A Simultaneous Parametric Maximum-Flow Algorithm for Finding the Complete Chain of Solutions

Bin Zhang, Julie Ward, Qi Feng[1]
Intelligent Enterprise Technologies Laboratory
HP Laboratories Palo Alto
HPL-2004-189
October 25, 2004*

maximum flow, networks, parametric flow networks, graphs, optimization, selection, sequencing

A natural extension of the maximum flow problem is the parametric maximum flow problem, in which some of the arc capacities in the network are functions of a single parameter λ. Previous approaches to the problem compute the maximum flow for a given sequence of parameter values sequentially taking advantage of the solution at the previous parameter value to speed up the computation at the next. In this paper, we present a new Simultaneous Parametric Maximum Flow (SPMF) algorithm that finds the maximum flow and a minimum cut of an important class of parametric networks for *all* values of parameter λ simultaneously. Instead of working with the original parametric network, a new non-parametric network is derived from the original and the SPMF gives a particular state of the flows in the derived network, from which the nested minimum-cuts under all λ-values are derived in a single scan of the vertices in a sorted order. SPMF simultaneously discovers *all* breakpoints of λ where the maximum flow as a step-function of λ jumps. The maximum flows at these λ-values are calculated in $O(m)$ time from the minimum-cuts; m is the number of arcs. Generalization beyond bipartite networks is also shown.

* Internal Accession Date Only
[1]Department of OR, University of Texas at Dallas, Richardson, TX, USA
Approved for External Publication

A Simultaneous Parametric Maximum-Flow Algorithm for Finding the Complete Chain of Solutions

Bin Zhang and Julie Ward
Hewlett-Packard Research Laboratories, Palo-Alto, CA 94304
Qi Feng
Department of OR, University of Texas at Dallas, Richardson, TX, USA, 75083

Abstract. A Simultaneous Parametric Maximum-flow (SPM) algorithm finds the maximum-flow and a min-cut of a bipartite parametric maximum-flow network simultaneously for all values of parameter λ. Instead of working with the original parametric network, a new network is derived from the original and the SPM gives a particular state of the flows in the derived network, from which the nested min-cuts under all lambda values is derived in a single scan of the nodes in a sorted order. *All* breakpoints of λ where the maximum-flow as a step-function of λ jumps is discovered by the SPM. The maximum flows at these λ-values are calculated in *O(m)* time from the min-cuts; *m* is the number of arcs. Generalization beyond bipartite networks is also shown.

Keywords. maximum-flow, networks, parametric networks, algorithms, graphs, optimization, linear programming, scheduling, selection, sequencing

AMS(MOS) subject classifications. 05C20, 05C85

1. Introduction

A network is a finite directed graph *G=(N,E)* with capacity constraints *C* and flows *F* on the arcs – summarized in a notation $\Omega = \{N, E, C, F\}$. One important class of network flow problems is to find the maximum flow under the capacity constraints, solved by two well known algorithms – the augmenting-path algorithm by Ford & Fulkerson (1956) and Elias et al (1956), and the Preflow-Push/Relabel algorithm by Goldberg & Tarjan (1986). The later is much faster.

The well-known maximum-flow min-cut theorem of Ford and Fulkerson states that the max-flow is equal to the sum of the capacities of the arcs at the boundary of a min-cut (see Section 1.3).

Gallo at. el. (1989) generalized the preflow-push algorithm to a class of parametric bipartite networks which has nested min-cuts under different parameter values. Taking advantage of the nested min-cut structure, they used the flows (truncated if necessary) under one parameter value as the initialization to the problem at the next parameter value. They proved that the worst-case complexity for a *given* sequence of parameter values is only larger by a constant factor than the worst-case complexity of the non-parametric preflow-push algorithm. A number of applications of the parametric max-flow network were presented in their paper.

1.1 Contributions of This Paper

The simultaneous parametric maximum-flow algorithm presented in this paper solves the maximum-flow problem of a parametric network at *all* parameter values simultaneously. It returns the *complete* sequence of maximum-flows and the nested min-cuts at all parameter values where the max-flow as a step-function jumps. This complete sequence of parameter values is discovered by the algorithm. We are not aware any algorithm published before that can do this. The paper by Gallo at. el. (1989) takes a sequence of parameter values as input.

The new algorithm is different from all previous published network flow algorithms. The new algorithm does not work directly with the original parametric network. Instead, it works on a derived network which no longer depends on the parameter (See Section 2). The new algorithm does not try to find the maximum-flow of the derived network instead it finds a particular state of the flows from which the maximum-flow and a min-cut of the original network under any parameter value can be derived. The new algorithm does not use preflows; there is never any excess at any node other than the source and the flow balance equations hold at all nodes at all times (except at the source and target). No labels of the nodes are used. The new algorithm is not the augmenting-path algorithm because it does not try to find the maximum-flow of the derived network; it may not push flow up to the full residue capacity of an augmenting-path; and it does not push flow along all augmenting paths.

1.2 A Few Basic Concepts and Terminologies

A number of basic notations and definitions are reviewed in this section for readability of the paper but may not be completely spelled out (see Ahuja, Magnanti & Orlin's book for more details).

Network and Flow: A *network* $\Omega = \{N, E, C, F\}$ consists of an underlying finite directed graph $G(N, E)$, a set $C = \{c_{i,j} \geq 0 \mid i \neq j; i, j = 1, ..., n\}$ of capacities of the arcs in E and a set of flows $F = \{f_{i,j} \mid c_{i,j} \geq f_{i,j} \geq 0; i \neq j; i, j = 1, ..., n\}$ through the arcs. Often a missing arc between an ordered pair of nodes is equalized with a zero capacity for mathematical analysis (but not for algorithmic discussions and implementation); with this understanding, E is not necessary in the network notation -- $\Omega = \{N, C, F\}$. Sometimes it is more convenient to sub-index capacities and flows by nodes directly, like $c_{w,w'}$ or $f_{w,w'}$.

Flows are balanced at each node other than *s* and *t*: $\sum_{w' \neq w} f_{w,w'} = 0$ for $\forall w \in N \setminus \{s, t\}$.

