Combinatorial Algorithms to Solve Network Interdiction and Scheduling Problems with Multiple Parameters

S.T. McCormick; GP Oriolo; B. Peis

Sauder School of Business, UBC; U. Rome; TU Berlin

JPOC 2013

Combinatorial Algorithms to Solve Network Interdiction and Scheduling Problems with Multiple Parameters

S.T. McCormick; GP Oriolo; B. Peis

Sauder School of Business, UBC; U. Rome; TU Berlin

JPOC 2013

S. Thomas McCormick
Sauder School of Business

University of British Columbia

Combinatorial Algorithms to Solve Network Interdiction and Scheduling Problems with Multiple Parameters

S.T. McCormick; GP Oriolo; B. Peis

Sauder School of Business, UBC; U. Rome; TU Berlin

JPOC 2013

S. Thomas McCormick

Sauder School of Business

The best research b-school in Canada!

University of British Columbia

Outline

1. Network Interdiction
   - What is it?
   - Interdiction curves

Outline

1. Network Interdiction
   - What is it?
   - Interdiction curves
2. LP Duality
   - Dual of interdiction

Outline

Outline

1. Network Interdiction
   - What is it?
   - Interdiction curves
2. LP Duality
   - Dual of interdiction
3. Parametric Min Cut
   - Parametric curves
4. The Breakpoint Subproblem
   - What is it?
   - Algorithms
   - Discrete Newton

Outline

Outline

What is Network Interdiction?

- We start with an ordinary max flow min cut network with source s, sink t, and capacities c. The capacity of cut S is $\mathrm{cap}_c(S)$.

What is Network Interdiction?

- We start with an ordinary max flow min cut network with source s, sink t, and capacities c. The capacity of cut S is $\mathrm{cap}_c(S)$.
- We have a second non-negative datum on each arc: r_{ij} is the removal cost of destroying arc $i \to j$; we could spend, e.g., $r_{ij}/2$ to reduce the capacity of $i \to j$ to $c_{ij}/2$.

What is Network Interdiction?

- We start with an ordinary max flow min cut network with source s, sink t, and capacities c. The capacity of cut S is $\mathrm{cap}_c(S)$.
- We have a second non-negative datum on each arc: r_{ij} is the removal cost of destroying arc $i \to j$; we could spend, e.g., $r_{ij}/2$ to reduce the capacity of $i \to j$ to $c_{ij}/2$.
  - In Min Cut we assume that the removal cost of $i \to j$ is proportional to its capacity c_{ij}, but here removal cost is independent of c_{ij}.

What is Network Interdiction?

- We start with an ordinary max flow min cut network with source s, sink t, and capacities c. The capacity of cut S is $\mathrm{cap}_c(S)$.
- We have a second non-negative datum on each arc: r_{ij} is the removal cost of destroying arc $i \to j$; we could spend, e.g., $r_{ij}/2$ to reduce the capacity of $i \to j$ to $c_{ij}/2$.
  - In Min Cut we assume that the removal cost of $i \to j$ is proportional to its capacity c_{ij}, but here removal cost is independent of c_{ij}.
- Finally, we have a budget $B \geq 0$ to spend on destroying arcs. Our objective is to spend at most B (maybe fractionally) in a way that minimizes the value of the residual flow.

What is Network Interdiction?

- We start with an ordinary max flow min cut network with source s, sink t, and capacities c. The capacity of cut S is $\mathrm{cap}_c(S)$.
- We have a second non-negative datum on each arc: r_{ij} is the removal cost of destroying arc $i \to j$; we could spend, e.g., $r_{ij}/2$ to reduce the capacity of $i \to j$ to $c_{ij}/2$.
  - In Min Cut we assume that the removal cost of $i \to j$ is proportional to its capacity c_{ij}, but here removal cost is independent of c_{ij}.
- Finally, we have a budget $B \geq 0$ to spend on destroying arcs. Our objective is to spend at most B (maybe fractionally) in a way that minimizes the value of the residual flow.
  - In Min Cut we remove arcs until there is zero flow left, but here we remove only as much as we can under the budget.

Removing arcs greedily

- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.

Removing arcs greedily

- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.
- Some thought shows that it is always optimal to destroy arcs belonging to some cut S (that may depend on B).

Removing arcs greedily

- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.
- Some thought shows that it is always optimal to destroy arcs belonging to some cut S (that may depend on B).
  - Proof: if the removed arcs do not belong to a single cut, we could move an arc to a cut to remove more flow.

Removing arcs greedily

- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.
- Some thought shows that it is always optimal to destroy arcs belonging to some cut S (that may depend on B).
  - Proof: if the removed arcs do not belong to a single cut, we could move an arc to a cut to remove more flow.
- Further thought reveals that we should destroy arcs of S greedily, from the max value of $\rho_e = c_e/r_e$ down to the minimum value: "bang for the buck".

Removing arcs greedily

- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.
- Some thought shows that it is always optimal to destroy arcs belonging to some cut S (that may depend on B).
  - Proof: if the removed arcs do not belong to a single cut, we could move an arc to a cut to remove more flow.
- Further thought reveals that we should destroy arcs of S greedily, from the max value of $\rho_e = c_e/r_e$ down to the minimum value: "bang for the buck".
  - Proof: again we could use a pairwise interchange argument.

