusion

Our paper introduced a general version of stable flow problem. We show the existence of a stable flow and apolynomial time algorithm to find such a solution. This provides an initial step in the analysis of generalized flows with ordinal preferences. We beleive that our results has significant implications in the design and analysis of supply chain networks with agents' hetergenous preferences.

We further show that unlike the traditional stable flow proble, the minimum cost stable generalized flow is NP hard. This shows that the generalized stable flow is fundamentally more difficult than its original formulation.

Acknowledgment

This research is partly supported by National Science Foundation Grants AST- 1443965, CMMI 1728165.

References

A. Abdulkadiroglŭ and T. Sönmez. School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747, June 2003.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1 edition, Feb. 1993. ISBN 013617549X.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Elsevier, 2014.

E. M. Azevedo and J. W. Hatfield. Existence of equilibrium in large matching markets with complementarities. *Manuscript, Wharton School, Univ. Pennsylvania*, 2015.

M. Baïou and M. Balinski. The stable allocation (or ordinal transportation) problem. *Mathematics of Operations Research*, 27(3):485–503, 2002.

Y.-K. Che, J. Kim, and F. Kojima. Stable matching in large economies. *Manuscript, Department of Economics, Columbia University*, 2015.

Á. Cseh and J. Matuschke. New and simple algorithms for stable flow problems. *arXiv preprint arXiv:1309.3701*, 2013.

Á. Cseh, J. Matuschke, and M. Skutella. Stable flows over time. *Algorithms*, 6(3):532–545, 2013.

B. C. Dean and S. Munshi. Faster algorithms for stable allocation problems. *Algorithmica*, 58(1):59–81, 2010.

B. C. Dean and N. Swar. The generalized stable allocation problem. In *WALCOM*, pages 238–249. Springer, 2009.

T. Fleiner. On stable matchings and flows. In *WG*, pages 51–62. Springer, 2010.

T. Fleiner, Z. Jankó, A. Tamura, and A. Teytelboym. Trading networks with bilateral contracts. 2016.

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

J. W. Hatfield, S. D. Kominers, A. Nichifor, M. Ostrovsky, and A. Westkamp. Stability and competitive equilibrium in trading networks. *Journal of Political Economy*, 121(5): 966–1005, 2013.

J. W. Hatfield, S. D. Kominers, A. Nichifor, M. Ostrovsky, and A. Westkamp. Chain stability in trading networks. Technical report, Working paper, 2015.

P. E. Haxell and G. T. Wilfong. A fractional model of the border gateway protocol (bgp). In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 193–199. Society for Industrial and Applied Mathematics, 2008.

R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the optimal stable marriage. *Journal of the ACM (JACM)*, 34(3):532–543, 1987.

L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

S. Kintali. Scarf is PPAD-complete. *CoRR*, abs/0812.1601, 2008.

T. Nguyen and R. Vohra. Near-feasible stable matchings with couples. 2016.

N. Olver and L. A. Végh. A simpler and faster strongly polynomial algorithm for generalized flow maximization. *CoRR*, abs/1611.01778, 2016.

M. Ostrovsky. Stability in supply chain networks. *American Economic Review*, 98(3): 897–923, 2008a.

M. Ostrovsky. Stability in supply chain networks. *The American Economic Review*, pages 897–923, 2008b.

A. E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American Economic Review*, 89(4):748–780, 1999.

A. E. Roth, T. Sönmez, and M. U. Ünver. A kidney exchange clearinghouse in new england. *American Economic Review*, pages 376–380, 2005.

H. E. Scarf. An elementary proof of a theorem on the core of an n person game.

H. E. Scarf. The core of an n person game. *Econometrica: Journal of the Econometric Society*, pages 50–69, 1967.

Appendix

A Proof of Theorem 3.1

We prove the correctness of the reduction in section 3.2.1 by showing that f is a TL-SF if and only if x^* is a Scarf's solution.

Scarf $\rightarrow$ TL-SF:

Suppose x^* is a solution of Scarf, then x^* dominates every column of A. First we have to show flow feasibility of f.

