# Short Course on Parametric Network Flow

S. Thomas McCormick *

June 27, 2011

**Abstract**

There are many important applied problems that can be formulated as max flow/min cut where the arc capacities depend on a scalar parameter $\lambda$. We call this class of problems parametric network flow for short. Ideally we would like to be able to find an efficient algorithm that will find a max flow and a min cut for *every* value of $\lambda$. In general this is not possible as there are too many critical values of $\lambda$ where the optimal solutions change. Thus we focus on sub-classes where there are provably a small number of critical values. In particular we mostly restrict to classes where capacities are linear in $\lambda$, and where the topology of the parametrized arcs guarantees that optimal cuts are *nested* in $\lambda$. (Happily, such classes include most of the important applications.) For such classes we show how to adapt the Push Relabel max flow/min cut algorithm such that it can find optimal max flows and min cuts for all values of $\lambda$ in the same asymptotic time as a single max flow. If time permits, we will also consider some extensions of these ideas.

## 1 Problems

**This is an updated and corrected version of what was handed out at the summer-school, with answers to all the problems. Answers are given in sans-serif font. If you have any corrections, questions, or further comments about any of this, please email me at `Tom.McCormick@sauder.ubc.ca`.**

**Question 1.** Let $G = (N, A)$ be a max flow network with return arc $t \to s$. Define $S = \{s \to i \in A\}$, i.e., the subset of arcs with tail $s$. For $F \subseteq S$ define $v(F)$ to be the max flow value in the network with capacities $u'_a$ defined by $u'_a = 0$ for $a \in S - F$, and $u'_a = u_a$ otherwise, i.e., we set the capacities of arcs in $S - F$ to zero and otherwise leave the capacities alone. Thus $v(\emptyset) = 0$ and $v(S)$ is the optimal max flow value in the original $G$.

**(a)** Prove that $F_1 \subseteq F_2 \subseteq S$ implies that $v(F_1) \leq v(F_2)$, i.e., that $v(F)$ is monotone.

Let $x(F)$ be a max flow corresponding to $v(F)$, so that $\text{val}(x(F)) = v(F)$. Since $F_1 \subseteq F_2$, $x(F_1)$ is a feasible flow for the $F_2$ network, so $v(F_2) \geq \text{val}(x(F_1)) = v(F_2)$.

**(b)** Prove that $v(F)$ is submodular.

We need to show that $v(F_1) + v(F_2) \geq v(F_1 \cup F_2) + v(F_1 \cap F_2)$. Let $x(F_1 \cap F_2)$ be a max flow corresponding to $v(F_1 \cap F_2)$. Use any augmenting path algorithm starting from $x(F_1 \cap F_2)$ to extend

*Sauder School of Business, University of British Columbia, Vancouver, BC V6T 1Z2 Canada. Supported by an NSERC Operating Grant.

it to a max flow $x(F_1 \cup F_2)$ corresponding to $v(F_1 \cup F_2)$. Note that for $s \to i \in F_1 \cap F_2$ we must have $x(F_1 \cup F_2)_{si} = x(F_1 \cap F_2)_{si}$, since the optimality of $x(F_1 \cap F_2)$ implies that no augmenting path in the $F_1 \cup F_2$ network can use any arc of $F_1 \cap F_2$. This implies that

$$\begin{aligned}\sum_{s\to i\in F_1} x(F_1\cup F_2)_{si} + \sum_{s\to i\in F_2} x(F_1\cup F_2)_{si} &= \sum_{s\to i\in F_1\cup F_2} x(F_1\cup F_2)_{si} + \sum_{s\to i\in F_1\cap F_2} x(F_1\cup F_2)_{si}\\ &= v(F_1\cup F_2) + v(F_1\cap F_2).\end{aligned}$$

Now conformal decomposition implies that we can transform $x(F_1 \cup F_2)$ into a flow $x'(F_1)$ feasible for the $F_1$ network such that $x(F_1 \cup F_2)_{si} = x'(F_1)_{si}$ for all $s \to i \in F_1$, and similarly we can transform $x(F_1 \cup F_2)$ into a flow $x'(F_2)$ feasible for the $F_2$ network such that $x(F_1 \cup F_2)_{si} = x'(F_2)_{si}$ for all $s \to i \in F_2$. Note that

$$\sum_{s\to i\in F_1} x'(F_1)_{si} + \sum_{s\to i\in F_2} x'(F_2)_{si} = \sum_{s\to i\in F_1\cap F_2} x(F_1\cap F_2)_{si} + \sum_{s\to i\in F_1\cup F_2} x(F_1\cup F_2)_{si}$$

since every $s \to i$ arc is counted exactly the same number of times (0, 1, or 2) on both sides.

Then $v(F_1) \geq \sum_{s\to i\in F_1} x'(F_1)_{si}$ and $v(F_2) \geq \sum_{s\to i\in F_2} x'(F_2)_{si}$, so

$$\begin{aligned}v(F_1)+v(F_2) &\geq \sum_{s\to i\in F_1} x'(F_1)_{si} + \sum_{s\to i\in F_2} x'(F_2)_{si}\\ &= \sum_{s\to i\in F_1\cup F_2} x(F_1\cap F_2)_{si} + \sum_{s\to i\in F_1\cup F_2} x(F_1\cup F_2)_{si}\\ &= v(F_1\cup F_2)+v(F_1\cap F_2).\end{aligned}$$

Note that (a) and (b) imply that $Q = \{y \in \mathbb{R}^S \mid y(F) \leq v(F)\ \forall F \subseteq S\}$ is a *polymatroid*.

The *flow polyhedron* is $P(G) = \{x \in \mathbb{R}^A \mid x \text{ is a feasible flow in } G\}$. If $X \subseteq A$ is a subset of arcs, then the *projection* of $P(G)$ onto $X$ is $P(X) = \{y \in \mathbb{R}^X \mid \exists x \in P(G) \text{ s.t. } y_a = x_a\ \forall a \in X\}$, i.e., the linear algebraic projection of $P(G)$ onto the components in $X$.

**(c)** We would like to show that $Q = P(S)$, i.e., that the flow polyhedron projected onto the arcs with tail $s$ is a polymatroid. The only thing left to prove is that every $q \in Q$ also belongs to $P(S)$, i.e., that if $q \in Q$ then there is a feasible flow $x$ in $G$ whose projection on $S$ is $q$. Prove this.

Let $G(q)$ be the max flow network with $u_{si}$ replaced by $q_{si}$ for all $s \to i \in S$, let $x(q)$ be a max flow in $G(q)$, and define $S(q)$ to be a corresponding min cut. If $x(q)$ saturates all arcs in $S$ we are done, so to get a contradiction assume that there is at least one $s \to i \in S$ with $x(q)_{si} < q_{si}$. This implies that the set $I = \{s \to i \in S \mid i \in S(q)\}$ is non-empty.

Now conformally decompose $x(q)$ into flows on $s$–$t$ paths. Note that any path $P$ whose first arc $s \to j$ is not in $I$ cannot contain any other arc of $\delta^+(S(q))$ besides $s \to j$ (since it would then have to also contain an arc of $\delta^-(S(q))$, and $x(q)$ is zero on all such arcs). Thus when we subtract out the flow of all such paths from $x(q)$, we get a new flow $x(I)$ which still satisfies complementary slackness with $S(q)$, so that $x(I)$ is a max flow in the network corresponding to $v(I)$. But $\text{val}(x(I)) < \sum_{s\to i\in I} x(q)_{si} \leq \sum_{s\to i\in I} q_{si} \leq v(I)$ (by feasibility of $q$ for $Q$), contradicting that $x(I)$ is a max flow for the $v(I)$ network.

**(d)** Note that $S = \delta^+(\{s\})$. This makes it tempting to conjecture that if $C = \delta^+(T)$ for some $s$–$t$ cut $T$, then $P(C)$ is also a polymatroid. Prove that this is true or give a counterexample showing that this conjecture is false.

The conjecture is false: Consider the network in Figure 1 with $N = \{s, 1, 2, t\}$, $A = \{s \to 1, s \to$

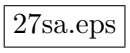


Figure 1: Counterexample for #27 (a)

$2, 1 \to t, 2 \to t, 1 \to 2\}$, and $u = (4, 10, 10, 4, 1)$. Let $T = \{s, 1\}$ so that $C = \{s \to 2, 1 \to 2, 1 \to t\}$. Put $X = \{s \to 2, 1 \to 2\}$ and $Y = \{1 \to 2, 1 \to t\}$, so that $X \cap Y = \{1 \to 2\}$ and $X \cup Y = \{s \to 2, 1 \to 2, 1 \to t\}$. Then $v(X) = v(Y) = 4$, $v(X \cap Y) = 1$, and $v(X \cup Y) = 8$, so that $v(X) + v(Y) = 4 + 4 \not\geq 1 + 8 = v(X \cap Y) + v(X \cup Y)$. Thus $v$ is not submodular, so $P(C)$ is not a polymatroid.

**Question 2.** Consider the special case of min-cost flow on a max flow network where $c_a = 0$ for all $a \in A$ (including $t \to s$) except for arcs with tail $s$, where $c_{si} \leq 0$. Thus we are essentially looking for an $s$–$t$ flow that maximizes $\sum_i |c_{si}| x_{si}$. Use Question 1 to solve this min-cost flow problem in $O(1)$ max flows time.