Removing arcs greedily

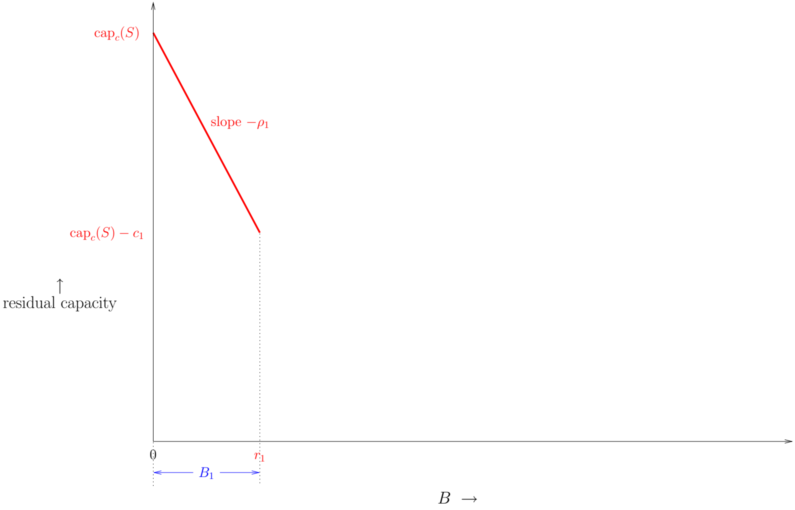
- Thus if $B = 0$, then the interdiction value is cap_c^*, the ordinary min cut value; for $B \geq \mathrm{cap}_r^*$, the interdiction value is 0.
- Some thought shows that it is always optimal to destroy arcs belonging to some cut S (that may depend on B).
  - Proof: if the removed arcs do not belong to a single cut, we could move an arc to a cut to remove more flow.
- Further thought reveals that we should destroy arcs of S greedily, from the max value of $\rho_e = c_e / r_e$ down to the minimum value: "bang for the buck".
  - Proof: again we could use a pairwise interchange argument.
- So let's get some idea of how much flow we can remove by destroying arcs from a fixed cut S.

The interdiction curve for a fixed cut S

Assume that we concentrate all our destruction on arcs of S.

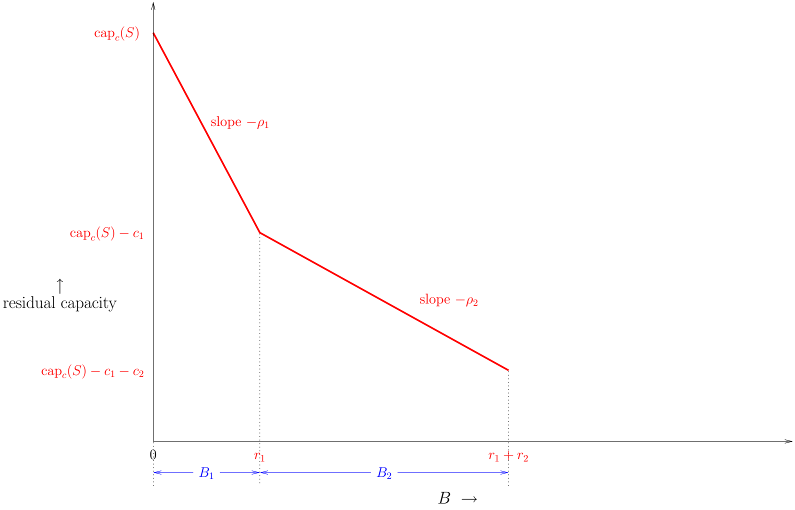
The interdiction curve for a fixed cut S

Assume that we concentrate all our destruction on arcs of S.



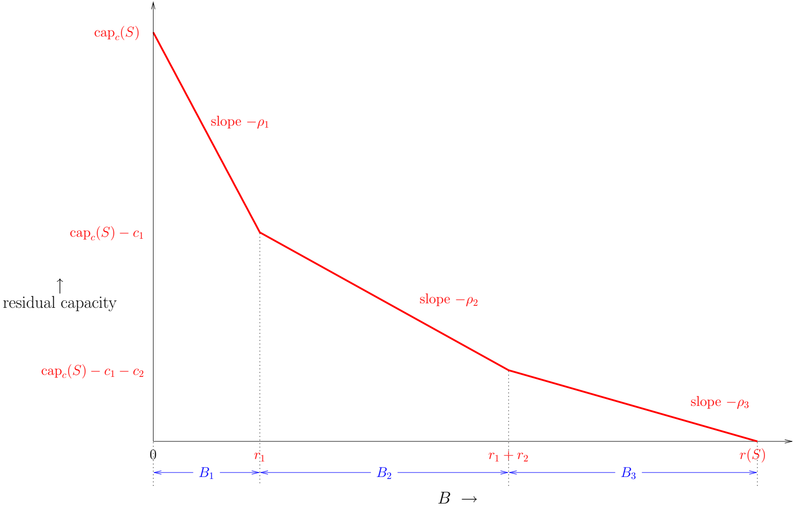
The interdiction curve for a fixed cut S

Assume that we concentrate all our destruction on arcs of S.



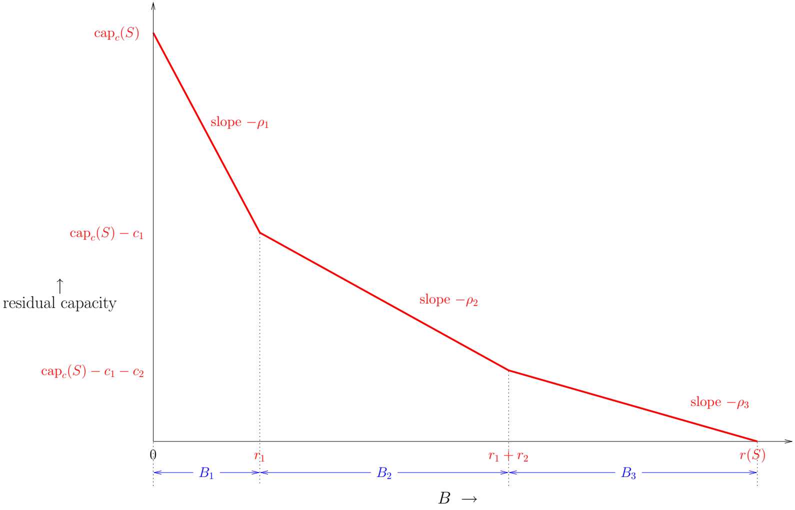
The interdiction curve for a fixed cut S

Assume that we concentrate all our destruction on arcs of S.