Assume row v^{in} dominates column x_v^{in} and x_v^{out}, then row v^{in} prefers other nonzero entries (column x_e where $e = uv \in E$ and column x_v^{out}) to column x_v^{in}. Row v^{in} prefers column x_v^{in} the most, so all other entries of x^* must be zero and ${x^*}_v^{in} = b_{v^{in}}$. In order to obey constraint row v^{out}, g_v must follow equation 1, since otherwise $b_{v^{out}} < b_{v^{in}} = {x^*}_v^{in}$ implies that constraint row v^{out} will be violated. Therefore, ${x^*}_v^{in} = M(v)$, ${x^*}_v^{out} = 0$, and $f_{in}(v) = \sum_{e=uv \in E} {x^*}_e = 0$. From constraint row v^{out}:

$$\sum_{e=vu \in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} \leqslant M(v) + b_v \implies f_{out}(v) = \sum_{e=vu \in E} {x^*}_e \leqslant b_v$$

By similar argument, if row v^{out} dominates column x_v^{in} and x_v^{out}, g_v must follow equation 2, so ${x^*}_v^{in} = 0$, ${x^*}_v^{out} = M(v)$, and $f_{out}(v) = \sum_{e=vu\in E} {x^*}_e = 0$. From constraint row v^{in}:

$$a_v \sum_{e=uv\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} \leqslant M(v) + b_v \implies f_{in}(v) = \sum_{e=vu\in E} {x^*}_e \leqslant \frac{b_v}{a_v}$$

The remaining case is row v^{in} dominates column x_v^{out} and row v^{out} dominates column x_v^{in}. Both row v^{in} and row v^{out} bind, so we have either:

$$\begin{cases} a_v \sum_{e=uv\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} = M(v) \\ \sum_{e=vu\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} = M(v) + b_v \end{cases} \implies a_v \sum_{e=uv\in E} {x^*}_e + b_v = \sum_{e=vu\in E} {x^*}_e$$

if g_v follows equation 1 or

$$\begin{cases} a_v \sum_{e=uv\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} = M(v) + b_v \\ \sum_{e=vu\in E} {x^*}_e + {x^*}_v^{in} + {x^*}_v^{out} = M(v) \end{cases} \implies a_v \sum_{e=uv\in E} {x^*}_e - b_v = \sum_{e=vu\in E} {x^*}_e$$

if g_v follows equation 2.

In either case, $f_{out}(v) = g_v(f_{in}(v))$. Besides, if g_v follows equation 1, then $\sum_{e=vu\in E} {x^*}_e \geqslant b_v$. Similarly, if g_v follows equation 2, then $\sum_{e=uv\in E} {x^*}_e \geqslant \frac{b_v}{a_v}$.

For stability, given x^* dominates all columns of A, we show a proof by contradiction. Suppose f is not stable by assigning $f(e) = {x^*}_e$ for each $e \in E$. That is, there is a blocking walk $W = (v_1, ..., v_k)$ with vector $V_W = (r_1, ..., r_{k-1})$ that satisfies the three conditions in Definition 2.5. Let r_i be the first positive entry in V_W, consider the following cases:

1. If $i > 1$, then $r_{i-1} = 0$. By point 1.(c) in Definition 2.5, the only way to let $f_{out}(v_i)$ increase and $f_{in}(v_i)$ remain the same is g_{v_i} follows equation 1 and $f(v_{i-1}v_i) = 0$. Therefore, row ${v_i}^{in}$ dominates column $x_{v_i}^{in}$ and $x_{v_i}^{out}$, ${x^*}_{v_i}^{in} = M(v_i)$, and ${x^*}_{v_i}^{out} = 0$. This indicates row ${v_i}^{out}$ does not dominate column $x_{v_iv_{i+1}}$ because $x_{v_i}^{in} > 0$ and $x_{v_i}^{in}$ is the least preferred column. Row v_iv_{i+1} does not dominate column $x_{v_iv_{i+1}}$ because edge v_iv_{i+1} still has at least r_i remaining capacity. Hence, row ${v_{i+1}}^{in}$ must dominate column $x_{v_iv_{i+1}}$.