Question 1 (c) showed that the feasible flows projected onto the arcs with tail $s$ form a polymatroid, so we can optimize via the Greedy Algorithm. Re-number the nodes $i$ such that $s \to i \in A$ as $1, 2, \ldots, k$ so that we have $c_{s1} \leq c_{s12} \leq \ldots \leq c_{sk}$. In this case the Greedy Algorithm amounts to temporarily setting the capacities of all $s \to i$ arcs to zero. Then we loop for $i = 1, 2, \ldots, k$, where at iteration $i$ we have already restored capacities $u_{s1}$, $u_{s2}$, $\ldots$, $u_{s,i-1}$ and we additionally restore capacity $u_{si}$. We then compute a max flow for this new capacity starting from the max flow for the previous version of the capacities.

Note that all capacities at $s$ are monotone non-decreasing, and capacities at $t$ are constant, throughout this algorithm, so this falls into the GGT framework. Thus we can solve this problem in $O(1)$ max flows time.

**Question 3.** A common situation in practice is that we have a finite set $N$ of objects, and the *benefit* of selecting object $i$ is $b_i$; if $b_i < 0$, then $|b_i|$ is the cost of selecting $b$. We want to select the subset of $N$ of maximum benefit. This is easy: just select the subset of $N$ of elements $i$ with $b_i > 0$.

What makes the problem complicated is that there are precedence constraints represented by a set $A$ of arcs on $N$, so that if $i \to j \in A$ and we select element $i$, then we must also select element $j$. Then subset $C \subseteq N$ is feasible iff for all $i \in C$ and $i \to j \in A$, we also have $j \in C$; such a subset is called a *closure*. The problem we want to solve is the *max closure* problem: $\max_{C \text{ a closure}} b(C)$, i.e., the net benefit of a closure.

Examples of this:

1. **Open-pit mining:** $N$ is blocks of earth, $b_i$ is the net profit/cost of mining block $i$, $i \to j \in A$ if we need to mine block $j$ to get at block $i$.

2. **Vacation problem:** $N = G \cup B$, where $G$ is a set of "good" items with benefits $b_i > 0$, and $B$ is a set of "bad" items with costs $c_j > 0$. To bring good item $i$ on your vacation, it is necessary to bring the subset of bad items $\{j \in B \mid i \to j \in A\}$.

3. **Assigning residues to protein positions:** $N = P \cup B$, where $P$ is a set of positions on the backbone of a protein chain, and $B$ is a set of pairs of positions (*bonds*). We are

selecting the subset of positions that will be filled with polar residues. There is a cost $c_j$ for putting a polar residue in position $j$, but if we put polar residues in both positions $i$ and $j$ where $(i, j) \in B$, then we get a benefit $b_{(i,j)}$. (This comes from a paper by Jon Kleinberg.)

Construct a max flow network from $(N, A)$ as follows. Define $P = \{i \in N \mid b_i > 0\}$ and $M = \{j \in N \mid b_j < 0\}$. Add a source $s$ and a sink $t$. For each node $i \in P$, make an arc $s \to i$ with $u_{si} = b_i$. For each node $j \in M$, make an arc $j \to t$ with $u_{jt} = |b_j|$. For each arc $i \to j \in A$, put $u_{ij} = +\infty$. Prove that $C$ is a feasible closure iff $C + s$ is a finite capacity cut in this network, and that minimizing the capacity of $C + s$ is equivalent to maximizing $b(C)$. What is the relation between the objective value of the max flow problem and the objective value of the max closure problem?

Note that $C$ is a feasible closure iff $\delta^+(C) = \emptyset$ (w.r.t. $(N, A)$). Thus the only arcs in $\delta^+(C + s)$ (w.r.t. the max flow network) are arcs $s \to i$ with $i \notin C$, and arcs $j \to t$ with $j \in C$, and all these arcs have finite capacity, so $C + s$ is a finite capacity cut.

Conversely, if $C + s$ is a finite capacity cut, then there cannot exist any $a \in A \cap \delta^+(C)$, else this $a$ would make $\text{cap}(C + s) = +\infty$, so $C$ is a closure.

Now for $C$ a closure, $\text{cap}(C + s) = u(P - C) + u(M \cap C) = b(P - C) - b(M \cap C) = b(P) - b(P \cap C) - b(M \cap C) = b(P) - b(C)$. Therefore $\min_C \text{cap}(C + s) = \min_C (b(P) - b(C)) = b(P) - \max_C b(P)$, so minimizing $\text{cap}(C)$ does indeed maximize $b(C)$. The relation between the two objective values is that their sum is $b(P)$, a constant.

**Question 4.** Consider the open-pit mining problem formulated as an optimal closure problem as in Question 3, keeping the same notation.

**(a)** Recall that a min cut in the constructed max flow problem identifies an optimal closure. Thus if there are multiple min cuts, there are multiple optimal closures. In practice, the miners would like us to report an optimal closure with a minimum number of nodes (because in any larger optimal closure, the additional blocks will only break even for us instead of making us money). Once we've computed a max flow, show that there is a unique such optimal closure with a minimum number of nodes, and show how to compute it in $O(m)$ time.

The lecture showed that there is a unique minimal min cut (i.e., the intersection of all min cuts), which can be computed in $O(m)$ time via breadth-first search from $s$ of the residual graph w.r.t. any max flow.

In practice the miners are even more picky than this. There is a huge fixed cost to starting a new mine that is paid for by impatient investors who want to be paid back as soon as possible. To keep them happy we must identify a closure whose ratio of net benefit to number of blocks is as high as possible, i.e., that maximizes $\frac{b(C)}{|C|}$.

A standard technique to deal with such ratio problems is to parametrize the denominator. That is, we consider the parametric problem where we maximize the (linear) objective $b(C) - \lambda|C|$. Define $b(\lambda) = \max_{C \text{ a closure}} b(C) - \lambda|C|$, the maximum value of the parametric function over any closure.

**(b)** Show that the parametric max flow network corresponding to the parametric objective $b(C) - \lambda|C|$ is a (reverse) GGT-type network (which may have piecewise linear capacity functions).

For each $i \in N$ define $b_i^\lambda = b_i - \lambda$, so that $b(C) - \lambda|C| = b^\lambda(C)$. Construct the usual max flow instance corresponding to the closure problem with benefits $b^\lambda$ in place of $b$ as in Question 3.

To show that this parametric network belongs to the GGT framework, consider the effect on capacities at $s$ and $t$ when we change from $\lambda = \lambda_1$ to $\lambda = \lambda_2$ with $\lambda_1 < \lambda_2$. If we had $b_i^{\lambda_1} < 0$, then $b_i^{\lambda_2} < b_i^{\lambda_1}$ so we get that $u_{it}(\lambda_2) = |b_i^{\lambda_2}| > |b_i^{\lambda_1}| = u_{it}(\lambda_1)$, so the parametric capacity at $t$ has increased. If we have $b_i^{\lambda_1} > b_i^{\lambda_2} \geq 0$, then $u_{si}(\lambda_2) = b_i^{\lambda_2} < b_i^{\lambda_1} = u_{si}(\lambda_1)$, so the parametric capacity at $s$ has decreased. If we have $b_i^{\lambda_1} \geq 0 \geq b_i^{\lambda_2}$, then $u_{si}(\lambda_2) = 0 \leq b_i^{\lambda_1} = u_{si}(\lambda_1)$, and $u_{it}(\lambda_2) = |b_i^{\lambda_2}| \geq 0 = u_{it}(\lambda_1)$, so the parametric capacity at $s$ has decreased and the parametric capacity at $t$ has increased. This all satisfies the (reverse) GGT framework. Note that the capacities are now piecewise linear, as the capacity switches from $s \to i$ to $i \to t$ when $b_i - \lambda$ goes from positive to negative; alternatively, one could get rid of these piecewise linear capacities by adding a large constant to the capacities $u_{si}$ and $u_{it}$ for all $i \neq s, t$, thereby moving all the parameters to the $s \to i$ arcs.

Assume that there is at least one closure with $b(C) > 0$, so that $b(0) > 0$. As $\lambda \to \infty$, for any closure $C$ with $C \neq \emptyset$, $b(C) - \lambda|C|$ becomes negative. Since this is a parametric linear program, we know that $b(\lambda)$ is piecewise linear, so it is continuous, so there must exist some $\lambda > 0$ with $b(\lambda) = 0$. Since $C = \emptyset$ is a closure with $b(\emptyset) = 0 = |\emptyset|$, once $b(C) - \lambda|C|$ hits zero it stays at zero. Define $\lambda^* = \min\{\lambda \mid b(\lambda) = 0\}$, so that $\lambda^* > 0$.

**(c)** Prove that $\lambda^* = \max_{C \text{ a closure}} b(C)/|C|$.

Let $C^*$ be a closure solving $\max_{C \text{ a closure}} b(C)/|C|$ with objective value $z^* = b(C^*)/|C^*|$. Now $b(C^*) - z^*|C^*| = 0$. If there was some closure $C'$ with $b(C') - z^*|C'| > 0$ for $\lambda = z^*$, then $b(C')/|C'| > z^* = b(C^*)/|C^*|$, contradicting the optimality of $C^*$. Thus $\lambda^* \leq z^*$. For $\lambda < z^*$, $b(C^*) - \lambda|C^*| > b(C^*) - z^*|C^*|$, so that $b(\lambda) > 0$, and so $\lambda^* \geq z^*$, and so $\lambda^* = z^*$.