The interdiction curve for a fixed cut S

This curve is piecewise linear convex.

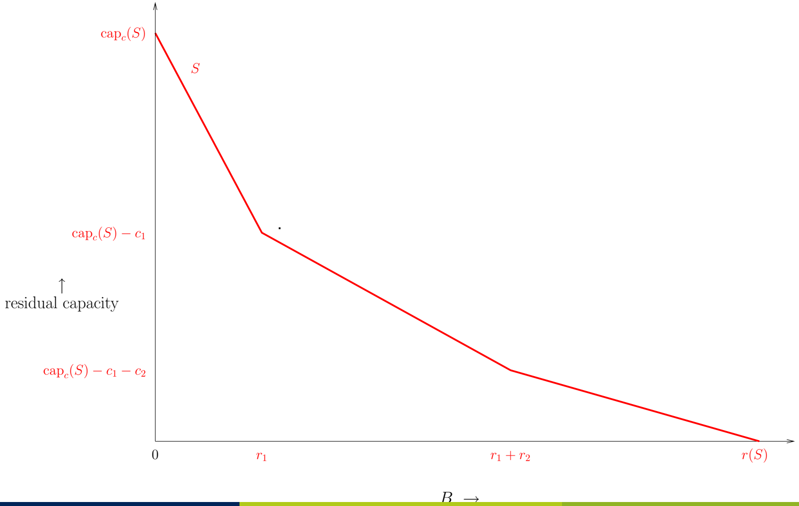


The overall interdiction curve: the B-profile

We overlay the cut-wise interdiction curves to get the overall curve.

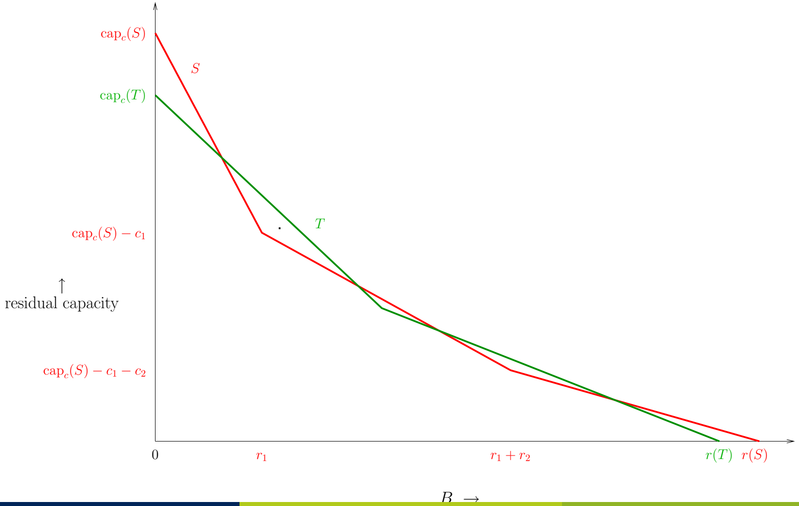
The overall interdiction curve: the B-profile

We overlay the cut-wise interdiction curves to get the overall curve.



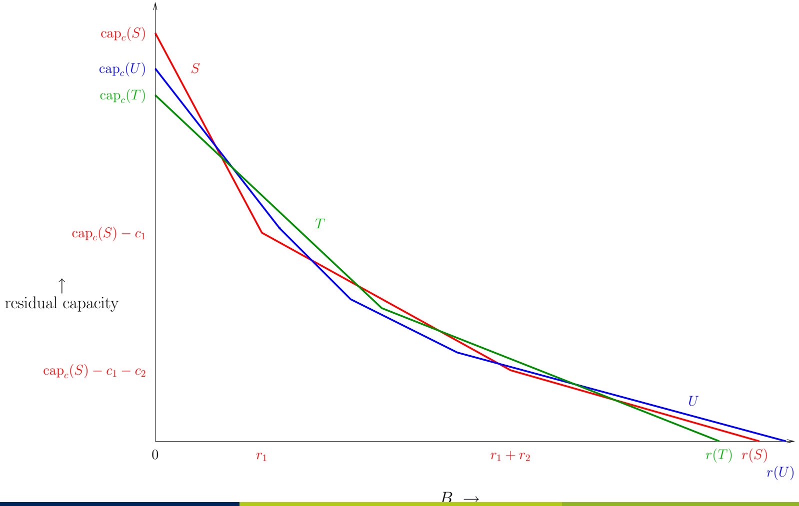
The overall interdiction curve: the B-profile

We overlay the cut-wise interdiction curves to get the overall curve.



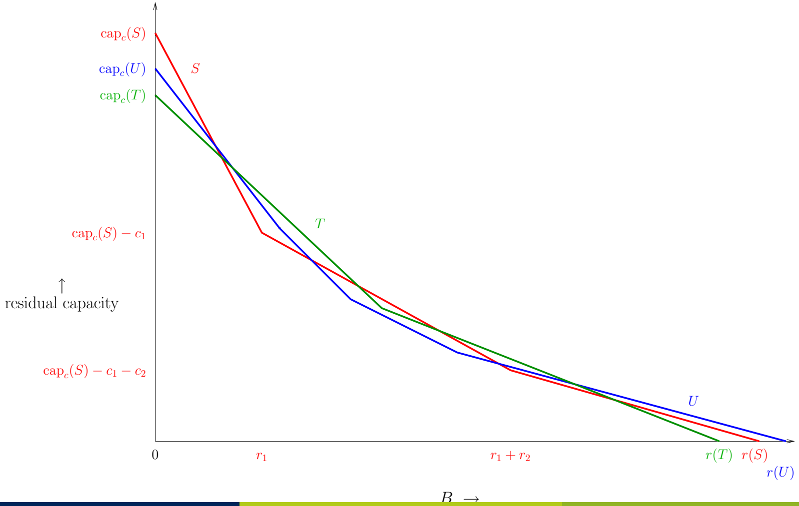
The overall interdiction curve: the B-profile

We overlay the cut-wise interdiction curves to get the overall curve.