$N = \{s, t\} \bigcup \{w_i\}_{i=1}^{n}$ contains two special nodes -- the *source s* with only outgoing arcs and the *target t* with only incoming arcs.

No multiarcs (arcs with the same head and the same tail) and loops are in E.

Residue Capacity: Working with increments of flows, it is more convenient to look at the remaining capacity of an arc, $\hat{c}_{i,j} = c_{i,j} - f_{i,j}$, called *residue capacity*. When an arc (w_i, w_j) carries a positive flow, reducing that flow is the same as adding a reversed flow to it; the maximum amount of reversed flow allowed is equal to the current forward flow,

which is called the residue capacity of the reversed arc: $\hat{c}_{j,i} = f_{i,j}$. $\hat{C} = \{\hat{c}_{i,j} \mid i, j = 1,...,n\}$, the set of residue capacities.

Residue Network: A residue network is the network $\hat{\Omega} = \{N, \hat{C}, F\}$, with $-f_{j,i} = f_{i,j}$, a somewhat confusing notation, which means that adding a positive flow in the reversed direction of an arc is to subtract that amount of flow from the existing forward flow. A residue arc is an ordered pair of nodes (w_i, w_j) with positive residue capacity $\hat{c}_{w_i,w_j} > 0$. A path in the residue network is called a residue path. The capacity along a residue path is the minimum residue capacity of the residue arcs in the path.

Augmenting-Path is a path from the source to the target with positive residue capacity in the residue network. Augmenting path maximum flow algorithm is the most intuitive maximum flow algorithm; it keeps finding augmenting paths to increase the total flow from *s* to *t* until no more augmenting paths left.

***s-t*-Cut, Min-Cut and Maximum flow:** A *s-t*-cut is a partition of N into two disjoint subsets, an s-partition containing *s* and an *t*-partition containing *t*. The capacity of an *s-t*-cut is the sum of the capacities of the arcs from a node in *s*-partition to a node in *t*-partition. A min-cut is an *s-t*-cut with minimum capacity among all *s-t*-cuts. The maximum flow of a network is the maximum amount of flow possible from the source to the target. The maximum-flow min-cut theorem of Ford and Fulkerson states that the max-flow is equal to the capacity of a min-cut.

Preflow-Push-Relabel Algorithm is a faster algorithm for finding maximum-flow in a network than the augmenting path algorithm. Instead of pushing flow along a complete augmenting path, it allows flows to be pushed along one arc at a time up to the capacity of the arc. Excess of flows at a node is allowed; flow balance equations become "total incoming flows ≥ total outgoing flows". The excess flows are pushed in the direction controlled by a height-label assigned to each node. Flows can be pushed only from nodes at higher heights to nodes at lower heights. When a node with excess has no lower-height-nodes to push to, its height is raised by a minimum increment to turn the situation around. For details, see the original paper by Goldberg & Tarjan (1986).

The rest of the paper is organized in 3 sections. In Section 2, the generic version of the simultaneous parametric max-flow (SPM) algorithm on bipartite networks is introduced. Its convergence, correctness and completeness are proven. In Section 3, generalization beyond bipartite networks is shown. Section 4 concludes the paper with suggestions of next steps.

2. A Simultaneous Parametric Maximum Flow Algorithm that Gives the Complete Nested Chain of Min-Cuts

We first work with bipartite networks because of their simpler structure.

A parametric network model $\Omega = \{N, E, C_\lambda, F\}$ is a network with some of its capacities depending on a parameter. Two disjoint subsets of nodes are sometimes named separately as $U = \{u_i \mid i = 1,...,n_1\}$ and $V = \{v_l \mid l = 1,...,n_2\}$. $N = \{s,t\} \bigcup W$ and $W = U \bigcup V$.

An arc exists from s to every node in U and from every node in V to t. Arcs between U and V may go in either direction. $E = \{(s,u_i)\}_{i=1}^{n_1} \bigcup \{(v_j,t)\}_{j=1}^{n_2} \bigcup \{(w,w') \mid w \in U,\ w' \in V;\ \text{or } w' \in U, w \in V\}$.

Capacities $c_{i,j} \in [0,+\infty]$, $i \neq j, i, j = 1,...,n$. Capacity from s to any $u \in U$ is a continuous monotone increasing function of λ, and from any $v \in V$ to t a continuous monotone decreasing function of λ, where $\lambda \in [a,b]$, if $a = -\infty$ and/or $b = +\infty$, the value of the capacity at infinity is defined by the limit, which exists and can be infinity. Under these assumptions, the inverse functions of the capacities exist, are continuous and monotone.

Based on the maximum-flow/min-cut theorem, the maximum-flow under a fixed λ is equal to the capacity of a min-cut (N_s, N_t). **Fig. 1** shows an aggregated view of a max-flow/min-cut solution.

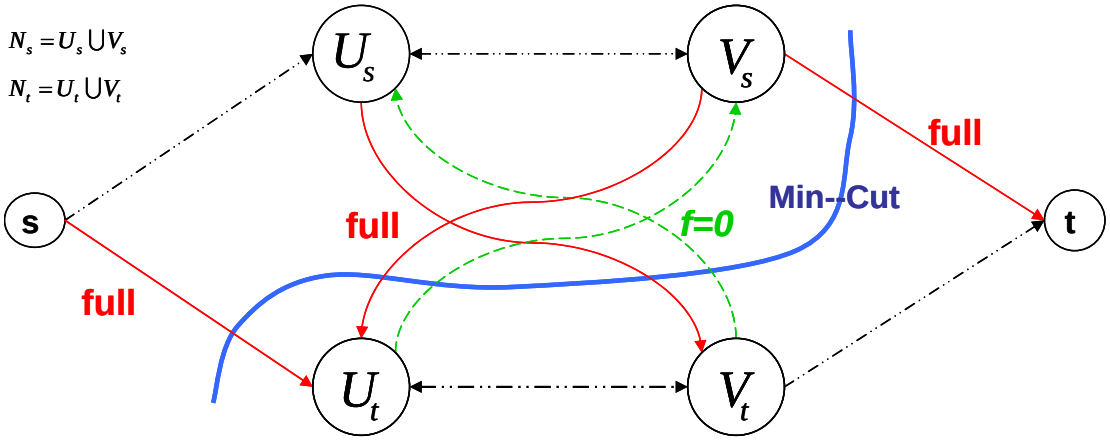

Figure 1. The diagram shows an aggregated view of a min-cut. Each big circle is the aggregate of a set of nodes. The directed arcs are the aggregates of all directed arcs from one entity to another. This picture shows that there is no augmenting path from *s* to *t*.