Here is a reasonable algorithm for computing $\lambda^*$. Start with $\lambda_0 = 0$ and compute a max flow; if there are no closures with positive net benefit we stop here. Otherwise we find closure $C_1$. To get $b(\lambda) \leq 0$ we in particular need $b(C_1) - \lambda|C_1| \leq 0$, or $\lambda \geq b(C_1)/|C_1| = \lambda_1$. Compute a new max flow for $\lambda_1$ (using reverse GGT; note that the slope of a cut changes whenever $b_i - \lambda$ crosses zero, and so we might have to have $O(n)$ more steps in our GGT where we stop at one of these capacity function breakpoints, but this does not change the complexity of GGT). If there are no closures with positive net benefit in this network, then we know from (c) that $\lambda^* = \lambda_1$ and that the corresponding max ratio closure is $C_1$. Otherwise we get an optimal closure $C_2$ for $\lambda_1$ and compute a new lower bound $b(C_2)/|C_2| = \lambda_2$ on $\lambda$, and we continue like this. [This algorithm is sometimes called *Dinkelbach's Algorithm*, or (discrete) *Newton's Algorithm*. This algorithm can also apply to non-GGT parametric max flow, and non-max flow parametric problems, but the GGT version works out especially nicely as indicated below.]

**(d)** Prove that $\lambda_h > \lambda_{h-1}$ for $h > 1$.

We know that $b(C^h) - \lambda_{h-1}|C^h| > 0$, else we would have stopped the algorithm. Then we get that $\lambda_h = b(C^h)/|C^h| > \lambda_{h-1}$.

**(e)** Prove that $C_h \subset C_{h-1}$ for $h > 1$ (a *strict* subset). [This implies that this algorithm performs at most $n-1$ iterations. Since we are solving a GGT parametric max flow network for a monotone sequence of $\lambda$, we can do the whole algorithm in the time of $O(1)$ max flows. This is a fairly typical example of how the on-line capability of GGT is used in practice.]

From the lecture we already know that $C_h \subseteq C_{h-1}$, the only question is whether the inclusion is strict. If we had $C_h = C_{h-1}$, then we would have that $\lambda_{h-1} = b(C_{h-1})/|C_{h-1}| = b(C_h)/|C_h| = \lambda_h$, contradicting (d).

**Question 5.** Here is a non-obvious application where we want to know *all* breakpoints and the corresponding min cuts of a GGT parametric max flow capacity function $\text{cap}(\lambda)$. We have a max flow network with $k$ arcs with tail $s$, and we name the nodes so that these $k$ arcs are $s \to s_1$, $s \to s_2$, ..., $s \to s_k$. The importance of having flow in arc $s \to s_i$ is given by its weight $w_i$ (given data). Then if $x$ is a feasible flow, the goodness of flow on $s \to s_i$ is $x_{s,s_i}/w_i$ (so a big $w_i$ means that $x_i$ must be bigger to be good).

Intuitively we want a flow $x$ that is good on all $s \to s_i$ arcs simultaneously. A very strong way to enforce this is to ask that

[1] $x$ be a max flow. Subject to [1], we ask that

[2] the minimum $x_{s,s_i}/w_i$ over all $i$ is as large as possible. Subject to [1] and [2], we ask that

[3] the next largest $x_{s,s_i}/w_i$ over all $i$ is as large as possible. ... Subject to [1]–[$j$] we ask that

[$j+1$] the $j$th largest $x_{s,s_i}/w_i$ over all $i$ is as large as possible. ... Subject to [1]–[$k$] we ask that

[$k+1$] the largest $x_{s,s_i}/w_i$ over all $i$ is as large as possible.

Such a flow is called a *lexicographic max flow*.

We attack this problem by considering the GGT max flow network with capacities $u_{s,s_i}(\lambda) = w_i\lambda$.

**(a)** As a warm-up, let's solve the problem of finding a max flow subject to [1] and [2] only, called *maximin flow sharing*. Define $l_{\min}$ to be the smallest breakpoint of $\text{cap}(\lambda)$ that is larger than zero. Define a non-parametric network $G_l$ with lower bounds $w_i l_{\min}$ on each $s \to s_i$, and the other data being the same. Prove that an optimal flow $x(l_{\min})$ for the $l_{\min}$ network gives a feasible flow for $G_l$. (Note that $x(l_{\min})$ will usually be fractional.)

Since a min cut associated with $\lambda = 0$ is $\{s\}$, the cut $\{s\}$ is also the min cut associated with the interval $[0, l_{\min}]$. Breakpoint $l_{\min}$ is determined by a new cut $S_1$ such that the capacity of $S_1$ at $l_{\min}$ equals the capacity of $\{s\}$ at $l_{\min}$. Since $\{s\}$ is the min cut associated with the interval $[0, l_{\min}]$, every max flow for every $\lambda \in [0, l_{\min}]$, including $\lambda = l_{\min}$, must saturate every arc of $\delta^+(\{s\})$, i.e., $x(l_{\min})_{s,s_i} = w_i l_{\min}$ for all $i$, so that $x(l_{\min})$ is indeed feasible for the $l_{\min}$ network.

**(b)** Continuing with (a), consider a residual max flow network $G_r$ w.r.t. $x(l_{\min})$ where $r_{s,s_i} = \infty$ and $r_{s_i,s} = 0$ for all $i$, and we have the usual $r_{ij}$ otherwise. Compute a max flow $y$ in $G_r$. Prove that $x(l_{\min}) + y$ solves the maximin flow sharing problem. [Warning, this is described wrong in the GGT paper, which contains a few other (known) errors.]

Note that $\min_i x(l_{\min})_{s,s_i}/w_i = l_{\min}$, so we are claiming that $l_{\min}$ is the optimal objective value of [2] for maximin flow sharing. To get a contradiction, suppose that there is some other flow $x'$ with $\min_i x'_{s,s_i}/w_i > l_{\min}$. Recall from (a) that $x(l_{\min})$ was complementary slack with both cuts $\{s\}$ and $S_1$. Set $S_1' = S_1 - s$. By conservation we know that $x(l_{\min})(\delta^+(S_1')) = x(l_{\min})(\delta^-(S_1'))$, by complementary slackness with $\{s\}$ we know that $x(l_{\min})(\delta^+(S_1')) = u(l_{\min})(\delta^+(S_1')) = l_{\min} w(\delta^+(S_1'))$,

and by complementary slackness with $S_1$ we know that $x(l_{\min})(\delta^-(S_1')) = u(\delta^-(S_1'))$. This gives that $l_{\min} w(\delta^+(S_1')) = u(\delta^-(S_1'))$.

Our assumption on $x'$ implies that $x'(\delta^+(S_1')) > l_{\min} w(\delta^+(S_1'))$, and feasibility of $x'$ implies that $x'(\delta^-(S_1')) \leq u(\delta^-(S_1'))$. Together these give $x'(\delta^+(S_1')) > x'(\delta^-(S_1'))$ contradicting that $x'$ satisfies conservation. Thus $l_{\min}$ is indeed the optimal value of [2] among max flows.

Now changing from $x(l_{\min})$ to $x(l_{\min}) + y$ keeps the property of having objective value $l_{\min}$ for [2], and changes to a max flow, so this must be an optimal solution to maximin flow sharing.

**(c)** Now consider the full lexicographic problem. Use GGT to compute all of the breakpoints of $\text{cap}(\lambda)$. For each $s_i$ there is some corresponding breakpoint $\lambda(s_i)$ such that $s_i$ moves from the $t$ side of the parametric min cut to the $s$ side of the parametric min cut at breakpoint $\lambda(s_i)$. Now define a non-parametric max flow problem with capacities $u'$, where $u'_{s,s_i} = w_i \lambda(s_i)$, and $u'_a = u_a$ otherwise. Prove that any max flow in this non-parametric max flow network must be a lexicographic max flow. [All of this can be computed in $O(1)$ max flows time via GGT.]

Let's proceed by induction on the breakpoints. Part (b) establishes the base of the induction for the first breakpoint $l_{\min}$. Now construct an inductive network $G_1$ as follows. Start with the residual network w.r.t. $x(l_{\min})$, and delete all the nodes in $S_1'$ (leaving the capacities of the remaining $s \to s_i$ arcs as $w_i \lambda$). Note that deleting the nodes in $S_1'$ does not really affect max flows in $G_1$ since no residual $s$–$t$ path can exit $S_1'$ since it is a min cut for $x(l_{\min})$. Now in $G_1$ we have regained the property that the cut $\{s\}$ is a min cut at $\lambda = 0$, and now the first breakpoint in $G_2$ will be $\lambda_2 - l_{\min}$ (where $\lambda_2$ is the second breakpoint for the original network), determined by the next min cut $S_3$.

So, by the same reasoning as in (b), a max flow $x(2)$ in $G_2$ must saturate all remaining $s \to s_i$ arcs, so that $x(l_{\min}) + x(2)$ satisfies [2] and [3]. Now construct $G_3$ as the residual network of $G_2$ w.r.t. $x(2)$ minus the nodes of $S_3' = S_3 - s$ and get a max flow $x(3)$ in $G_3$ such that $x(l_{\min}) + x(2) + x(3)$ satisfies [2]–[4], etc.

We end up with a max flow $x^*$ that satisfies [1]–[$k+1$] and which saturates all $s \to s_i$ arcs w.r.t. capacities $u'$. We see that $x^*$ satisfies [1] (is a max flow) because no feasible flow can put any more flow through each $S_i'$ than $x^*$ does. Conversely, any max flow w.r.t. $u'$ must saturate each $S_i'$, and so also satisfy [2]–[$k+1$], so any max flow w.r.t. $u'$ is an optimal lexicographic max flow.