2. If $i = 1$, then row v_1v_2 does not dominate column $x_{v_1v_2}$ because edge v_1v_2 still has at least r_1 remaining capacity. By point 2 in Definition 2.5, if $v_1 = s$ then row ${v_2}^{in}$

must dominate column $x_{v_1v_2}$ because there are no corresponding rows in A for vertex s; if $v_1v_2 \succ_{v_1} v_1u$ for some other edge $v_1u \in E$ and $f(v_1u) > 0$, then row ${v_1}^{out}$ does not dominate column $x_{v_1v_2}$ because it prefers column x_{v_1u} more, so row ${v_2}^{in}$ must dominate column $x_{v_1v_2}$.

In either case, row ${v_{i+1}}^{in}$ must dominate column $x_{v_iv_{i+1}}$. Let r_j be the first zero entry in V_W where $r_{j-1} > 0$, if there is no such entry, then $j = k$. Consider the subvector $(r_i, r_{i+1}, ..., r_{j-1})$ of V_W where all entries are positive. Since row ${v_{i+1}}^{in}$ must dominate column $x_{v_iv_{i+1}}$, we have ${x^*}^{in}_{v_{i+1}} > 0$ and ${x^*}^{out}_{v_{i+1}} = 0$. Row $v_{i+1}v_{i+2}$ cannot dominate column $x_{v_{i+1}v_{i+2}}$ because edge $v_{i+1}v_{i+2}$ still has at least r_{i+1} remaining capacity. Row ${v_{i+1}}^{out}$ cannot dominate column $x_{v_{i+1}v_{i+2}}$ because column $x^{in}_{v_{i+1}}$ is the least preferred and ${x^*}^{in}_{v_{i+1}} > 0$. Therefore, row ${v_{i+2}}^{in}$ must dominate column $x_{v_{i+1}v_{i+2}}$. By repeating analogous arguement, we eventually have row ${v_j}^{in}$ must dominate column $x_{v_{j-1}v_j}$.

Consider the following cases of j:

1. If $r_{j-1} > 0$ and $r_j = 0$, then the only way to let $f_{in}(v_j)$ increase and $f_{out}(v_j)$ remain the same is g_{v_j} follows equation 2 and $f(v_jv_{j+1}) = 0$. Therefore, row ${v_j}^{out}$ dominates column $x^{in}_{v_j}$ and $x^{out}_{v_j}$, ${x^*}^{in}_{v_j} = 0$, and ${x^*}^{out}_{v_j} = M(v_j)$. This indicates row ${v_j}^{in}$ does not dominate column $x_{v_{j-1}v_j}$ because ${x^*}^{out}_{v_j} > 0$ and $x^{out}_{v_j}$ is the least preferred column.

2. If $j = k$, by point 3 in Definition 2.5, if $v_k = t$ then row ${v_k}^{in}$ is not defined because there are no corresponding rows in A for vertex s; if $v_{k-1}v_k \succ_{v_k} uv_k$ for some other edge $uv_k \in E$ and $f(uv_k) > 0$, then row ${v_k}^{in}$ does not dominate column $x_{v_{k-1}v_k}$ because it prefers column x_{uv_k} more.

From the aforementioned cases, either row ${v_j}^{in}$ is not defined or ${v_j}^{in}$ does not dominate column $x_{v_{j-1}v_j}$. We have a contradiction. Thus, by assigning $f(e) = {x^*}_e$ for each $e \in E$, feasibility and stability are guaranteed, we obtain a TL-SF.

TL-SF $\rightarrow$ Scarf:

Suppose f is stable, first set ${x^*}_e = f(e)$ for each $e \in E$. If ${x^*}_e = c(e)$ then row e dominates column x_e. The remaining is to assign ${x^*}^{in}_v$ and ${x^*}^{out}_v$ such that there are rows dominating column x^{in}_v, column x^{out}_v where $v \in V$, and column x_e where $e \in E$ is not saturated.