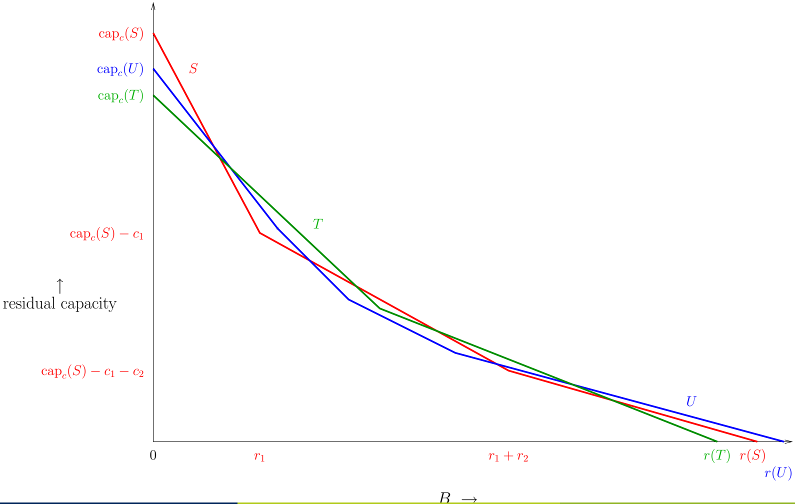
The overall interdiction curve: the B-profile

For a given value of B, we just select which S gives the minimum value at B, so the overall curve is the minimum of all the cut-wise curves.



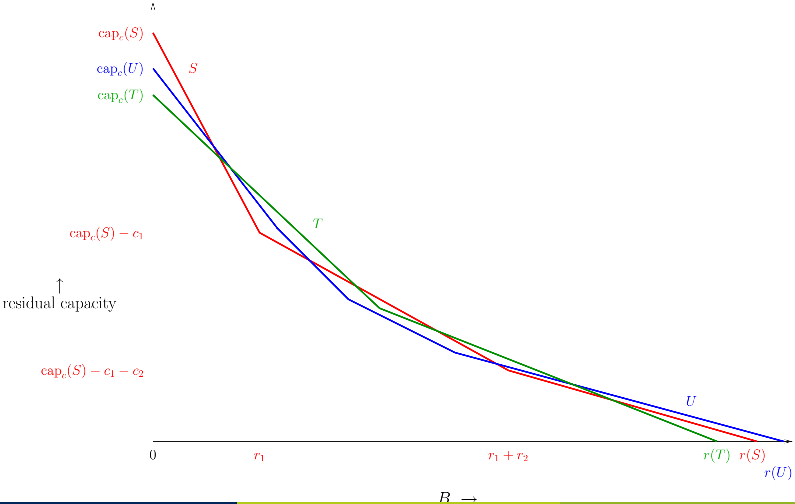
The overall interdiction curve: the B-profile

Unfortunately, the minimum of a bunch of convex curves is not in general convex.



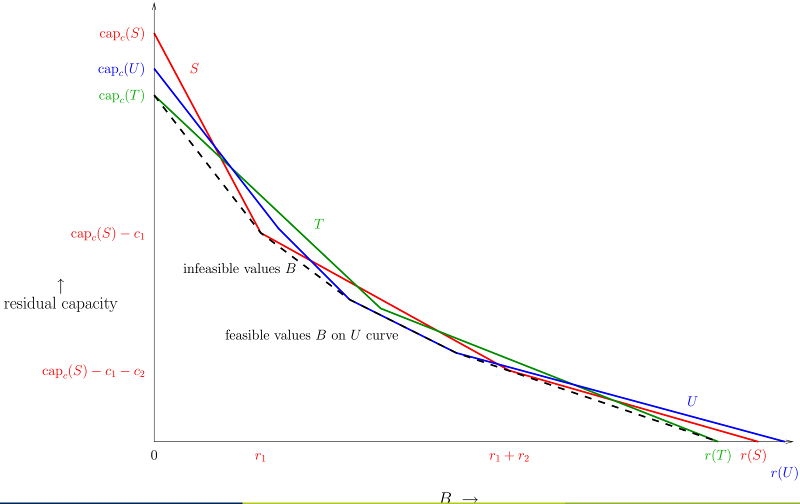
The overall interdiction curve: the B-profile

This is why Network Interdiction is NP Hard (Phillips '93; Wood '93).



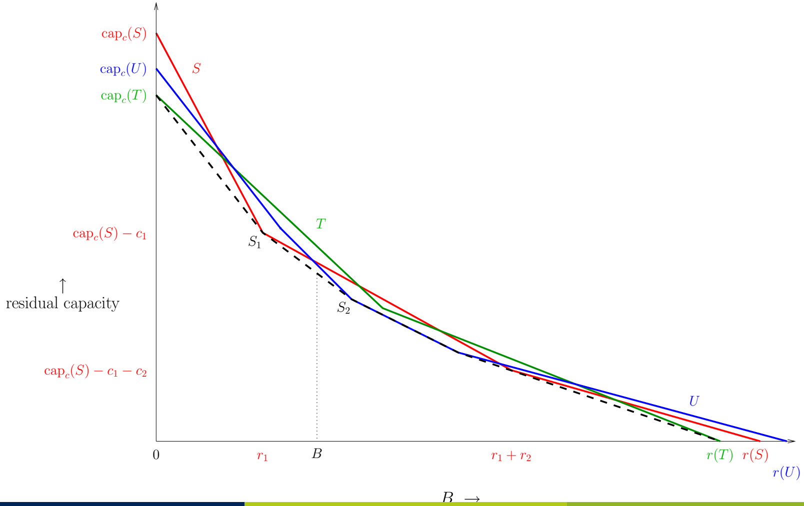
The overall interdiction curve: the B-profile

If we take the lower envelope, or convex hull, of the overall interdiction curve, we get something tractable, the B-profile.