**Question 6.** This problem shows that some parametric optimization problems are so special that we can solve them even faster than by using GGT.

We say that set function $f : 2^E \to \mathbb{R}$ is *of type 2* if we can determine the value of $f$ on any set based on knowing only its values on the empty set, singletons, and sets of size two. It can be shown that submodular set functions of type 2 are precisely the submodular functions that are min cut functions of max flow networks.

Here's an example of a submodular set function of type 2 that is not obviously a network function: Suppose that we want to schedule $n$ jobs on a single machine where once we start a job we can't interrupt it (no *preemption*), and job $j$ has *processing time* $p_j > 0$. Let $J = \{1, 2, \ldots, n\}$ denote the set of jobs.

Define $C_j$ to be the *completion time* of job $j$. We want to find constraints characterizing $Q$, the convex hull of $\{C \in \mathbb{R}^n \mid C_j \text{ is a set of feasible completion times}\}$ (why do we need convex hull here? Because this set is not even connected, and so is not convex). For $S \subseteq J$ let's consider the possible constraint $\sum_{j \in S} p_j C_j \geq g(S)$ for $Q$ for some as-yet unknown RHS $g(S)$. Clearly the best possible RHS is $g(S) = \min_{C \in Q} \sum_{j \in S} p_j C_j$. Furthermore, it is clear that this minimum will be attained by a schedule that puts all jobs in $S$ first with no *idle time*, i.e., as soon as one

job finishes, the next one starts. Define $s = |S|$. Such a schedule is uniquely determined by a permutation $\pi$ where $\pi_1$ is the first job, $\pi_2$ is the second job, ..., $\pi_n$ is the last job, and where $S = \{\pi_1, \ldots, \pi_s\}$.

**(a)** Compute $C_{\pi_j}$ for all $j$, and use this to figure out what $g(S)$ is.

We have $C_{\pi_1} = p_{\pi_1}$, $C_{\pi_2} = C_{\pi_1} + p_{\pi_2} = p_{\pi_1} + p_{\pi_2}$, ..., $C_{\pi_n} = C_{\pi_{n-1}} + p_{\pi_n} = p_{\pi_1} + p_{\pi_2} + \ldots + p_{\pi_n}$. Therefore for this $\pi$ the value of $\sum_{j \in S} p_j C_j$ is $p_{\pi_1} p_{\pi_1} + p_{\pi_2}(p_{\pi_1} + p_{\pi_2}) + \ldots + p_{\pi_s}(p_{\pi_1} + p_{\pi_2} + \ldots + p_{\pi_s}) = \frac{1}{2}\left(\left(\sum_{j \in S} p_j\right)^2 + \sum_{j \in S} p_j^2\right)$. Since this expression is independent of $\pi$, it must equal $g(S)$.

**(b)** Prove that $g(S)$ is supermodular.

Now $g(S + k + l) - g(S + l) = \frac{1}{2}(p_k^2 + p_k p(S + l)) \geq \frac{1}{2}(p_k^2 + p_k p(S)) = g(S + k) - g(S)$, and so $g$ is supermodular.

**(c)** Prove that $g(S)$ is of type 2.

Now $g^0 = 0$, $g_j^1 = p_j^2$, and $g_{ij}^2 = p_i^2 + p_j^2 + p_i p_j$, and so $\hat{g}^0 = 0$, $\hat{g}_j^1 = p_j^2$, and $\hat{g}_{ij}^2 = p_i p_j$. Then it is easy to verify that for any $S$, $g(S) = \hat{g}^0 + \sum_{i \in S} \hat{g}_i^1 + \sum_{\{i,j\} \in \binom{S}{2}} \hat{g}_{ij}^2$ and so $g$ is of type 2. [This also gives an easier, alternate proof that $g$ is supermodular, as this is equivalent to $\hat{g}_{ij}^2 \geq 0$.]

Suppose that we have some (possibly fractional) point $\bar{x}$ (probably coming from the LP relaxation of some more complicated scheduling formulation) and we want to determine whether $\bar{x} \in Q$ or not. Then we would like to solve the separation problem

$$\min_{S \subseteq J} \left( \sum_{j \in S} p_j \bar{x}_j - g(S) \right),$$

which is Submodular Function Minimization (SFM) since $g$ is supermodular. That is, if the answer to this problem is non-negative, this proves that $\bar{x} \in Q$, and it not, an optimal $S$ gives a constraint violated by $\bar{x}$.

**(d)** Prove that we can assume w.l.o.g. that $\bar{x}_j \geq p_j$.

If not, then for $S = \{j\}$, the constraint $p_j \bar{x}_j \geq p_j^2$ is already violated, and these are easy to check in linear time.

**(e)** Now show how to use max flow/min cut to solve this separation problem (here we are taking advantage of the fact that this is a special type of submodular function to solve the problem much faster than general SFM).

Let's use $\hat{x}$ for the type 2 parameters for the SFM function $\sum_{j \in S} p_j \bar{x}_j - g(S)$. Then $\hat{x}^0 = 0$, $\hat{x}_j^1 = p_j(\bar{x}_j - p_j)$ (which we can assume is non-negative by (d)), and $\hat{x}_{ij}^2 = -p_i p_j$, which is non-positive, as required. Now we just construct the selection max flow problem as usual, with capacities $c_{sj} = p_j(\bar{x}_j - p_j)$ on the left, and $c_{ij,t} = p_i p_j$ on the right.

[Note: this doesn't quite match what Ridha and I derived in Paris, which had just $p_j \bar{x}_j$ as capacities on one side. But the only real difference is that the other part of the term, $-p_j^2$, got moved onto an "$ii$" node on the other side.]

Now just to be annoying, let's figure out a much simpler and faster separation algorithm.

**(f)** Suppose that $S^*$ minimizes $\sum_{j \in S} p_j \bar{x}_j - g(S)$. Prove that $j \in S^*$ iff $\bar{x}_j \leq p(S^*)$.

Define $D(S) = \sum_{j \in S} p_j \bar{x}_j - g(S)$. Notice that for $j \in S^*$, $D(S^*) - D(S^* - j) = p_j(\bar{x}_j - p(S^*))$. Since $S^*$ minimizes, we have $\bar{x}_j \leq p(S^*)$.

Conversely, suppose that $k \notin S^*$. Similarly, $D(S^* + k) - D(S^*) = p_k(\bar{x}_k - (p(S^*) + p_k))$, and since $S^*$ minimizes, $\bar{x}_k - p_k \geq p(S^*)$, or $\bar{x}_k > p(S^*)$.

**(g)** Now use (f) to design a fast and simple algorithm for solving the separation problem. How fast is your algorithm?

From (f), if $j \in S^*$, then every $k$ with $\bar{x}_k \leq \bar{x}_j$ must also be in $S^*$. Thus it suffices to sort the $\bar{x}_j$ (which takes $O(n \log n)$ time) and to check each set of the $k$ smallest values of $\bar{x}_j$ to see if it violates. This all takes $O(n \log n)$ time.

[This comes from Section 5 of M. Queyranne "Structure of a simple scheduling polyhedron", *Math. Prog.*, **58**, 263–285 (1993), from an idea of P. Tseng.]

Note: It's pretty easy to prove that for and submodular function of type 2, if $\hat{f}^2_{ij} = -p_i p_j$ for numbers $p_i > 0$, then the same algorithm works (using $\chi_k = (\hat{f}^1_k + p_k^2)/p_k$ for the sorting, with essentially the same proof. It's not so clear in the other direction: for such $p_j$ to exist, would need to have $p_j = \sqrt{\hat{f}^1_{ji} \hat{f}^1_{jk} / \hat{f}^1_{ik}}$ for all $i, j, k$, but then what?

Define $t_{\max}$ to be the largest value of $t$ such that all of these constraints are satisfied if we start everything at $t$ instead of at 0, i.e., the largest $t$ such that $\bar{x}_j - t$ satisfies all constraints (note that $t_{\max}$ may be negative). Thus $t_{\max}$ is an upper bound on how late we could start and still feasibly schedule all the jobs ($t_{\max}$ is not exact, because, e.g., for the instance with $J = \{1, 2\}$, $p = (1, 1)$, and $\bar{x} = (1.5, 1, 5)$, it is easy to see that $t_{\max} = 0$, but in fact we would have to choose $t = -0.5$ to get a feasible schedule). This is a parametric min cut problem that is easily seen to fit into the GGT framework, and so we could solve it that way.

**(h)** Instead of using GGT, use (g) to design a fast and simple algorithm for computing $t_{\max}$. How fast is your algorithm?

Note that $t_{\max}$ equals $\max_t\{t \mid \min_S(\sum_{j \in S} p_j(\bar{x}_j - t) - g(S)) \geq 0\}$. From (g) we know that $\min_S(\sum_{j \in S} p_j(\bar{x}_j - t) - g(S))$ is attained at the set $S_k$ consisting of the $k$ smallest values of $\bar{x}_j - t$ for some $k$; note that the $S_k$ are independent of $t$.