For $v_1 \in V$, if $v_1 = s$ or there exists v_2 and u both in V such that $v_1v_2 \succ_{v_1} v_1u$, $f(v_1u) > 0$, and v_1v_2 is unsaturated, then column $x_{v_1v_2}$ cannot be dominated by row v_1v_2 and row ${v_1}^{out}$. This forces row ${v_2}^{in}$ to dominate column $x_{v_1v_2}$. We should set ${x^*}^{in}_{v_2} = b_{{v_2}^{in}} - a_{v_2} \sum_{uv_2 \in E} {x^*}_{uv_2}$ and ${x^*}^{out}_{v_2} = 0$. Note that v_1v_2 satisfies point 2 of Definition 2.5.

For v_2 and v_3 both in V, if g_{v_2} follows equation 1, $f_{in}(v_2) = 0$ and $f(v_2v_3) < b_{v_2}$, then $b_{{v_2}^{in}} = M(v_2)$ and $b_{{v_2}^{out}} = M(v_2) + b_{v_2}$. The only way to dominate column $x^{in}_{v_2}$ is to set $x^{*in}_{v_2} = M(v_2)$ and $x^{*out}_{v_2} = 0$ so that row ${v_2}^{in}$ dominates column $x^{in}_{v_2}$.

We say vertices $v_2, v_2, ..., v_k$ are *inflow-dominated* if:

1. There is a walk $W = (v_1, v_2, ..., v_k)$ where $f(v_iv_{i+1}) < c(v_iv_{i+1})$ for $i = 1, ..., k-1$.
2. Either one of the cases hold:
   (a) v_1v_2 satisfies point 2 of Definition 2.5
   (b) $f_{in}(v_2) = 0$ and $f(v_2v_3) < c(v_2v_3)$.

We know that $x^{*in}_{v_2} = b_{{v_2}^{in}} - a_{v_2} \sum_{uv_2 \in E} {x^*}_{uv_2}$ ($\sum_{uv_2 \in E} {x^*}_{uv_2} = 0$ in case 2.(b)) and $x^{*out}_{v_2} = 0$. Since v_2v_3 is unsaturated, row ${v_2v_3}^{out}$ cannot dominate column $x_{v_2v_3}$. Row ${v_2}^{out}$ cannot dominate colulmn $x_{v_2v_3}$ because $x^{*in}_{v_2} > 0$ and column $x^{in}_{v_2}$ is the least preferred. This forces row ${v_3}^{in}$ to dominate column $x_{v_2v_3}$. We should set $x^{*in}_{v_3} = b_{{v_3}^{in}} - a_{v_3} \sum_{uv_3 \in E} {x^*}_{uv_3}$ and $x^{*out}_{v_3} = 0$. By repeating anologous reasoning, for any inflow-dominated vertex v_i where $i = 2, ..., k$, we should set $x^{*in}_{v_i} = b_{{v_i}^{in}} - a_{v_i} \sum_{uv_i \in E} {x^*}_{uv_i}$ and $x^{*out}_{v_i} = 0$.

Similarly, we say vertices $v_1, v_2, ..., v_{k-1}$ are *outflow-dominated* if:

1. There is a walk $W = (v_1, v_2, ..., v_k)$ where $f(v_iv_{i+1}) < c(v_iv_{i+1})$ for $i = 1, ..., k-1$.
2. Either one of the cases hold:
   (a) $v_{k-1}v_k$ satisfies point 3 of Definition 2.5
   (b) $f_{out}(v_{k-1}) = 0$ and $f(v_{k-2}v_{k-1}) < c(v_{k-2}v_{k-1})$.

By similar reasoning, for any outflow-dominated vertex v_i where $i = 1, ..., k-1$, we should set $x^{*in}_{v_i} = 0$ and $x^{*out}_{v_i} = b_{{v_i}^{out}} - \sum_{v_iu \in E} {x^*}_{v_iu}$.

No vertex in V can be both inflow-dominated and outflow-dominated, otherwise there is a blocking walk which contradicts to Definition 2.5.

For vertex $v \in V$ that is neither inflow-dominated nor outflow-dominated, it is guaranteed that column x_e where e has v as an endpoint is already dominated by some rows. We can arbitrarily assign non-negative values for x^{*in}_{v} and x^{*out}_{v} such that $x^{*in}_{v} + x^{*out}_{v} = \min\{b_{v^{in}} - a_v \sum_{uv \in E} {x^*}_{uv}, b_{v^{out}} - a_v \sum_{uv \in E} {x^*}_{uv}\}$. ■

B Proof of Corrolary 5.1

This is a direct result by combining section 3.2.1 and Theorem 5.1. We will use the same notaion as in section 3.2.1 for the TL-network instance.