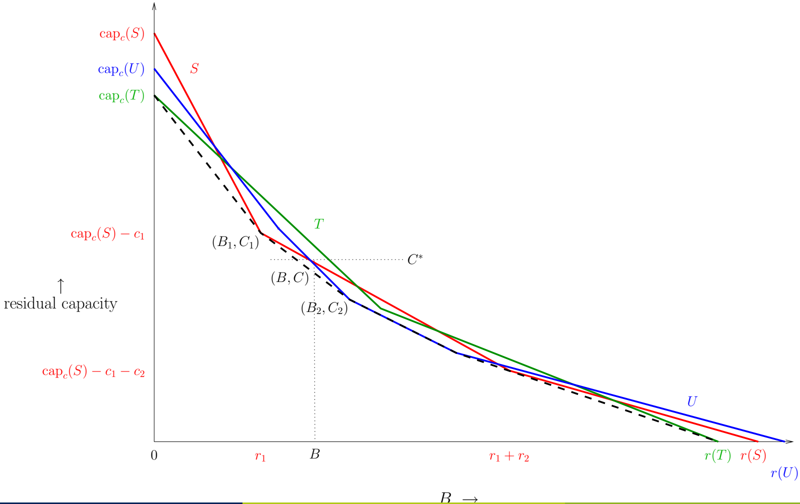
The overall interdiction curve: the B-profile

Now budget B corresponds to a convex combination of points coming from the interdiction curves of (one or) two cuts, S_1 and S_2.



The overall interdiction curve: the B-profile

S_1 corresponds to breakpoint (B_1, C_1), S_2 to (B_2, C_2), and we have λ s.t. $B = \lambda_1 B_1 + \lambda_2 B_2$; define $C = \lambda_1 C_1 + \lambda_2 C_2 \leq C^* =$ opt. resid. capacity.



Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then . . .

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
    - Choose some $\epsilon > 0$; then ...
    - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \leq 1 + \epsilon$.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then ...
  - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \leq 1 + \epsilon$.
  - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
    - Choose some $\epsilon > 0$; then ...
    - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \leq 1 + \epsilon$.
    - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$.
    - Suppose that $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$. Then $B_1 \leq B$ and so $\epsilon\frac{C_1}{C} \leq 1 + \epsilon$, or $C_1 \leq (1 + 1/\epsilon)C \leq (1 + 1/\epsilon)C^*$.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then ...
  - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \le 1 + \epsilon$.
  - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \le 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \le 1 + \epsilon$.
  - Suppose that $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \le 1 + \epsilon$. Then $B_1 \le B$ and so $\epsilon\frac{C_1}{C} \le 1 + \epsilon$, or $C_1 \le (1 + 1/\epsilon)C \le (1 + 1/\epsilon)C^*$.
  - If instead $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \le 1 + \epsilon$, then $C_2 \le C \le C^*$ and so $\frac{B_2}{B} \le 1 + \epsilon$, or $B_2 \le (1 + \epsilon)B$.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then ...
  - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \le 1 + \epsilon$.
  - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \le 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \le 1 + \epsilon$.
  - Suppose that $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \le 1 + \epsilon$. Then $B_1 \le B$ and so $\epsilon\frac{C_1}{C} \le 1 + \epsilon$, or $C_1 \le (1 + 1/\epsilon)C \le (1 + 1/\epsilon)C^*$.
  - If instead $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \le 1 + \epsilon$, then $C_2 \le C \le C^*$ and so $\frac{B_2}{B} \le 1 + \epsilon$, or $B_2 \le (1 + \epsilon)B$.
  - Thus we can choose S_1 and under-use the budget but have a factor $1 + 1/\epsilon$ too much residual capacity, or choose S_2 and have less than C^* residual capacity, but overrun the budget by a factor of $1 + \epsilon$.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then ...
  - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \leq 1 + \epsilon$.
  - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$.
  - Suppose that $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$. Then $B_1 \leq B$ and so $\epsilon\frac{C_1}{C} \leq 1 + \epsilon$, or $C_1 \leq (1 + 1/\epsilon)C \leq (1 + 1/\epsilon)C^*$.
  - If instead $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$, then $C_2 \leq C \leq C^*$ and so $\frac{B_2}{B} \leq 1 + \epsilon$, or $B_2 \leq (1 + \epsilon)B$.
  - Thus we can choose S_1 and under-use the budget but have a factor $1 + 1/\epsilon$ too much residual capacity, or choose S_2 and have less than C^* residual capacity, but overrun the budget by a factor of $1 + \epsilon$.
- The algorithmic question is then: Given B, how do we find S_1 and S_2? This shows that we also want B_1, B_2, C_1 and C_2.