Instead of working with the original parametric network, a new network $\bar{\Omega} = \{N, E, \bar{C}, \bar{F}\}$ is derived from the original. The SPM algorithm, developed in Section 2.2, gives a particular state of the flows in $\bar{\Omega}$, which allows a min-cut of the original network under *any* λ be derived in a single scan of the nodes. If the nodes are sorted by a particular key, shown later, the complete chain of nested min-cuts under *all* λ can be derived from a single scan of the nodes. Sorting takes *O(n*log*(n))* time. *All* breakpoints of λ where the maximum-flow as a step-function jumps are discovered by the SPM. The maximum flows at these λ-values are calculated in *O(m)* time from the min-cuts; *m* is the number of arcs.

2.1 Removing the capacity constraints on the arcs from the source or to the target

The new network $\bar{\Omega} = \{N, E, \bar{C}, \bar{F}\}$ is derived from the original network Ω by keeping the same underlying graph and the capacity constraints on any arc between two nodes in *W*, but replacing all the capacity constraints by $+\infty$ on the arcs from the source *s* or to the target *t* (See **Fig. 2**). In the derived network, flows through these arcs are allowed to take any value in $[0, +\infty)$. What value they will take is determined by the SPM algorithm.

$\bar{\Omega}$ does not depend on λ after removing all λ-dependent capacities in Ω. A set of lambda values are calculated from the flows in $\bar{\Omega}$ using the inverse capacity functions of the original network:

$$\lambda_{s,i} = c_{s,i}^{-1}(\bar{f}_{s,i}), \quad i = 1, ..., n_1 \quad \text{and} \quad \lambda_{l,t} = c_{l,t}^{-1}(\bar{f}_{l,t}), \quad l = 1, ..., n_2. \tag{2.1}$$

The inverse functions exist because of monotocity of $c_{s,i}()$ and $c_{l,t}()$. The use of these lambda values becomes clear in the next sub-section.

2.2 The rules for altering flows in the derived network $\bar{\Omega} = \{N, E, \bar{C}, \bar{F}\}$

We call a path of this type $s \to u_i \to v_l \to t$ *a simple* residue path if one of the two cases is true: case a) $\lambda_{s,i} < \lambda_{l,t}$ and $\hat{c}_{u_i \to v_l} > 0$ or case b) $\lambda_{s,i} > \lambda_{l,t}$ and $\hat{c}_{v_l, u_i} > 0$. $\hat{c}_{u_i, v_l}$ and $\hat{c}_{v_l, u_i}$ are the residue capacities of the directed arc in the residue network.

The generic version of the Simultaneous Parametric Maximum-flow (SPM) algorithm:

Initialization: Any feasible flow of $\bar{\Omega}$ will do. Starting with completely zero flows is allowed -- all $\lambda_{s,i}, i = 1, ..., n_1$ are at their minimum, all $\lambda_{l,t}, l = 1, ..., n_2$ are at their maximum.

Main Algorithm: Repeat the following operation until no more simple residue path left.

One operation: for (any) a simple residue path $s \to u_i \to v_l \to t$, do:

a) if $\lambda_{s,i} < \lambda_{l,t}$, pushing $\delta flow = \min\{\hat{c}_{u_i, v_l}, \delta f_{i,l}\}$ from *s* to *t* through the path;
b) if $\lambda_{s,i} > \lambda_{l,t}$, pushing $\delta flow = \min\{\hat{c}_{v_l, u_i}, -\delta f_{i,l}\}$ from *t* to *s* through the reversed residue path ($\delta f_{i,l} < 0$ in this case).

$\delta f_{i,l}$ is the amount of flow that makes the two new lambda values equal after the push if the push is not constrained by the residue capacity:

$$c_{s,i}^{-1}(\bar{f}_{s,i} + \delta f_{i,l}) = c_{l,t}^{-1}(\bar{f}_{l,t} + \delta f_{i,l}). \tag{2.2}$$

After pushing the flow, update the two lambda values according to the new flows:

$$\lambda_{s,i}^{'} = c_{s,i}^{-1}(\bar{f}_{s,i} + \delta flow) \text{ and } \lambda_{l,t}^{'} = c_{l,t}^{-1}(\bar{f}_{l,t} + \delta flow). \tag{2.3}$$

Equation (2.2) always has a solution $\delta f_{i,l}$ because as $\delta f_{i,l}$ starts from zero and moving in the direction that makes $c_{s,i}^{-1}(\overline{f}_{s,i} + \delta f_{i,l}) \in [a,b]$ and $c_{l,t}^{-1}(\overline{f}_{l,t} + \delta f_{i,l}) \in [a,b]$ closer, the smaller number approaches *b* and the larger number approaches *a* (<*b*). There must exist a $\delta f_{i,l}$ satisfying (2.2).

The flow is always pushed from the end with a smaller lambda value to the end with a larger lambda value and one of the following inequality holds with the new lambda values $\lambda'_{s,i}$ and $\lambda'_{l,t}$:

Case a) if $\lambda_{s,i} < \lambda_{l,t}$, $\lambda_{s,i} < \lambda'_{s,i} \le \lambda'_{l,t} < \lambda_{l,t}$ or

Case b) if $\lambda_{s,i} > \lambda_{l,t}$, $\lambda_{s,i} > \lambda'_{s,i} \ge \lambda'_{l,t} > \lambda_{l,t}$. (2.4)

After each operation, either the residue capacity from the end with lower lambda value to the end with higher lambda value becomes zero or the two new lambda values become the same. A single operation will never reverse the order of the two lambda values even though the order could be reversed in the future by other operations. (See Fig. 2 & 3.)