Thus we can sort the $\bar{x}_j$ just one time (costing $O(n \log n)$) and use this to generate the $n$ sets $S_k$. Each $S_k$ generates a line in the space with $x$-coordinate $t$ and $y$ coordinate $\sum_{j \in S_k} p_j(\bar{x}_j - t) - g(S_k)$. Define $t_k$ as the $x$ intercept of this line, i.e., where $\sum_{j \in S_k} p_j(\bar{x}_j - t_k) - g(S_k) = 0$ (so that $t_k = [\sum_{j \in S_k} p_j \bar{x}_j - g(S_k)]/p(S_k)$). Then clearly $t_{\max} = \min_k t_k$.

Compute the partial sums $P_1 = p_1$ and $X_1 = p_1 \bar{x}_1$, then $P_k = P_{k-1} + p_k$ and $X_k = X_{k-1} + p_k \bar{x}_k$, $k = 2, \ldots, n$ in $O(n)$ time. Then compute $g(S_1) = p_1 P_1$, $g(S_2) = g(S_1) + p_2 P_2$, $\ldots$, $g(S_n) = g(S_{n-1}) + p_n P_n$ again in $O(n)$ time. Then $t_k = [X_k - g(S_k)]/P_k$, $k = 1, \ldots, n$, and $t_{\max}$ can also be computed in $O(n)$ time. Thus the whole algorithm takes $O(n \log n)$ time.

**Question 7.** Let's suppose that we want to solve the special case of parametric GGT Max Flow/Min Cut where the network is bipartite ($N = \{s\} \cup \{t\} \cup L \cup R$ with all arcs in $(s, L)$, $(L, R)$, and $(R, t)$) and capacities are constant everywhere except that $u_{si} = \lambda$ for all $i$ with $s \to i \in A$. (This special structure is reasonably common in applications; the results in this problem can be extended to more general kinds of parametric GGT problems, or even to Max Flow in general.) Roughly speaking, this kind of parametric max flow is asking us to find a max flow such that the flows on the arcs out of $s$ are as equal as possible.

To make this more precise, suppose that we consider the network where everything is the same except that we replace capacities $u$ with $\hat{u}$, where $\hat{u}$ matches $u$ except that $\hat{u}_{si} = \infty$ for all $s \to i \in A$. Suppose that $\hat{x}$ is some max flow in the $\hat{u}$ network. Suppose that we have $i, j \in L$ and $k \in R$ such that $i \to k$, $j \to k \in A$, $\hat{x}_{si} < \hat{x}_{sj}$ ($i$ and $j$ are "unbalanced"), and $\hat{x}_{ik} < \hat{u}_{ik}$, $\hat{x}_{jk} > 0$ (we can feasibly push flow forward around the cycle $s \to i \to k \leftarrow j \leftarrow s$). Then we could push $\min(\hat{u}_{ij} - \hat{x}_{ij}, (\hat{x}_{sj} - \hat{x}_{si})/2)$ flow around this cycle and make $\hat{x}$ "more balanced". Notice that repeating such steps will produce fractional flows.

Let's totally ignore the complexity of doing such balancing steps and consider only the end result. We call a max flow $\bar{x}$ *balanced* if for each triple $i, j, k$ with $i, j \in L$, $k \in R$, $i \to k$, $j \to k \in A$ and $\bar{x}_{si} < \bar{x}_{sj}$, we have that either $\bar{x}_{ik} = \hat{u}_{ik}$, or $\bar{x}_{jk} = 0$. The amazing fact that we are going to prove here is that a balanced flow in a sense represents max flows and min cuts for *all* values of $\lambda \geq 0$ simultaneously.

Suppose that we want a max flow and min cut for value $\bar{\lambda} \geq 0$. Define $S(\bar{\lambda}) = \{i \in L \mid \bar{x}_{si} < \bar{\lambda}\}$, $T(\bar{\lambda}) = \{k \in R \mid \exists i \in S(\bar{\lambda}) \text{ with } i \to k \in A \text{ and } \bar{x}_{ik} < \hat{u}_{ik}\}$, $\bar{S}(\bar{\lambda}) = L - S(\bar{\lambda})$, and $\bar{T}(\bar{\lambda}) = R - T(\bar{\lambda})$. Define a flow $x(\bar{\lambda})$ by reducing flow on paths $s \to i \to k \to t$ with $i \in \bar{S}(\bar{\lambda})$ and $k \in \bar{T}(\bar{\lambda})$ until $x(\bar{\lambda})_{si} = \bar{\lambda}$ for all $i \in \bar{S}(\bar{\lambda})$.

**(a)** Prove algorithmically that such an $x(\bar{\lambda})$ exists, and give the running time of your algorithm for computing $x(\bar{\lambda})$.

For $i \in \bar{S}(\bar{\lambda})$, let $s \to i \to k \to t$ be a path with positive flow passing through $i$. If $k \in T(\bar{\lambda})$, then there is some $j \in S(\bar{\lambda})$ with $\hat{x}_{jk} < \hat{c}_{jk}$, and so $s \to j \to k \leftarrow i \leftarrow s$ is an unbalanced cycle that we could push flow around and make $\hat{x}_{sj} < \bar{\lambda}$ closer to $\hat{x}_{si} \geq \bar{\lambda}$, contradicting that $\bar{x}$ is balanced. Thus $k \in \bar{T}(\bar{\lambda})$. When we find such a path we can reduce flow in each arc by $\min(\hat{x}_{si} - \bar{\lambda}, \hat{x}_{ik}, \hat{x}_{kt})$.

It takes $O(m)$ time to build the subnetwork of arcs that can participate in such paths. Then we could do, e.g., DFS on this graph. Each time we discover a path ($O(1)$ work), at least one arc is eliminated from this subnetwork, and so there is only $O(m)$ total work.

**(b)** Prove that $x(\bar{\lambda})$ is a max flow for the original capacities when $\lambda = \bar{\lambda}$, and that $C = \{s\} \cup S(\bar{\lambda}) \cup T(\bar{\lambda})$ is a min cut.

Since $x(\bar{\lambda})_{si} \leq \bar{\lambda}$ for all $i \in L$ and $u$ matches $\hat{u}$ elsewhere, $x(\bar{\lambda})$ is feasible. Now it suffices to show complementary slackness between $x(\bar{\lambda})$ and $C$.

We must show that $x(\bar{\lambda})_{si} = u_{si} = \bar{\lambda}$ for each $i \in \bar{S}(\bar{\lambda})$, but this is clear from the construction. We must show that $x(\bar{\lambda})_{jk} = 0$ for $j \in \bar{S}(\bar{\lambda})$ and $k \in T(\bar{\lambda})$. But $k \in T(\bar{\lambda})$ because there exists $i \in S(\bar{\lambda})$ with $i \to k \in A$ and $\bar{x}_{ik} < \hat{u}_{ik}$. But this must imply that $\bar{x}_{jk} = 0$, else we could balance around $s \to i \to k \leftarrow j \leftarrow s$. We must show that $x(\bar{\lambda})_{ik} = \hat{u}_{ik}$ for $i \in S(\bar{\lambda})$ and $k \in \bar{T}(\bar{\lambda})$, but this is clear because $k \in \bar{T}(\bar{\lambda})$ because all such $x(\bar{\lambda})_{ik} = \hat{u}_{ik}$. Finally, we must show that $\bar{x}_{kt} = \hat{u}_{kt}$ for $k \in T(\bar{\lambda})$. If not, then there would be an augmenting path $s \to i \to k \to t$ with $i \in S(\bar{\lambda})$ w.r.t. $x(\bar{\lambda})$; this would also be augmenting for $\bar{x}$, contradicting that it is a max flow.

[Taken from "Balancing Applied to Maximum Network Flow Problems (Extended Abstract)", Tarjan, Ward, Zhang, Zhou, Mao, HP Labs.]

Notice that this immediately gives an alternate proof that min cuts for different values of $\lambda$ are nested.

**(c)** In many situations we don't care about the max flows, but we only want the min cuts. Give an algorithm that can read out the $O(n)$ different min cuts from the $\bar{x}$ flow. What is the running time of this algorithm?

First the algorithm sorts the $i \in L$ by their $\bar{x}_{si}$ values. Suppose that we find that the distinct values of $\bar{x}_{si}$ are $0 = \lambda_0 < \lambda_1 < \lambda_2 < \ldots < \lambda_k$ where $k = O(n)$. Then use the ordering to compute the level sets $S_h = \{i \in L \mid \bar{x}_{si} \leq \lambda_h\}$. Then $S_h$ is the $L$-side of the min cut for the interval $[\lambda_h, \lambda_{h+1}]$. This takes $O(n \log n)$ time.

Now for each $k \in R$ run through all the $i \to k \in A$ and label $k$ with the largest $\lambda_h$ such that there is some $i \to k$ with $\bar{x}_{ik} < \hat{u}_{ik}$, and then sort the $k \in R$ by these labels. The complements of the similar level sets on the $R$-side give the $\bar{T}_h$ which are the $R$-sides of the min cut for the same interval. This takes $O(n \log n)$ time for the sort, and $O(m)$ time to run through all the arcs, for a total of $O(m + n \log n)$ time.

**Question 8.** This problem shows that GGT-type parametric flow can be seen as a special case of a more general parametric optimization theory due to Topkis.

Suppose that $E$ is a finite ground set and that $g(S, \lambda)$ is a function where $S \subseteq E$ and $\lambda$ is a scalar parameter. One example would be where $E$ is $N - \{s, t\}$ in a parametric max flow network with capacities $u_{ij}(\lambda)$, and $g(S, \lambda)$ is the value of cut $S + \{s\}$ w.r.t. $\lambda$.