Linearizing the overall curve: the B-profile

- Burch et al '02 show that we can use S_1 and S_2 to get a *pseudo-approximation* algorithm for Network Interdiction:
  - Choose some $\epsilon > 0$; then ...
  - $\lambda_1\left(\frac{B_1}{B} + \epsilon\frac{C_1}{C}\right) + \lambda_2\left(\frac{B_2}{B} + \epsilon\frac{C_2}{C}\right) = \frac{\lambda_1 B_1 + \lambda_2 B_2}{B} + \epsilon\frac{\lambda_1 C_1 + \lambda_2 C_2}{C} \leq 1 + \epsilon$.
  - Thus either $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$ or $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$.
  - Suppose that $\frac{B_1}{B} + \epsilon\frac{C_1}{C} \leq 1 + \epsilon$. Then $B_1 \leq B$ and so $\epsilon\frac{C_1}{C} \leq 1 + \epsilon$, or $C_1 \leq (1 + 1/\epsilon)C \leq (1 + 1/\epsilon)C^*$.
  - If instead $\frac{B_2}{B} + \epsilon\frac{C_2}{C} \leq 1 + \epsilon$, then $C_2 \leq C \leq C^*$ and so $\frac{B_2}{B} \leq 1 + \epsilon$, or $B_2 \leq (1 + \epsilon)B$.
  - Thus we can choose S_1 and under-use the budget but have a factor $1 + 1/\epsilon$ too much residual capacity, or choose S_2 and have less than C^* residual capacity, but overrun the budget by a factor of $1 + \epsilon$.
- The algorithmic question is then: Given B, how do we find S_1 and S_2? This shows that we also want B_1, B_2, C_1 and C_2.
- Burch et al write a linear program that can do it, but here we want a combinatorial algorithm to do it.

Outline

The linear program and its dual

- The normal min cut dual LP is

$$\begin{array}{rcll} \min \sum_{u \to v} c_{uv} y_{uv} & & & \\ \text{s.t. } d_u - d_v + y_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\ d_t - d_s + y_{ts} & \geq & 1 & \\ y_{uv} & \geq & 0 & \text{all } u \to v. \end{array}$$

The linear program and its dual

- The normal min cut dual LP is

$$\begin{array}{rcll} \min \sum_{u \to v} c_{uv} y_{uv} & & & \\ \text{s.t. } d_u - d_v + y_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\ d_t - d_s + y_{ts} & \geq & 1 & \\ y_{uv} & \geq & 0 & \text{all } u \to v. \end{array}$$

- To get an interdiction version, put a second dual variable z_{uv} on each $u \to v$ that represents what fraction of $u \to v$ we are going to destroy.

The linear program and its dual

- The normal min cut dual LP is

$$\begin{array}{rcll} \min \sum_{u\to v} c_{uv} y_{uv} & & & \\ \text{s.t. } d_u - d_v + y_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\ d_t - d_s + y_{ts} & \geq & 1 & \\ y_{uv} & \geq & 0 & \text{all } u \to v. \end{array}$$

- To get an interdiction version, put a second dual variable z_{uv} on each $u \to v$ that represents what fraction of $u \to v$ we are going to destroy.
- The new LP is then

$$\begin{array}{rcll} \min \sum_{u\to v} c_{uv} y_{uv} & & & \\ \text{s.t. } d_u - d_v + y_{uv} + z_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\ d_t - d_s + y_{ts} & \geq & 1 & \\ \sum_{u\to v} r_{uv} z_{uv} & \leq & B & \\ y_{uv},\ z_{uv} & \geq & 0 & \text{all } u \to v. \end{array}$$

The linear program and its dual

- Repeat the new LP with dual variables:

$$
\begin{array}{rrcll}
 & \min \sum_{u\to v} c_{uv} y_{uv} & & & \\
x_{uv}: & \text{s.t. } d_u - d_v + y_{uv} + z_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\
x_{ts:}: & d_t - d_s + y_{ts} & \geq & 1 & \\
\lambda: & \sum_{u\to v} r_{uv} z_{uv} & \leq & B & \\
 & y_{uv},\ z_{uv} & \geq & 0 & \text{all } u \to v.
\end{array}
$$

The linear program and its dual

- Repeat the new LP with dual variables:

$$
\begin{array}{rrcll}
 & \min \sum_{u\to v} c_{uv} y_{uv} & & & \\
x_{uv}: & \text{s.t. } d_u - d_v + y_{uv} + z_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\
x_{ts:}: & d_t - d_s + y_{ts} & \geq & 1 & \\
\lambda: & \sum_{u\to v} r_{uv} z_{uv} & \leq & B & \\
 & y_{uv},\ z_{uv} & \geq & 0 & \text{all } u \to v.
\end{array}
$$

- When we "primalize" this interdiction dual LP we get new primal variable λ corresponding to the dual constraint $\sum_{u\to v} r_{uv} z_{uv} \leq B$, and the z_{uv}'s give us a second set of capacities.

The linear program and its dual

- Repeat the new LP with dual variables:

$$
\begin{array}{rrcll}
 & \min \sum_{u\to v} c_{uv} y_{uv} & & & \\
x_{uv}: & \text{s.t. } d_u - d_v + y_{uv} + z_{uv} & \geq & 0 & \text{for } u \to v \neq t \to s, \\
x_{ts:}: & d_t - d_s + y_{ts} & \geq & 1 & \\
\lambda: & \sum_{u\to v} r_{uv} z_{uv} & \leq & B & \\
 & y_{uv},\ z_{uv} & \geq & 0 & \text{all } u \to v.
\end{array}
$$

- When we "primalize" this interdiction dual LP we get new primal variable λ corresponding to the dual constraint $\sum_{u\to v} r_{uv} z_{uv} \leq B$, and the z_{uv}'s give us a second set of capacities.
- The primal interdiction LP is

$$
\begin{array}{rrcl}
 & \max_{x,\lambda} \ (x_{ts} - B\lambda) & & \\
d: & \text{s.t. conservation} & & \\
y_{uv}: & 0 \leq x_{uv} & \leq & c_{uv} \\
z_{uv}: & x_{uv} - r_{uv}\lambda & \leq & 0.
\end{array}
$$

The linear program and its dual

- Repeat the primal interdiction LP and highlight the two capacities:

$$
\begin{aligned}
\max_{x,\lambda}\ (x_{ts} - B\lambda) \\
\text{s.t. conservation} \\
0 \le x_{uv} &\le c_{uv} \\
x_{uv} - r_{uv}\lambda &\le 0.
\end{aligned}
$$

The linear program and its dual

- Repeat the primal interdiction LP and highlight the two capacities:

$$\begin{array}{rcl} \max_{x,\lambda}\ (x_{ts} - B\lambda) & & \\ \text{s.t. conservation} & & \\ 0 \le x_{uv} & \le & c_{uv} \\ x_{uv} - r_{uv}\lambda & \le & 0. \end{array}$$

- The two capacity constraints simplify into

$$x_{uv} \le \min(c_{uv}, \lambda r_{uv}),$$

a *parametric capacity* in the scalar parameter λ.