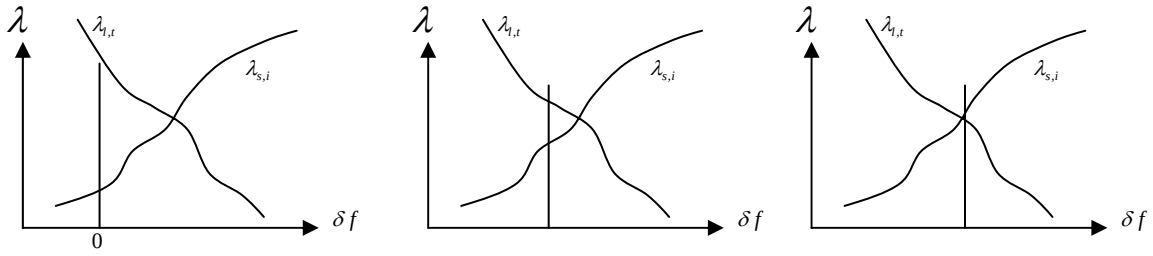

Figure 2. Case a) $\lambda_{s,i} < \lambda_{l,t}$: Left – before push $\delta f = 0$; middle – after push the residue capacity becomes zero; right – after push, $\lambda_{s,i} = \lambda_{l,t}$ but residue capacity of the path may not be zero.

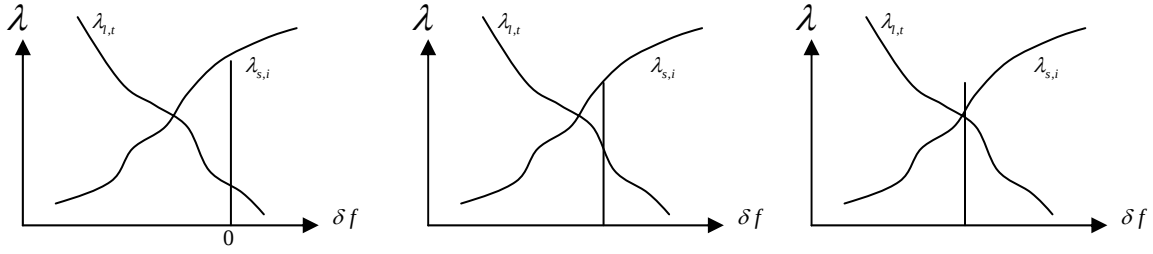

Figure 3. Case b) $\lambda_{s,i} > \lambda_{l,t}$: Left – before push $\delta f = 0$; middle – after push the residue capacity becomes zero; right – after push, $\lambda_{s,i} = \lambda_{l,t}$ but the residue capacity may not be zero.

Propert A: When SPM stops (a proof is given later), there is no more augmenting path (or even an arc with positive residue capacity) going from a node with lower lambda value to a node with higher lambda value.

This property allows us to easily derive min-cuts of the original network under all values of λ.

2.3 Deriving min-cuts of the original parametric maximum-flow problem

Using the derived network $\overline{\Omega} = \{N, E, \overline{C}, \overline{F}\}$ with the flows given by the SPM algorithm, a min-cut of $\Omega = \{N, E, C_\lambda, F\}$ under any λ value is derived in a single linear scan of the nodes. Partition nodes N into $(N_{s,\lambda}, N_{t,\lambda})$ as follows

$$N_{s,\lambda} = \{s\} \bigcup \{u_i \mid \lambda_{s,i} = c_{s,i}^{-1}(\overline{f}_{s,i}) < \lambda\} \bigcup \{v_l \mid \lambda_{l,t} = c_{l,t}^{-1}(\overline{f}_{l,t}) < \lambda\} \text{ and } N_{t,\lambda} = N \setminus N_{s,\lambda}. \quad (2.5)$$

From **Property A** in the previous section, there is no augmenting path from $N_{s,\lambda}$ to $N_{t,\lambda}$ in the derived network.

Putting the capacity constraints $\{c_{s,i}(\lambda) \mid i = 1,...,n_1\}$ and $\{c_{l,t}(\lambda) \mid l = 1,...,n_2\}$ back to $\overline{\Omega} = \{N, E, \overline{C}, \overline{F}\}$ by truncating the flows where the constraint is violated, we recover the original network at parameter value λ with the maximum flow. For $u_i \in U_{t,\lambda}$ with $\lambda \le \lambda_{s,i}$, $c_{s,i}(\lambda) \le c_{s,i}(\lambda_{s,i}) = \overline{f}_{s,i}$ because $c_{s,i}(\lambda)$ is non-decreasing; these flows $\overline{f}_{s,i}$ are truncated. Without changing the flows from $N_{s,\lambda} \setminus \{s\}$ to $N_{t,\lambda}$, the truncated flows can be rebalanced at $U_{t,\lambda}$ by reducing the flows from $U_{t,\lambda}$ to $V_{t,\lambda}$ and rebalanced again at $V_{t,\lambda}$ by reducing flows from $V_{t,\lambda}$ to *t* (trivial to prove using flow balance equations at each node. See **Fig. 4** and **Fig. 6**).

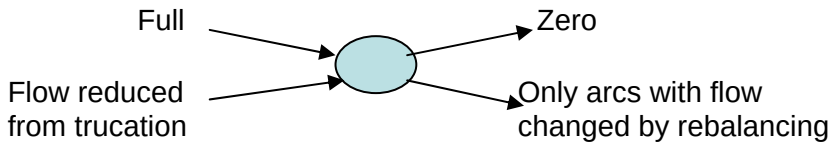

Figure 4. Flow rebalancing caused by truncation of flows from the source to *U*.

Similarly, for $v_l \in V_{s,\lambda}$ with $\lambda > \lambda_{l,t}$, $c_{l,t}(\lambda) \le c_{l,t}(\lambda_{l,t}) = \overline{f}_{l,t}$ because $c_{l,t}(\lambda)$ is non-increasing; these flows $\overline{f}_{l,t}$ are truncated. Without changing the flows from $N_{s,\lambda}$ to $N_{t,\lambda} \setminus \{t\}$, the truncated flows can be rebalanced at $V_{s,\lambda}$ by reducing the flows from $U_{s,\lambda}$ to $V_{s,\lambda}$ and rebalanced again at $U_{s,\lambda}$ by reducing flows from *s* to $U_{s,\lambda}$ (See **Fig. 5** and **Fig. 6**).