Given the TL-network (G, s, t, c), $G = (V \cup \{s, t\}, E)$, and $\succ_v$ for $v \in V$, construct the three-layer AL-network with (G', s', t', c'), $G' = (V' \cup \{s', t'\}, E')$, and $\succ_{v'}$ for $v' \in V'$ as the following:

1. Create $s_{out} \in V'$ and $t_{in} \in V'$.
2. For each $v \in V$, create v_{in}, v_{out}, m_v^{in}, and m_v^{out} in V'.
3. Let $M(v) = \max(\sum_{uv \in E} c(uv), \sum_{vu \in E} c(vu), b_v) + 1$.
4. If g_v is in the form of equation 1, set $c'(s'v_{in}) = M(v)$ and $c'(s'v_{out}) = M(v) + b_v$.
5. If g_v is in the form of equation 2, set $c'(s'v_{in}) = M(v) + b_v$ and $c'(s'v_{out}) = M(v)$.
6. Set $c'(s's_{out}) = \sum_{su \in E} c(su) + 1$ and $c'(t_{in}t') = \sum_{ut \in E} c(ut) + 1$.
7. For each $e = uv \in E$, create $m_e \in E'$ and set $c'(u_{out}m_e) = c(uv)$ and $c'(m_e v_{in}) = a_v c(uv)$.
8. Set $c'(v_{out}m_v^{in}) = c'(v_{out}m_v^{out}) = c'(m_v^{in}v_{in}) = c'(m_v^{out}v_{in}) = M(v) + b_v$.
9. For each $v_{out} \in V'$, v_{out} prefers $v_{out}m_v^{out}$ the most and $v_{out}m_v^{in}$ the least, the preference of $v_{out}m_e$ for some $e = vu \in E$ is the same as $\succ_v$.
10. For each $v_{in} \in V'$, v_{in} prefers $m_v^{in}v_{in}$ the most and $m_v^{out}v_{in}$ the least, the preference of $m_e v_{in}$ for some $e = uv \in E$ is the same as $\succ_v$.
11. The preference of s_{out} and t_{in} is arbitrary.
12. Set $g_{s_{out}}(x) = x$ and $g_{t_{in}}(x) = x$. For each $v \in V$, set $g_{v_{in}}(x) = x$, $g_{v_{out}}(x) = x$, $g_{m_v^1}(x) = x$, and $g_{m_v^2}(x) = x$.
13. For each $e = uv \in E$, set $g_{m_e}(x) = a_v x$ if $v \neq t'$. If $v = t$, set $g_{m_e}(x) = x$.

Suppose f' is an AL-SF of the three-layer AL-network, to obtain the TL-SF f for the old TL-network, for each $e = uv \in E$, set $f(uv) = f'(u_{out}m_e)$. ∎

C Stable Flow and Generalized Stable Allocation

In this section, we introduce the generalized stable allocation (GSA) problem and show a reduction from TL-SF to GSA and a reduction from GSA to 3-layer-AL-SF.

C.1 The Reductions

The bi-directional reducibility between the stable allocation and the traditional stable flow problem was shown in Fleiner [2010]. We overview analogous statements, a reduction from GSA to 3-layer-AL-SF and a reduction from TL-SF to GSA.

COROLLARY C.1 *GSA can be reduced to 3-layer-AL-SF.*

Proof. For an instance of GSA including $G = (I \cup J, C)$, $\succ_i$ for $i \in I$, $\succ_j$ for $j \in J$, and the edge capacity function u, we construct the instance of 3-layer-AL-SF including (G, s, t, c), $G = (V \cup \{s, t\}, E)$, $\succ_v$ for $v \in V$, and g_v for $v \in V$:

1. Create s, t, $V = \{v_i | i \in I\} \cup \{v_j | j \in J\} \cup \{v_{ij} | ij \in C\}$.
2. Set $c(sv_i) = 1$ for every $i \in I$ and $c(v_j t) = 1$ for every $j \in J$.
3. Set $c(v_i v_{ij}) = u(ij)$, $c(v_{ij} j) = \mu(ij)u(ij)$, and $g_{v_{ij}}(x) = \mu(ij)x$ for every $ij \in C$.
4. $v_i v_{ij} \succ_{v_i} v_i v_{ij'}$ in the 3-layer-AL-SF instance if and only if $ij \succ_i ij'$ in the GSA instance.
5. $v_{ij} v_j \succ_{v_j} v_{i'j} v_j$ in the 3-layer-AL-SF instance if and only if $ij \succ_j i'j$ in the GSA instance.

By setting $x(ij) = f(v_i v_{ij})$, we have a solution for GSA. One can check that we have GSA if and only if we have 3-layer-AL-SF. If the assignment is not stable, then there exists a walk such that all three conditions in Definition 2.5 do not hold, as a result, we can find a blocking walk from s or some v_i to t or some v_j. If we can find a blocking walk in our 3-layer-AL-SF instance, then from the structure of the three-layer AL-network, it must be from s or some v_i to t or some v_j. In either case, all three conditions in Definition 6.3 do not hold. ■

COROLLARY C.2 *TL-SF can be reduced to GSA.*

Proof. As shown in Corollary 5.1, TL-SF can be reduced to 3-layer-AL-SF. Therefore, we can just follow the analogous approach described in Corollary 5.1 and use the same notations. Suppose we apply Corollary 5.1 on the orignal TL-network and obtain the three-layer AL-network. That is, we are given the three-layer AL-network (G', s', t', c') where $G' = (V' \cup \{s', t'\}, E')$. Construct the GSA instance with $G = (I \cup J, C)$ as the following:

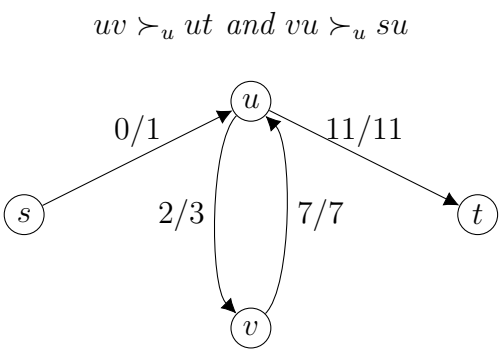
1. Let $I = \{i_s\} \cup \{i_v | v_{out} \in V'\}$ and $J = \{j_t\} \cup \{j_v | v_{in} \in V'\}$.
2. Let $C = \{i_w j_v | m_{wv} \in E'\} \cup \{i_w j_w | m_v^{out} \in V'\} \cup \{j_w i_w | m_v^{in} \in V'\}$.
3. Set $u(i_w j_v) = \frac{c'(w_{out} v_{in})}{c'(s' w_{out})}$ for each $m_{wv} \in E'$.
4. Set $u(i_w j_w) = u(j_w i_w) = \frac{c'(w_{out} m_w^{out})}{c'(s' w_{out})}$ for each $m_w^1 \in E'$.
5. Set $\mu(i_w j_v) = \frac{a_v c'(s' w_{out})}{c'(v_{in} t')}$ and $\mu(i_w j_w) = \mu(j_w i_w) = \frac{c'(s' w_{out})}{c'(w_{in} t')}$.
6. For each $w_{out} \in V'$, i_w prefers $i_w j_w$ the most and $j_w i_w$ the least, the preference of i_w for $i_w j_v$ where $m_{wv} \in E'$ is the same as $\succ_{w_{out}}$.
7. For each $v_{in} \in V'$, j_v prefers $j_v i_v$ the most and $i_v j_v$ the least, the preference of j_v for $i_w j_v$ where $m_{wv} \in E'$ is the same as $\succ_{v_{in}}$.
8. The preference of i_s and j_t is arbitrary.

Clearly, the flow is stable in the original TL-network if and only if the allocation is stable in the new GSA instance by analogous argument in Corollary 5.1. By setting $f(wv) = c(s' w_{out}) x(i_w j_v)$ for each $wv \in E$, we have a solution of TL-SF. ■

Remark. Although the bipartite graph G as an undirected graph, $i_v j_v$ and $j_v i_v$ will be regarded as different edges for convenience.