The linear program and its dual

- Repeat the primal interdiction LP and highlight the two capacities:

$$\begin{aligned} \max_{x,\lambda}\ (x_{ts} - B\lambda) & \\ \text{s.t. conservation} & \\ 0 \le x_{uv} & \le c_{uv} \\ x_{uv} - r_{uv}\lambda & \le 0. \end{aligned}$$

- The two capacity constraints simplify into

$$x_{uv} \le \min(c_{uv}, \lambda r_{uv}),$$

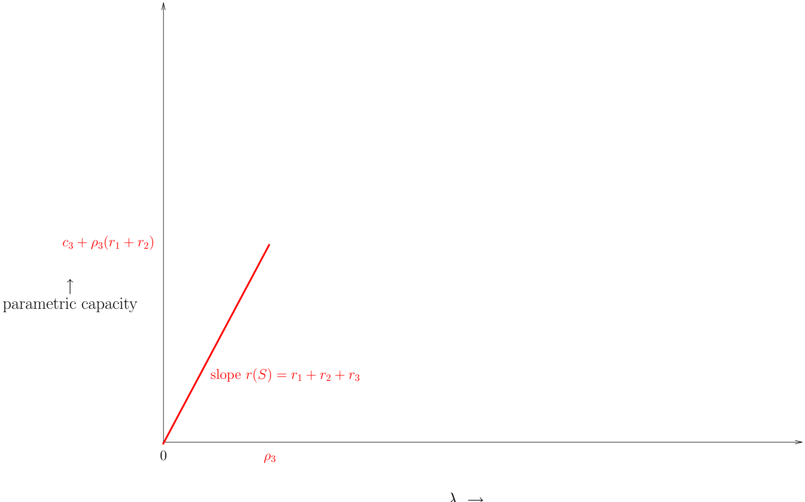
  a *parametric capacity* in the scalar parameter λ.

- So let's investigate the behavior of this parametric min cut problem.

Outline

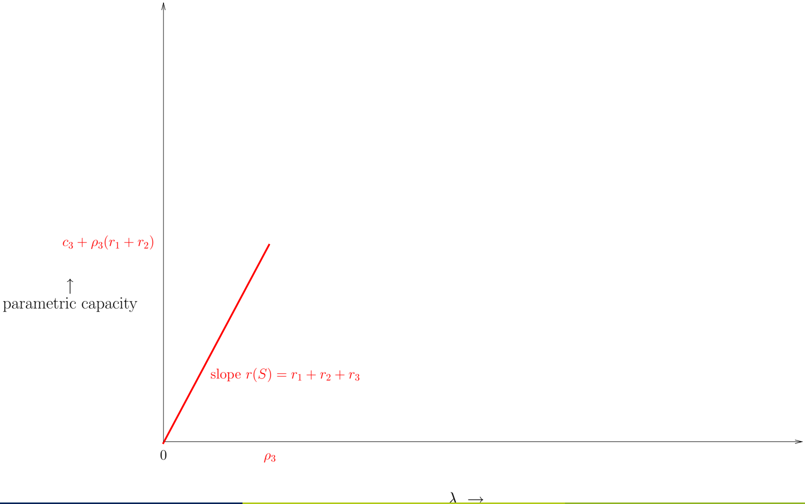
Parametric capacity of fixed cut S

When λ is small, $\mathrm{cap}(S, \lambda) = \lambda \mathrm{cap}_r(S)$.



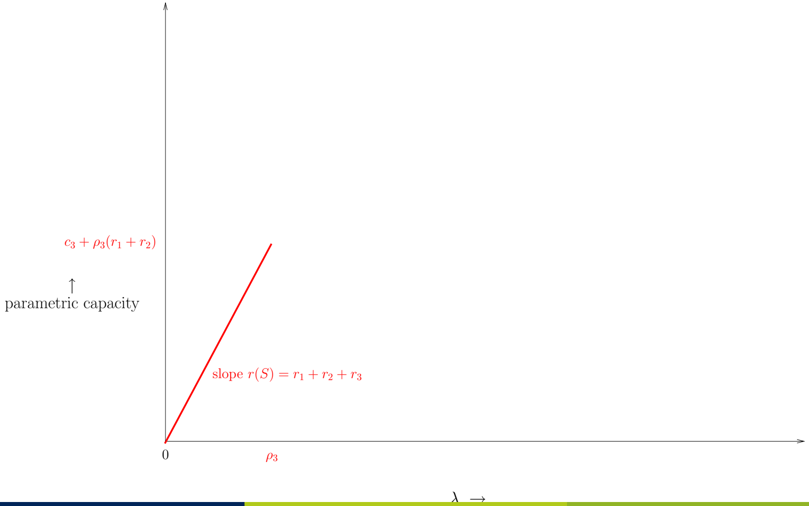
Parametric capacity of fixed cut S

This continues as long as $\lambda r_{uv} \leq c_{uv}$ for all $u \to v \in \delta^+(S)$, or $\lambda \leq \rho_{uv}$.