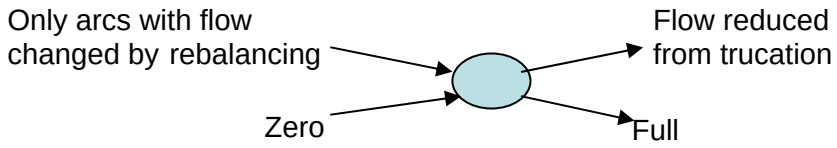

Figure 5. Flow rebalancing caused by truncation of flows from *V* to the target.

After the truncations and rebalancing, a maximum flow in the original network with min-cut $(N_{s,\lambda}, N_{t,\lambda})$ is achieved (See **Fig. 6** and compare it with **Fig. 1**).

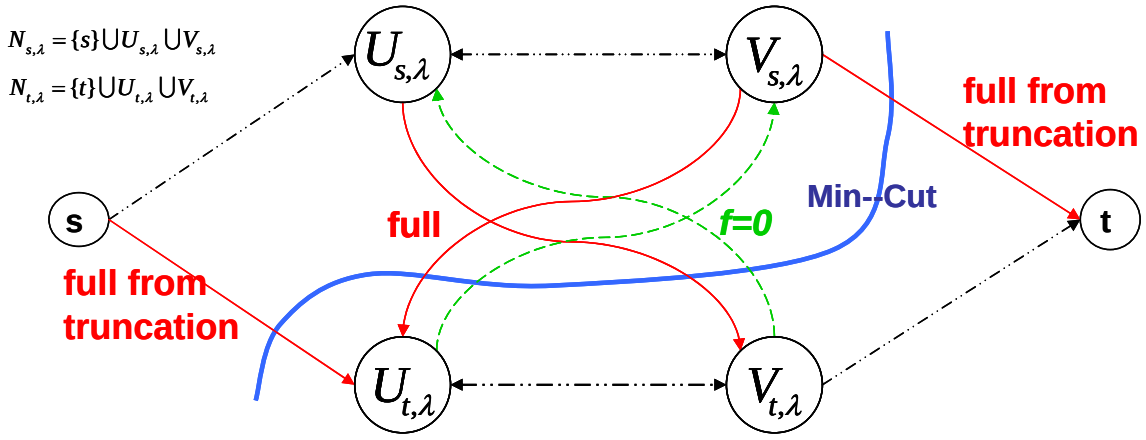

Figure 6. Putting the capacity constraints $c_{s,i}(\lambda)$ and $c_{l,t}(\lambda)$ back by truncating the flows that violate the constraints, we get $\Omega = \{N, E, C_\lambda, F\}$ back with a max-flow from the truncated flows and a min-cut $(N_{s,\lambda}, N_{t,\lambda})$. The black double-dotted arcs are the only (aggregated) arcs with flows possibly changed by rebalancing. The red solid arcs have flow up to the maximum capacity. The green dash arcs have zero flow.

For the purpose of finding a min-cut and the maximum-flow under λ, there is no need to carry out the truncations and rebalancing, which are only for proving the correctness of the SPM algorithm. A single scan of the nodes in N in the *increasing* order of their associated λ values gives all min-cuts as monotone increasing step set-function $N_{s,\lambda}$ of λ and the λ values at which the sizes of the min-cut partitions change. Sorting the nodes by their associated lambda value takes $O(n\log(n))$ time.

The min-cuts are nested, which has been shown before in Gallo at. el. (1989), is also a by-product of this proof.

All maximum flow values of the original network as a step function of λ can be calculated in $O(m)$ time, where m is the number of arcs, if the arcs are also scanned along with the scanning of the nodes to get the capacities of the min-cuts. Since the min-cuts are nested, each arc is scanned twice – once when one of its ends changes membership (from *t*-partition to *s*-partition) and its capacity from $N_{s,\lambda}$ to $N_{t,\lambda}$ is added to the maximum-flow; and the second time when the other end changes its membership and its (previous) capacity from $N_{s,\lambda}$ to $N_{t,\lambda}$ is subtracted from the maximum-flow. The complete procedure for getting all min-cuts and the naximum-flows is summarized in the *Post_Processing Algorithm:*

(There is no need to distinguish lambda values associated with a node in U or a node in V. We use the simpler notation $\{\lambda_i \mid i = 1,...,n\}$ for the set of lambda values.)

Input: The derived network $\bar{\Omega} = \{N, E, \bar{C}, \bar{F}\}$ with $\bar{F}$ given by the SPM.

Step 1: Sort the nodes in $N \setminus \{s,t\}$ by their associated lambda values λ_i, *i=1,…,n*. For convenience, assume that $\lambda_1 \le ... \le \lambda_n$.

Step 2: Set *i=1*, $N_{s,\lambda_i} = \{s\}$, $N_{t,\lambda_i} = N \setminus N_{s,\lambda_i}$, $flow_{\max} = 0$.

Step 3: $\Delta_i = \{w \in W \mid \lambda_w = \lambda_i\}$.

For $\forall w \in \Delta_i$ and $\forall w' \in \{w' \mid w' \in N_{s,\lambda_i} \,\&\, c_{w',w} > 0\}$, $flow_{\max} = flow_{\max} - c_{w',w}$;

For $\forall w \in \Delta_i$ and $\forall w' \in \{w' \mid w' \in N_{t,\lambda_i} \setminus \Delta_i \,\&\, c_{w,w'} > 0\}$, $flow_{\max} = flow_{\max} + c_{w,w'}$;

$N_{s,\lambda_{i+1}} = N_{s,\lambda_i} \bigcup \Delta_i$, $N_{t,\lambda_{i+1}} = N \setminus N_{s,\lambda_{i+1}}$;

output: $flow_{\max}$ and Δ_i (or $(N_{s,\lambda_{i+1}}, N_{t,\lambda_{i+1}})$), the maximum flow and a min-cut for $\lambda \in (\lambda_i, \lambda_{i+1})$.

If $i = i+1 \le n$, go to the beginning of Step 3. else DONE.