We suppose that $g(S, \lambda)$ is submodular in $S$ for each fixed value of $\lambda$, and that it satisfies the following *Decreasing Differences* property for each $S \subseteq T$ and $\lambda' \geq \lambda$:

$$g(T, \lambda) - g(S, \lambda) \geq g(T, \lambda') - g(S, \lambda'). \tag{1}$$

**(a)** Prove that the following weaker version of (1) implies (1): For all $e \in E$, $S \subseteq E$, and $\lambda' \geq \lambda$,

$$g(S + e, \lambda) - g(S, \lambda) \geq g(S + e, \lambda') - g(S, \lambda'). \tag{2}$$

Enumerate $T - S$ as $\{e_1, e_2, \ldots, e_k\}$. Then using (2) repeatedly we get $g(T, \lambda) - g(S, \lambda) = (g(S + \{e_1, e_2, \ldots, e_k\}, \lambda) - g(S + \{e_1, e_2, \ldots, e_{k-1}\}, \lambda)) + (g(S + \{e_1, e_2, \ldots, e_{k-1}\}, \lambda) - g(S + \{e_1, e_2, \ldots, e_{k-2}\}, \lambda)) + \ldots + (g(S + \{e_1\}, \lambda) - g(S, \lambda)) \geq (g(S + \{e_1, e_2, \ldots, e_k\}, \lambda') - g(S + \{e_1, e_2, \ldots, e_{k-1}\}, \lambda')) + (g(S + \{e_1, e_2, \ldots, e_{k-1}\}, \lambda') - g(S + \{e_1, e_2, \ldots, e_{k-2}\}, \lambda')) + \ldots + (g(S + \{e_1\}, \lambda') - g(S, \lambda')) = g(T, \lambda') - g(S, \lambda')$.

**(b)** Suppose that $Q$ minimizes $g$ at $\lambda$ and $Q'$ minimizes $g$ at $\lambda'$. Prove that $Q \cap Q'$ also minimizes $g$ at $\lambda$, and $Q \cup Q'$ minimizes $g$ at $\lambda'$.

Using respectively the optimality of $Q$, submodularity of $g$, (1), and optimality of $Q'$ we get $0 \geq g(Q, \lambda) - g(Q \cap Q', \lambda) \geq g(Q \cup Q', \lambda) - g(Q', \lambda) \geq g(Q \cup Q', \lambda') - g(Q', \lambda') \geq 0$. Thus we get equality everywhere, and so we have that $g(Q, \lambda) = g(Q \cap Q', \lambda)$ (i.e., $Q \cap Q'$ is optimal for $\lambda$), and $g(Q, \lambda') = g(Q \cup Q', \lambda')$ (i.e., $Q \cup Q'$ is optimal for $\lambda'$).

**(c)** Prove that the parametric cut function is submodular and satisfies (1) when the $u_{ij}(\lambda)$ satisfy the GGT axioms.

We already know that the cut function is submodular for each fixed $\lambda$, so by (a) we need only verify (2). The only terms that do not cancel out are $u_{et}(\lambda)$ from $g(S + e, \lambda)$, $u_{se}(\lambda)$ from $g(S, \lambda)$, $u_{et}(\lambda')$ from $g(S + e, \lambda')$, and $u_{se}(\lambda')$ from $g(S, \lambda')$. The GGT axioms say that $u_{se}(\lambda') \geq u_{se}(\lambda)$ and $u_{et}(\lambda') \leq u_{et}(\lambda)$, or $u_{et}(\lambda) - u_{se}(\lambda) \geq u_{et}(\lambda') - u_{se}(\lambda')$, which is (2).

**(d)** It is well-known that the intersection and union of min cuts are again min cuts (if you don't know this, you should prove it for yourself). From this it follows that there is a unique minimal min cut $S_{\min}(\lambda)$ (i.e., the intersection of all min cuts) and maximal min cut $S_{\max}(\lambda)$

(i.e., the union of all min cuts) for each $\lambda$. Use (b) to give a non-algorithmic proof that if $\lambda \leq \lambda'$ that $S_{\min}(\lambda) \subseteq S_{\min}(\lambda')$ (and similarly for maximal min cuts), i.e., minimal/maximal min cuts are *nested* in $\lambda$.

Recall that $S_{\min}(\lambda)$ is the intersection of all min cuts for $\lambda$, and is the unique minimal min cut for $\lambda$. From (b), $S_{\min}(\lambda) \cap S_{\min}(\lambda')$ is also a min cut for $\lambda$, and it is a subset of $S_{\min}(\lambda)$, and so $S_{\min}(\lambda) \cap S_{\min}(\lambda') = S_{\min}(\lambda)$, implying that $S_{\min}(\lambda) \subseteq S_{\min}(\lambda')$ (and similarly for $S_{\max}(\lambda)$).

**(e)** It is tempting to think that if $Q$ is a min cut for $\lambda$ and $Q'$ is a min cut for $\lambda'$ with $\lambda' > \lambda$, that we must have that $Q \subseteq Q'$. Construct a counterexample to this conjecture.

Take any non-parametric max flow network with two non-nested min cuts $Q$ and $Q'$, and consider the capacities $u_{si}$ to be the parametric functions $u_{si} + 0 \cdot \lambda$. Then $Q$ is a min cut for any $\lambda$, and $Q'$ is a min cut for any $\lambda' > \lambda$, and they are not nested.

**(f)** Consider now the following *strict* version of (1): For $S \subset T$ and $\lambda < \lambda'$

$$g(T, \lambda) - g(S, \lambda) > g(T, \lambda') - g(S, \lambda'). \tag{3}$$

Prove that when (3) is true that if $Q$ is a min cut for $\lambda$ and $Q'$ is a min cut for $\lambda'$ with $\lambda' > \lambda$, that $Q \subseteq Q'$. (Thus with (3), *every* min cut for $\lambda$ is nested with *every* min cut for $\lambda'$.)

If $Q \not\subseteq Q'$, then $Q' \subset Q' \cup Q$ and so (3) applies to $S = Q'$ and $T = Q \cup Q'$. Then as in (b) we get $0 \geq g(Q, \lambda) - g(Q \cap Q', \lambda) \geq g(Q \cup Q', \lambda) - g(Q', \lambda) > g(Q \cup Q', \lambda') - g(Q', \lambda') \geq 0$, a contradiction. Hence we must have that $Q \subseteq Q'$. [Could further ask whether all GGT-type networks satisfy (3) (no they don't, as in (e)), and what further conditions would guarantee that a GGT-type network would satisfy (3) (e.g., having $u_{si} = a_i + b_i\lambda$ and $u_{it} = f_i - g_i\lambda$ with $\max\{b_i, g_i\} > 0$ for all $i$).]

**Question 9.** Let's try to generalize GGT Max Flow/Min Cut. Notice that the key observation that leads to being able to solve the parametric problem for the sequence of values $\lambda_1 < \lambda_2 < \ldots < \lambda_k$ in the same asymptotic time as a single max flow/min cut is this: Suppose that we have preflow $x^1$ and distance labels $d$ optimal for $\lambda_1$. Then we developed an $O(m)$ *Flow Update* subroutine which computes an initial flow $x'$ such that (1) $x'$ is feasible to $u(\lambda_2)$; (2) $x'$ is still a pre-flow; and (3) $d$ is still a valid labeling w.r.t. $x'$. Then we could continue Push-Relabel starting from $x'$ and $d$, and the same complexity analysis holds for the whole sequence as holds for a single max flow.

Here is another setting (due to Fleischer, coming from an application of flows over time) where the same trick can be used, but with non-GGT parametric capacities. We are given a parametric max flow/min cut network as usual, and a fixed cut $Q$ containing $s$ but not $t$. There is no arc $i \to j$ with $i \notin Q$ and $j \in Q$. Every arc $i \to j$ with $i, j \in Q$ has capacity $u_{ij}(\lambda) = a_{ij}\lambda$ for some $a_{ij} > 0$, and other capacities are constant. Note that as long as $Q \neq \{s\}$ this does not fall into the GGT class of parametric flow.

**(a)** Develop a Flow Update subroutine that runs in $O(m)$ time that achieves (1)–(3) above (and hence that shows that min cuts are again nested here, and can be computed in the same asymptotic time as a single min cut).

Consider the update $x'_{ij} = (\lambda_2/\lambda_2)x^1_{ij}$ for $i, j \in Q$, and $x'_{ij} = x^1_{ij}$ otherwise. It is clear that this satisfies (1). Due to the lack of arcs into $Q$, it is clear that (2) is also satisfied. Finally, since the flow update keeps saturated arcs saturated, and zero-flow arcs zero-flow, (3) is also satisfied.

**(b)** Prove or disprove (via constructing a counterexample) that the class of parametric flow problems considered here satisfies the Decreasing Differences property (1) of Question 8.

This class does *not* satisfy (1); see, e.g., the GMQT paper Figure 6/Example 4 for a counterexample.

**Question 10.** The point of this problem is that there are important parametric flow problems that do not fall into the GGT class, but that can be solved efficiently anyway.

Suppose that we are given a network $(N, A)$ with lower bounds $l$ and upper bounds $u$, and we are interested in whether there exists a feasible flow $x$ with $l \leq x \leq u$. Suppose that $S \subset N$ is a node subset. Then we define the *value* of $S$ to be $V(S) = l(\delta^-(S)) - u(\delta^+(S))$. Recall that Hoffman's Circulation Theorem (HCT) says that a feasible $x$ exists iff there does not exist any $S$ with $V(S) > 0$. Further recall that we can check whether a feasible $x$ exists by using a "Phase I Max Flow" subroutine, which produces some $S$ with $V(S) > 0$ when it cannot find a feasible flow.