Basically, the GSA construction is actually equivalent to the three-layer AL-network in Corollary 5.1 with some proper capacity scaling. To give an illustrative explanation, it may be useful to look up Example 10.

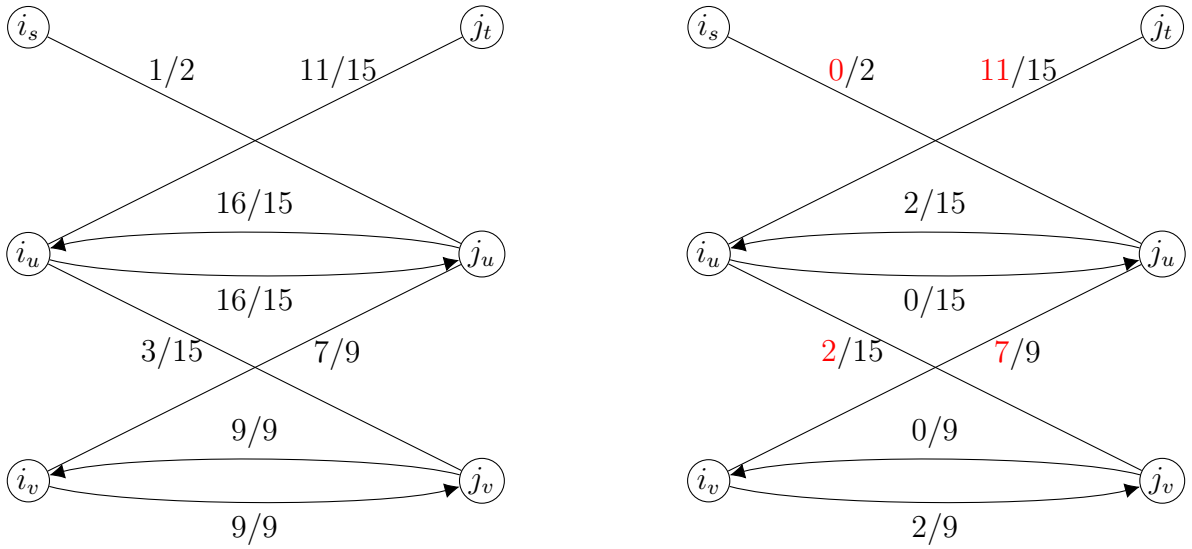
Example 10 *Consider the TL-SF instance in Example 9.*



$$g_u(x) = \begin{cases} 0 & \text{if } x < 0.5, \\ 2x - 1 & \text{otherwise.} \end{cases}$$

$$g_v(x) = \begin{cases} [0, 1] & \text{if } x = 0, \\ 3x + 1 & \text{otherwise.} \end{cases}$$

The left figure is the corresponding GSA instance obtained by Corollary C.2 with edge capacities in fraction. The right figure is the GSA solution.



The tables of multipliers and preferences are:

i	i_s	i_u	i_u	i_v	i_u	j_u	i_v	j_v
j	j_u	j_v	j_t	j_u	j_u	i_u	j_v	i_v
$\mu(ij)$	$\frac{4}{16}$	$\frac{45}{8}$	$\frac{15}{12}$	$\frac{18}{16}$	$\frac{15}{16}$	$\frac{15}{16}$	$\frac{9}{8}$	$\frac{9}{8}$

i/j	preference
i_u	$i_u j_u \succ_{i_u} i_u j_v \succ_{i_u} i_u j_t \succ_{i_u} j_u i_u$
j_u	$j_u i_u \succ_{j_u} i_v j_u \succ_{j_u} i_s j_u \succ_{j_u} i_u j_u$
i_v	$i_v j_v \succ_{i_v} i_v j_u \succ_{i_v} j_v i_v$
j_v	$j_v i_v \succ_{j_v} i_u j_v \succ_{j_v} i_v j_v$

We can see that not only equation 7, 8, and 9 but also all the conditions in Definition 6.3 are satisfied. The numerator of the stable allocation corresponds to the stable flow.