2.4 The completeness of the solution

Completeness means that for every λ value, the maximum flow and a min-cut of the original network is found. It follows the proof in the previous section immediately.

2.5 The convergence of the SPM algorithm

The "Main Algorithm" given above formula (2.2) tells what to do mathematically. In any implementation of the generic algorithm, a particular order of scanning the simple residue paths has to be specified which in general will have an impact on the runtime cost of the algorithm. We want the proof of convergence to be independent of such implementation details.

To complete the proof, we need a number of definitions and results from calculus. They (Lemma 1 to Lemma 5) are included in Appendix A.

Assumption A: Let (λ_i, λ_j) and (λ_i', λ_j') be an old and a new pair of lambdas before and after an operation on a simple residue path. For any sequence of such pairs, a finite amount of total upward movements $\sum_{i=1}^{+\infty} |\lambda_i - \lambda_i'| < +\infty$ is always match with a finite amount of total downward movements $\sum_{j=1}^{+\infty} |\lambda_j - \lambda_j'| < +\infty$.

Lemma: A sufficient condition for Assumption A to hold is $\Delta \ge \left|\frac{dc_{s,i}()}{d\lambda}\right| \ge \delta > 0, i = 1,...,n_1,$ and $\Delta \ge \left|\frac{dc_{l,t}()}{d\lambda}\right| \ge \delta > 0, l = 1,...,n_2,$ for some constants Δ, δ.

Proof: Assume that it is case a) $\lambda_{s,i} < \lambda_{l,t}$; the order case can be proved similarly. From (2.2) and $\delta flow = \min\{\hat{c}_{i,l}, \pm\delta_{i,l}\}$,

$$\lambda_{s,i}^{'} = c_{s,i}^{-1}(f_{s,i} + \delta flow) \text{ and } \lambda_{l,t}^{'} = c_{l,t}^{-1}(f_{l,t} + \delta flow) \tag{2.6}$$

Invert the second one in (2.6) and use the fact $f_{l,t} = c_{l,t}(\lambda_{l,t})$,

$$\delta flow = c_{l,t}(\lambda_{l,t}^{'}) - c_{l,t}(\lambda_{l,t}) \tag{2.7}$$

Combining the first one in (2.5) and (2.7),

$$\begin{aligned} &|\lambda_{s,i}^{'} - \lambda_{s,i}| = |c_{s,i}^{-1}(f_{s,i} + c_{l,t}(\lambda_{l,t}^{'}) - c_{l,t}(\lambda_{l,t})) - c_{s,i}^{-1}(f_{s,i})| \\ &\le \frac{1}{\delta}|c_{l,t}(\lambda_{l,t}^{'}) - c_{l,t}(\lambda_{l,t})| \le \frac{\Delta}{\delta}|\lambda_{l,t}^{'} - \lambda_{l,t}| \end{aligned} \tag{2.8}$$

Similarly, we have

$$|\lambda_{l,t}^{'} - \lambda_{l,t}| = |c_{l,t}^{-1}(f_{l,t} + c_{s,i}(\lambda_{s,i}^{'}) - c_{s,i}(\lambda_{s,i})) - c_{l,t}^{-1}(f_{l,t})| \le \frac{\Delta}{\delta}|\lambda_{s,i}^{'} - \lambda_{s,i}| \tag{2.9}$$

•

Convergence Theorem: Let $O_1, O_2, ..., O_k, ...$ be a sequence of operations happened in a run of the generic algorithm. Under Assumption A, the set of lambda values after the *k*th operation $\Lambda^{(k)} = \{\lambda_i^{(k)} \mid i = 1, ..., n\}$ converges as a sorted sequence. The convergence is defined as for $\forall \varepsilon > 0$, $\exists K > 0$, for $\forall k_1, k_2 > K$ and *j=1,…,*n, the difference between the *j*th number in the *sorted* sequence of $\Lambda^{(k_1)}$ and the *j*th number in the *sorted* sequence of $\Lambda^{(k_2)}$ is smaller than ε**.**

An even stronger result holds: the convergence is *absolute* in the sense that the sum of the absolute values of the changes at the *j*th position of the sorted sequence is finite for any *j=1,…,n***.**

Proof: Let $\lambda_{\min,0}^{(k)} = \min \Lambda^{(k)}$. $\lambda_{\min,0}^{(1)} \le \lambda_{\min,0}^{(2)} \le ... \le \lambda_{\min,0}^{(k)} \le ...$ is a non-decreasing sequence and bounded above from (2.4). It is absolutely convergent (see Lemma 1 in Appendix A). For any $\varepsilon > 0$ there exists a $K_1 > 0$ such that $|\lambda_{\min,0}^{(k)} - \lim_{k\to\infty} \lambda_{\min,0}^{(k)}| = \sum_{k'=k+1}^{+\infty} |\lambda_{\min,0}^{(k+1)} - \lambda_{\min,0}^{(k)}| < \varepsilon$ for all $k > K_1$. Let $\Lambda_1^{(k)} = \Lambda^{(k)} \setminus \{\lambda_{\min,0}^{(k)}\}$, if there are more than one lambda that equal to $\lambda_{\min,0}^{(k)}$, pick any one. The size of $\Lambda_1^{(k)}$ is *n-1* for any *k*. Let $\lambda_{\min,1}^{(k)} = \min \Lambda_1^{(k)}$. $\{\lambda_{\min,1}^{(k)}\}_{k=1}^{+\infty}$ is a semi-increasing sequence because all downward movements have to be matched with $\lambda_{\min,0}^{(k)}$'s upward movement, the sum of which is bounded. Following Lemma 4, $\{\lambda_{\min,1}^{(k)}\}_{k=1}^{+\infty}$ is absolutely convergent. There exists a $K_2 > K_1$ such that $\sum_{k=K_2}^{+\infty} |\lambda_{\min,1}^{(k)} - \lambda_{\min,1}^{(k+1)}| < \varepsilon$. The same

procedure can be repeated to $\Lambda_2^{(k)} = \Lambda_1^{(k)} \setminus \{\lambda_{\min,1}^{(k)}\}$ (if multiple lambda equal to the min, pick any one); and the semi-increasing sequence $\lambda_{\min,2}^{(k)} = \min \Lambda_2^{(k)}$, every downward step of which is matched with a upward step of one of the earlier min-lambdas $\lambda_{\min,0}^{(k)}, \lambda_{\min,1}^{(k)}$, which are absolutely convergent. Therefore $\{\lambda_{\min,2}^{(k)}\}_{k=1}^{+\infty}$ is absolutely convergent by Lemma 4. Repeating this process *n* times, we have all *n* sequences $\{\lambda_{\min,i}^{(k)}\}_{k=1}^{+\infty}$, *i=0…,n-1*, are absolutely convergent. There is only finite number of such sequences, a common *K>0* can be chosen. •

The proof can be done using the $\lambda_{\max,i}^{(k)}$ defined similarly.