Some strongly polynomial min-cost flow algorithms depend on computing *max mean cuts*: The mean value of a cut is its value divided by the number of arcs that cross the cut. Thus our numerator is $V(S) = l(\delta^-(S)) - u(\delta^+(S))$, and our denominator is $|\delta(S)|$. We want to compute $\mu^* = \max_S V(S)/|\delta(S)|$, and an associated max mean cut $S^*$ with $V(S^*)/|\delta(S^*)| = \mu^*$. We assume that there is at least one cut with $V(S) > 0$ (the data $l$, $u$ are infeasible), so that $\mu^* > 0$. For $\delta \geq 0$ we define $G(\delta)$ to be the network with data $l - \delta$ and $u + \delta$, i.e., where we relax the bounds by $\delta$, and so make it easier for the network to have a feasible flow.

**(a)** Define $V_\delta(S)$ to be the value of $S$ in $G(\delta)$. Prove that $V_\delta(S) = V(S) - \delta|\delta(S)|$.

We have $V_\delta(S) = (l - \delta)(\delta^-(S)) - ((u + \delta)(\delta^+(S)) + d(S)) = l(\delta^-(S)) - (u(\delta^+(S)) + d(S)) - \delta(|\delta^-(S)| + |\delta^+(S)|) = V(S) - \delta|\delta(S)|$.

**(b)** Define $\delta^*$ to be the smallest value of $\delta$ such that $G(\delta)$ is feasible. Prove that $\delta^* = \mu^*$.

If $\delta < \mu^*$, then $V_\delta(S^*) = V(S^*) - \delta|\delta(S^*)| = \mu^*|\delta(S^*)| - \delta|\delta(S^*)| > 0$, so $G(\delta)$ is infeasible, implying that $\delta^* > \delta$.

Now suppose that $\delta > \mu^*$ but that $G(\delta)$ is infeasible. Then there is some cut $T$ with $V_\delta(T) > 0$, or $V(T) - \delta|\delta(T)| > 0$, or $\delta < V(T)/|\delta(T)|$, contradicting that $\mu^*$ is the max mean cut value. Thus $G(\delta)$ must be feasible, and we are done.

**(c)** Suppose that we run Phase I Max Flow on $G(\delta)$, getting min cut $S(\delta)$. If we found that $G(\delta)$ was infeasible, then we recall from the proof of Hoffman's Circulation Theorem (HCT) that we have $V_\delta(S(\delta)) > 0$. Prove that in fact we have $V_\delta(S(\delta)) = \max_S V_\delta(S)$, i.e., that $S(\delta)$ is a *most positive cut* (MPC) in $G(\delta)$.

The proof of HCT shows that for any cut $S$, $V_\delta(S) = e(E) - \text{cap}(S)$. Since $e(E)$ is constant, maximizing $V_\delta(S)$ is equivalent to minimizing $\text{cap}(S)$, so a min cut of the max flow network is a most positive cut of $G(\delta)$.

Here is a natural algorithm based on (b) and (c):

This is the same algorithm we saw in Question 4, but in a non-GGT context (GGT can't be used here since there are parameters on all the arcs, not just the arcs at $s$ and $t$).

Define $\eta_i = |\delta(S_i)|$ and $v_i = V_{\delta_i}(S_i)$. Then Figure 2 illustrates how the algorithm works: Since $V_\delta(S_i) = V(S_i) - \delta\eta_i$, the graph of $V_\delta(S_i)$ as a function of $\delta$ is a line decreasing at slope $\eta_i$, and we choose $\delta_{i+1}$ as the point at which this line hits zero. This implies that $\delta_{i+1} - \delta_i = v_i/\eta_i$.

**(Discrete) Newton's Algorithm for Max Mean Cut**

Set $i = 0$, $\delta_0 = 0$ and run Phase I Max Flow on $G(\delta_0)$ getting MPC $S_0$.
While $V_{\delta_i}(S_i) > 0$
    Compute $\delta_{i+1} = V(S_i)/|\delta(S_i)|$, set $i \leftarrow i + 1$.
    Run Phase I Max Flow on $G(\delta_i)$ getting MPC $S_i$.
Now $\mu^* = \delta_i$ and $S_{i-1}$ is a max mean cut.

We want to analyze $k$, the number of iterations of this algorithm. [Note that this algorithm applies to any ratio problem where we can optimize a parametric objective.]

**(d)** Use (b) to prove that if the algorithm terminates, it does so with a correct answer for $\mu^*$ and $S^*$.

The algorithm terminates with a value $\delta_k$ such that $G(\delta_k)$ is feasible, so from (b) we know that $\mu^* \leq \delta_k$. But we also know that $\delta_k = V(S_{k-1})/|\delta(S_{k-1})|$ so we must have $\mu^* \geq \delta_k$, so $\delta_k = \mu^*$, and $\mu^* = V(S_{k-1})/|\delta(S_{k-1})|$, so $S_{k-1}$ is a max mean cut.

**(e)** Prove that for $0 \leq i < k$, we have $\frac{v_{i+1}}{v_i} + \frac{\eta_{i+1}}{\eta_i} \leq 1$. Hint: Use (c) and consider $S_{i+1}$ as a cut in $G(\delta_i)$.

By (c) $S_i$ is a MPC in $G(\delta_i)$ so we have $V_{\delta_i}(S_i) \geq V_{\delta_i}(S_{i+1})$. Now $l - \delta_{i+1} = l - (\delta_i + (\delta_{i+1} - \delta_i)) = l - (\delta_i + v_i/\eta_i)$ and similarly for $u$. Thus

$$\begin{aligned} v_i &= V_{\delta_i}(S_i) \geq V_{\delta_i}(S_{i+1}) \\ &= (l - \delta_i)(\delta^-(S_{i+1})) - ((u + \delta_i)(\delta^+(S_{i+1})) + d(S_{i+1})) \\ &= (l - \delta_{i+1})(\delta^-(S_{i+1})) - ((u + \delta_{i+1})(\delta^+(S_{i+1})) + d(S_{i+1})) + (v_i/\eta_i)\eta_{i+1} \\ &= v_{i+1} + (v_i/\eta_i)\eta_{i+1}. \end{aligned}$$

Re-arranging this gives $1 \geq v_{i+1}/v_i + \eta_{i+1}/\eta_i$, as desired.

**(f)** Define $U = \max_a \max\{|l_a|, |u_a|\}$. Use (e) to prove a bound of $O(\min(m, \log(nU)))$ on $k$.

Each iteration clearly strictly decreases $\eta_i$, so since $\eta_0 \leq m$ and $\eta_{k-1} \geq 1$ we must have $k \leq m$.

From (e), each iteration must either cut $v_i$ in half or $\eta_i$ in half. There can be at most $\log m$ iterations that cut $\eta_i$ in half. Each $v_i$ is a ratio of integers with numerator between 1 and $2mU$ and denominator between 1 and $m$, so $1/m \leq v_i \leq 2mU$, so that $v_i$ can be cut in half at most $O(\log(nU))$ times. [This analysis was first given by Ervolina and Mc.; a more careful accounting gives the better bound $k = O(\min(n, \frac{log(nU)}{1+\log\log(nU)-\log\log n}))$ (Radzik).]

# References


[1] Annotations in this sans-serif font

[2] Ahuja, R.K., Orlin, J.B., Stein, C., Tarjan, R.E. (1994). Improved algorithms for bipartite network flow. *SIAM J. Comput.*, **23**, 906–933.
Gives techniques for speeding up algorithms on bipartite networks that arise often in parametric applications.

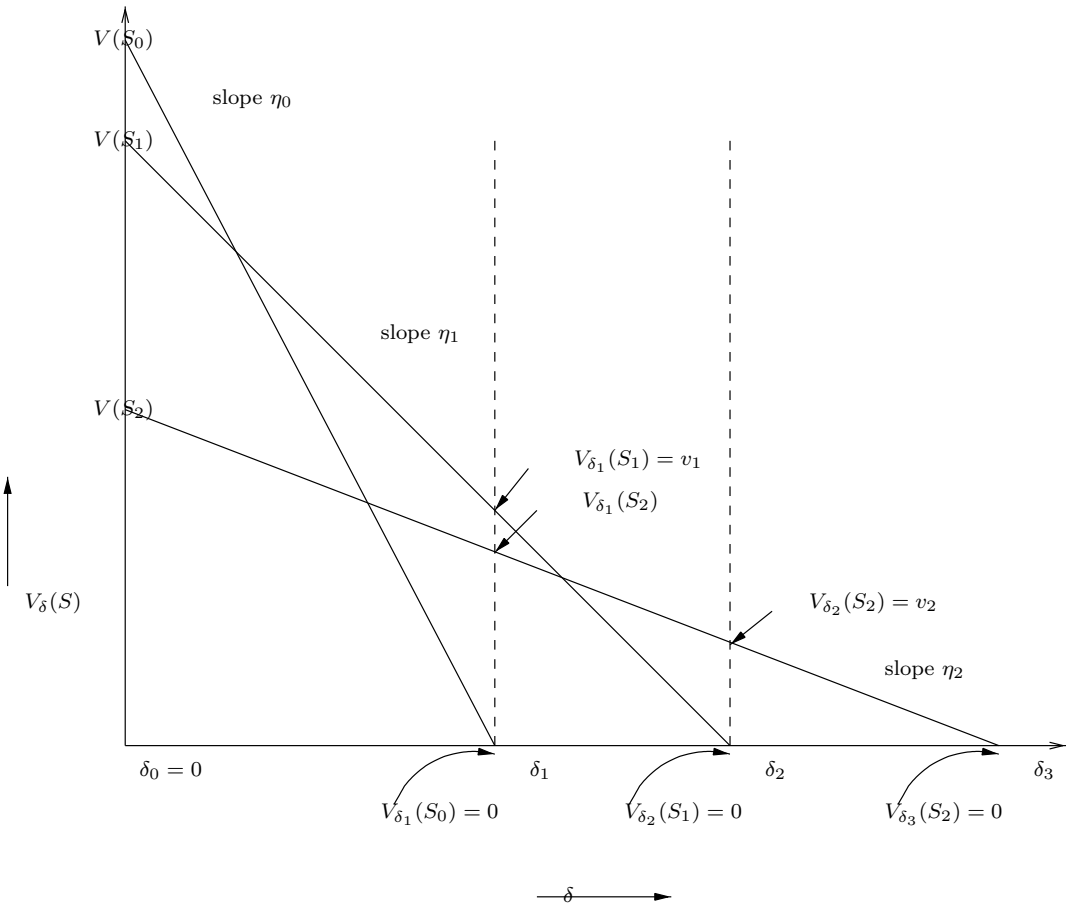


Figure 2: Picture of Newton's Algorithm


[3] Arai, T., S. Ueno, and Y. Kajitani (1993). Generalization of a Theorem on the Parametric Maximum Flow Problem. *Discrete Applied Mathematics*, **41**, 69–74.
Generalizes GGT to parameters at a fixed node.

[4] M.Babenko, J.Derryberry, A.V.Goldberg, R.E.Tarjan, and Y.Zhou (2007). Experimental Evaluation of Parametric Maximum Flow Algorithms, Experimental and Efficient Algorithms 6th International Workshop, WEA 2007 (C. Demetrescu, ed.), Lecture Notes in Computer Science, vol. 4525, Springer-Verlag, pp. 256–269.
Computationally compares GGT normal, simplified, and the Star Balancing algorithms.

[5] P. J. Carstensen (1983). Complexity of Some Parametric Integer and Network Programming Problems, *Mathematical Programming* **26**, 64–75.
Shows that general parametric max flow can have an exponential number of breakpoints.

[6] Chen, Y.L. (1994). Scheduling Jobs to Minimize Total Cost. *European J. of Operational Research*, **74**, 111–119.
Source of generalized GGT scheduling applications from [21].

[7] Dinkelbach, W. (1967). On Nonlinear Fractional Programming. *Management Science*, **13**, 492–498.
General technique for converting ratio optimization to parametric optimization.

[8] Eisner, M.J., Severance, D.G. (1976). Mathematical techniques for efficient record segmentation in large shared databases.
Early motivating application for GGT. *J. ACM*, **23**, 619–635.

[9] L. K. Fleischer (2001). Universally Maximum Flow with Piece-Wise Constant Capacity Functions, *Networks* **38**, 1–11.
Flows over time with non-GGT parametric application.

[10] Fleischer, L.K., Iwata, S. (2003). A Push-Relabel framework for submodular function minimization and applications to parametric optimization. *Discret. Appl. Math.*, **131**, 311–322.
A parametric application beyond max flow/min cut.

[11] Fujishige, S. (1991). *Submodular Functions and Optimization*, *Annals of Discrete Mathematics*, Vol. 47, North-Holland.
General reference for submodularity.

[12] Gallo, G., M.D. Grigoriadis, and R.E. Tarjan (1989). A Fast Parametric Maximum-Flow Algorithm and Applications. *SIAM J. Comput.* **18**, 30–55.
Basic paper showing how to solve parametric max flow/min cut faster for the GGT class.

[13] Goldberg, A.V., Rao, S. (1998). Beyond the flow decomposition barrier. *J. ACM*, **45**, 753–797.
Fastest current max flow algorithm in most cases, but one that seems not to GGT-ize.

[14] Goldberg, A.V., and R.E. Tarjan (1988). A New Approach to the Maximum Flow Problem. *JACM* 35, no. 4, 921–940.
The Push-Relabel max flow algorithm that most GGT algorithmic results are based on.

[15] F. Granot, S.T. McCormick, M.N. Queyranne, and F. Tardella (2011). Monotone parametric min cut revisited: structures and algorithms. To appear in *Math. Prog.*.
Contains many extensions of parametric min cut beyond the GGT class, more general than [21].

[16] Gusfield, D. and C. Martel (1992). A Fast Algorithm for the Generalized Parametric Minimum Cut Problem and Applications. *Algorithmica*, **7**, 499–519. Some extensions to GGT, notably to non-affine parametric capacities.

[17] Gusfield, D., Tardos, E. (1994). A faster parametric minimum-cut algorithm. *Algorithmica*, **11**, 278–290.
Shows GGT algorithmic result for Max Distance variant of Push-Relabel.

[18] Hochbaum, D.S. (2008). The pseudoflow algorithm: a new algorithm for the maximum flow problem. *Oper. Res.*, **58**, 992–1009.
Shows GGT algorithmic result for the same paper's Pseudoflow algorithm.

[19] Martel, C. (1989). A comparison of phase and nonphase network flow algorithms. *Networks*, **19**, 691–705.
Shows GGT algorithmic result for Dinic's Algorithm and Tarjan's Wave Algorithm.

[20] S.T. McCormick (1993). Approximate Binary Search Algorithms for Mean Cuts and Cycles. *OR Letters*, **14**, 129–132.
A faster way to do non-GGT parametric flow problem from [22].

[21] McCormick, S.T. (2000). Fast Algorithms for Parametric Scheduling come from Extensions to Parametric Maximum Flow. An extended abstract appears in *Proceedings of Symposium on the Theory of Computing* (1996) 319–328, full paper appears in *Operations Research*, **47**, 744–756.
Sits between [12] and [15] in considering parametric min cut problems beyond the GGT class with applications.

[22] McCormick, S.T., Ervolina, T.R. (1994). Computing maximum mean cuts. *Discret. Appl. Math.*, **52**, 53–70.
A non-GGT parametric flow problem solved via Discrete Newton.

[23] Megiddo, N. (1983). Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *JACM* **30**, 852–865.
A general way to adapt a non-parametric algorithm to solve a parametric problem.

[24] Milgrom, P., Shannon, C. (1994). Monotone comparative statics. *Econometrica*, **62**, 157–180.
How economists view GGT, with a more general version of Topkis' result.

[25] Radzik, T. (1992). Newton's Method for Fractional Combinatorial Optimization. *Proc. 33rd IEEE Annual Symp. of Foundations of Computer Science*, 659–669; see also "Parametric Flows, Weighted Means of Cuts, and Fractional Combinatorial Optimization" in *Complexity in Numerical Optimization*, World Scientific (1993), ed. by P. Pardalos, 351–386.
An improved analysis of the Discrete Newton algorithm from [22].

[26] Rhys, J.M.W. (1970). A selection problem of shared fixed costs and network flows. *Manag. Sci.*, **17**, 200–207.
An early motivating application for GGT.

[27] Scutellà, M.G. (2007). A note on the parametric maximum flow problem and some related reoptimization issues. *Ann. Oper. Res.*, **150**, 231–244.
Generalizes [3].

[28] Stone, H.S. (1978). Critical load factors in two-processor distributed systems. *IEEE Trans. Softw. Eng.*, **4**, 254–258.
An early motivating example for GGT.

[29] Serafini, P. (1996). Scheduling Jobs on Several Machines with the Job Splitting Property. *Operations Research*, **44**, 617–628.
A motivating example for [21].

[30] R. Tarjan, J. Ward,B. Zhang, Y. Zhou, and J. Mao (2006). Balancing applied to maximum network flow problems. In *Proc. ESA*, LNCS 4168, 612–623.
The Star Balancing Algorithm for parametric min cut, and shows how to adapt any max flow algorithm to solve a subclass of GGT parametric min cut in $O(\log(nU))$ iterations.

[31] Topkis, Donald M. (1978). Minimizing a Submodular Function on a Lattice. *Operations Research*, **26**, 305–321.
A general parametric optimization framework including GGT as a special case.

[32] Topkis, D.M. (1998). *Supermodularity and Complementarity*. Princeton University Press, Princeton.
The book version of [31] with an economics slant.

[33] D.H. Topkis and A.F. Veinott, Jr. (1973). Monotone Solutions of Extremal Problems on Lattices, Eighth International Symposium on Mathematical Programming (Stanford University), p. 131.
Earlier version of [31].

[34] J. Ward, B. Zhang, S. Jain, C. Fry, T. Olavson, H. Mishal, J. Amaral, D. Beyer, A. Brecht, B. Cargille, R. Chadinha, K. Chou, G. DeNyse, Q. Feng, C. Padovani, S. Raj, K. Sunderbruch, R. Tarja n, K. Venkatraman, J. Woods, and J. Zhou (2010). HP Transforms Product Portfolio Management with Operations Research", *Interfaces* **40**, 17–32.
Motivating application for Star Balancing Algorithm of [30].