For any computer implementation, the lambda values are rational numbers, they will stop changing when $\varepsilon > 0$ is smaller than the machine precision.

3. Generalization to General Networks

The SPM algorithm can be generalized to networks with the type of graph illustrated in **Fig. 7**. $N = \{s,t\} \bigcup U \bigcup W \bigcup V$. There are no arcs between nodes within *U* or nodes within *V*.

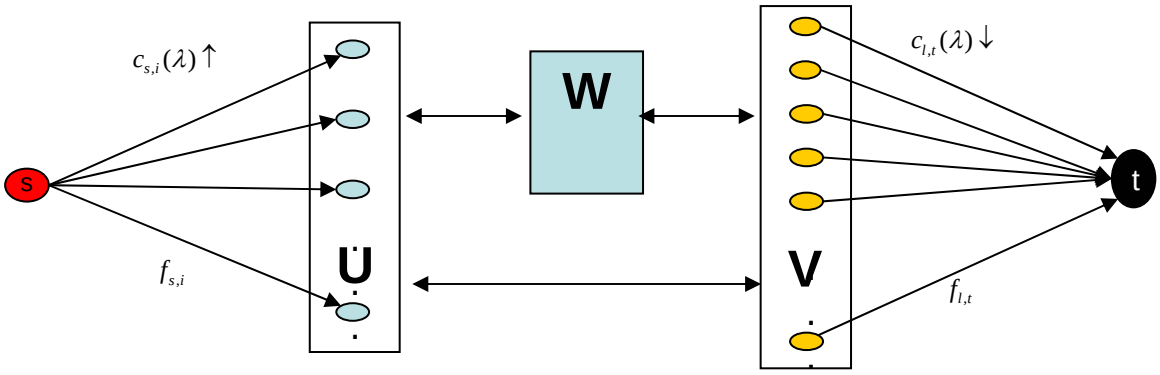

Figure 7. A More General Network.

A simple residue path is,

$$s \to u_i \to w_{k_1} \to \ldots \to w_{k_r} \to v_l \to t \tag{3.1}$$

satisfying one of the two:
case a) $\lambda_{s,i} < \lambda_{l,t}$ and $\hat{c}_{path} > 0$ ($\hat{c}_{path}$ is the residue capacity of the path) or
case b) $\lambda_{s,i} > \lambda_{l,t}$ and $\hat{c}_{reversed\_path} > 0$.
There may not be any w-nodes in a simple residue path. For parametric maximum-flow problems on a non-bipartite graph, a few changes have to be made to the algorithm.

SPM Algorithm: For any simple residue path, $s \to u_i \to w_{k_1} \to \ldots \to w_{k_r} \to v_l \to t$, do

a) if $\lambda_{s,i} < \lambda_{l,t}$, pushing $\delta flow = \min\{\hat{c}_{path}, \delta f_{i,l}\}$ from s to t through the augmenting path; or

b) if $\lambda_{s,i} > \lambda_{l,t}$, pushing $\delta flow = \min\{\hat{c}_{reversed\_path}, -\delta f_{i,l}\}$ from t to s through the reversed path ($\delta f_{i,l} < 0$ in this case).

$\delta f_{i,l}$ is the amount of flow that makes the two new lambda values equal after the push if the push is not constrained by the residue capacity:

$$c_{s,i}^{-1}(\overline{f}_{s,i} + \delta f_{i,l}) = c_{l,t}^{-1}(\overline{f}_{l,t} + \delta f_{i,l}). \tag{3.2}$$

$\delta f_{i,l}$ depends only on the two ends and independent of the middle part of the path. After pushing $\delta flow$, update the lambda values according to the new flow values.

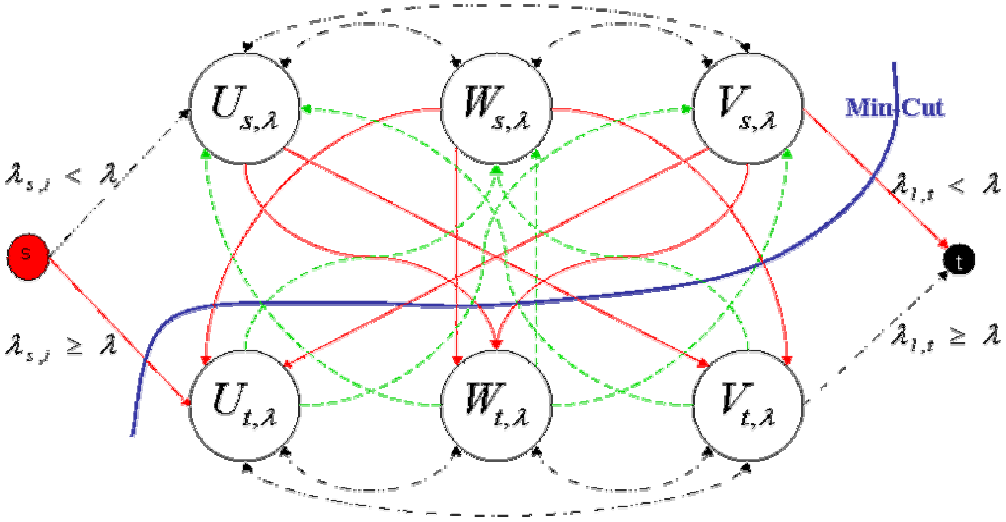

Figure 8. Flows are full to its capacity along all red solid arcs, which are all the directed arcs from the *s*-partition to the *t*-partition; zero flow along all green dash arcs, which are all the directed arcs from the *t*-partition to the *s*-partition. Flows along the black double-dash arcs are the only arcs that might be affected by truncation or rebalancing, which never go across the s-t-partitions.

When the SPM algorithm stops, for any $\lambda \in [a,b]$, there is no simple residue path from $N_{s,\lambda} = U_{s,\lambda} \cup V_{s,\lambda} = \{u \mid \lambda_{s,u} < \lambda\} \cup \{v \mid \lambda_{v,t} < \lambda\}$ to $N_{t,\lambda} = U_{t,\lambda} \cup V_{t,\lambda} = \{u \mid \lambda_{s,u} \geq \lambda\} \cup \{v \mid \lambda_{v,t} \geq \lambda\}$.

Putting all the capacities $c_{s,i}(\lambda)$, $c_{l,t}(\lambda)$ back and truncate the flows that violates the capacity constraint, a min-cut $(N_{s,\lambda} \cup W_{s,\lambda}, N_{t,\lambda} \cup W_{t,\lambda})$ can be generated from $(N_{s,\lambda}, N_{t,\lambda})$ in the same way as before (see **Fig. 8**).

The proof of convergence is the same as the proof in bipartite case.

4. Conclusions and Future Work

The SPM algorithm was implemented for a particular problem. On all large data sets we experimented with, it took SPM algorithm less time to get the complete curve than the time needed by the Preflow-Push algorithm to find the solution at a single lambda value.

We have not provided an analysis of complexity of the SPM algorithm. We would certainly like to see this is done in the future.

References

Ahuja, R.K., Magnanti, T.L., & Orlin, J.B., Network Flows, Prentice Hall, New Jersey 1993.

Elias, P., Feinstein, A., & Shannon, C.E. (1956), Note on Maximum Flow Through a Network, IRE Transformations on Information Theory IT-2, 117-119.

Ford & Fulkerson (1956), Maximum Flow Through a Network, Canadian Journal of Mathematics 8, 339-404.

Ford, L.R. & Fulkerson, D.R., Flows in Networks, Princeton University Press, Princeton, NJ, 1962.

Gallo, G., Grigoriadis, M. D., & Tarjan R.E., A Fast Parametric Maximum Flow Algorithm and Applications, SIAM J. Computing, Vol. 18, No. 1, pp. 30-55, February 1989.

Goldberg, A.V. & Tarjan, R.E., A New Approach to the Maximum Flow Problem, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 136-146; J. Assoc. Comput. Mach., 35 (1988).

Appendix A:

Definition 1: $\{\alpha_k\}_{k=1}^{+\infty}$ is *absolutely* convergent to a finite limit if $\sum_{k=1}^{+\infty}|\alpha_k-\alpha_{k+1}|<+\infty$.

Lemma 1: A bounded non-decreasing (or increasing) sequence is absolutely convergent.

Lemma 2: Let $\{\alpha_k\}_{k=1}^{+\infty}$ and $\{\beta_k\}_{k=1}^{+\infty}$ be two non-decreasing sequences with $\lim_{k\to\infty}\alpha_k=\alpha$ and $\lim_{k\to\infty}\beta_k=\beta<+\infty$ then $\lim_{k\to\infty}(\beta_k-\alpha_k)=\alpha-\beta$, where α can be $+\infty$.

Definition 2: $\{\alpha_k\}_{k=1}^{+\infty}$ is semi-increasing if $\sum_{a_k>a_{k+1}}(a_k-a_{k+1})<+\infty$; i.e. total decreasing movements is finite. $\{\alpha_k\}_{k=1}^{+\infty}$ is semi-decreasing if $\sum_{a_k<a_{k+1}}(a_k-a_{k+1})>-\infty$.

Lemma 3: A semi-increasing (or decreasing) sequence converges (to a finite or infinite number).

Lemma 4: A bounded semi-increasing (decreasing) sequence $\{\gamma_k\}_{k=1}^{+\infty}$ is also semi-decreasing (increasing). It is absolutely convergent.

Proof: Define $\delta_k=\sum_{\substack{\gamma_{k'}>\gamma_{k'+1}\\ k'+1\le k}}(\gamma_{k'}-\gamma_{k'+1})$. $\{\delta_k\}_{k=1}^{+\infty}$ is bounded non-decreasing and absolutely convergent (Lemma 1). $\sum_{(\gamma_{k+1}<\gamma_k)}[(\gamma_{k+1}-\gamma_k)+(\delta_{k+1}-\delta_k)]=0$ by the definition of δ_k.

$$\sum_{\gamma_{k+1}>\gamma_k}(\gamma_{k+1}-\gamma_k)=\sum_{\gamma_{k+1}>\gamma_k}(\gamma_{k+1}-\gamma_k)+\sum_{(\gamma_{k+1}<\gamma_k)}[(\gamma_{k+1}-\gamma_k)+(\delta_{k+1}-\delta_k)] \qquad (A.1)$$

First term in (A.1) $\sum_{\gamma_k>\gamma_{k+1}}(\gamma_k-\gamma_{k+1})<+\infty$ because $\{\gamma_k\}_{k=1}^{+\infty}$ is semi-increasing. Second term in (A.1) $\sum_{k=1}^{+\infty}[(\gamma_{k+1}+\delta_{k+1})-(\gamma_k+\delta_k)]<+\infty$ because $\{\gamma_k+\delta_k\}_{k=1}^{+\infty}$ is a bounded non-decreasing sequence and absolutely convergent. Therefore,

$$\sum_{\gamma_{k+1}>\gamma_k}(\gamma_{k+1}-\gamma_k)<+\infty. \qquad (A.2)$$

Finally,

$$\sum_{k=1}^{+\infty}|\gamma_k-\gamma_{k+1}|=\sum_{\gamma_k>\gamma_{k+1}}(\gamma_k-\gamma_{k+1})+\sum_{\gamma_{k+1}>\gamma_k}(\gamma_{k+1}-\gamma_k)<+\infty. \qquad (A.3)$$

•