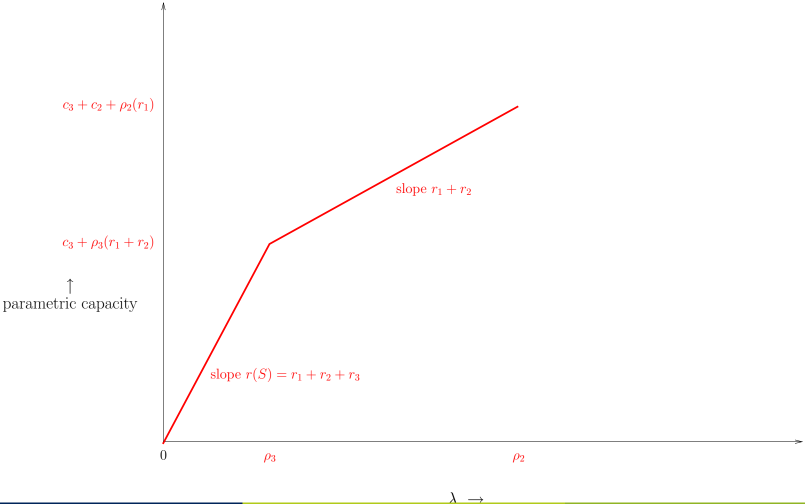
Parametric capacity of fixed cut S

Thus the first breakpoint is when λ hits $\min_{\delta^+(S)} \rho_{uv}$.



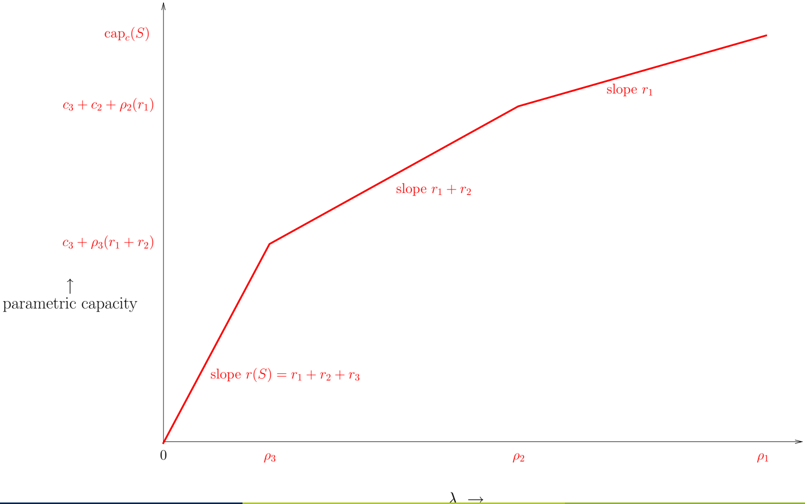
Parametric capacity of fixed cut S

Thus the first breakpoint is when λ hits $\min_{\delta^+(S)} \rho_{uv}$.



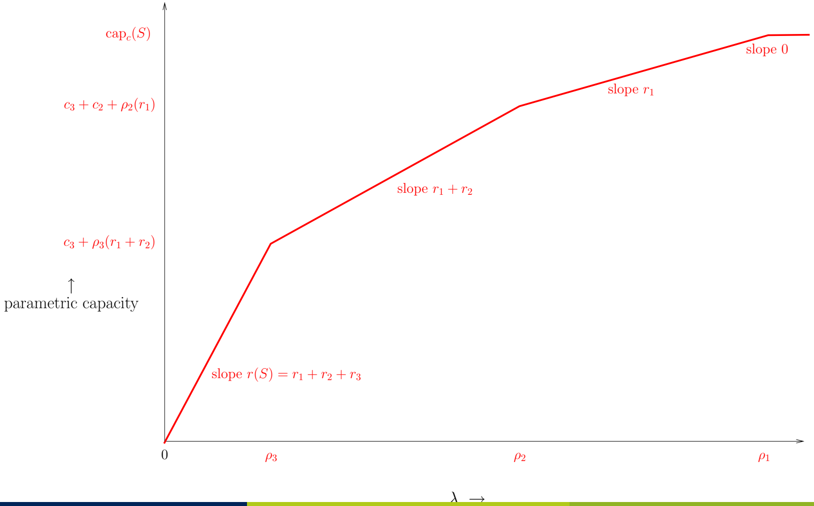
Parametric capacity of fixed cut S

Thus the first breakpoint is when λ hits $\min_{\delta^+(S)} \rho_{uv}$.



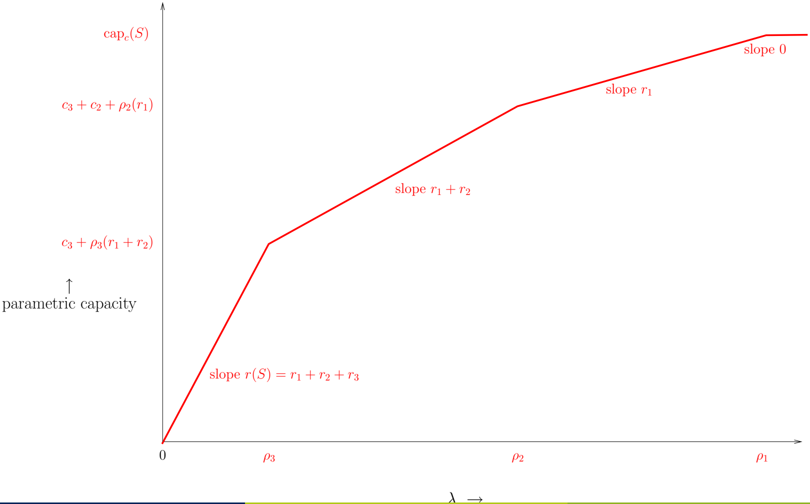
Parametric capacity of fixed cut S

Thus the first breakpoint is when λ hits $\min_{\delta^+(S)} \rho_{uv}$.



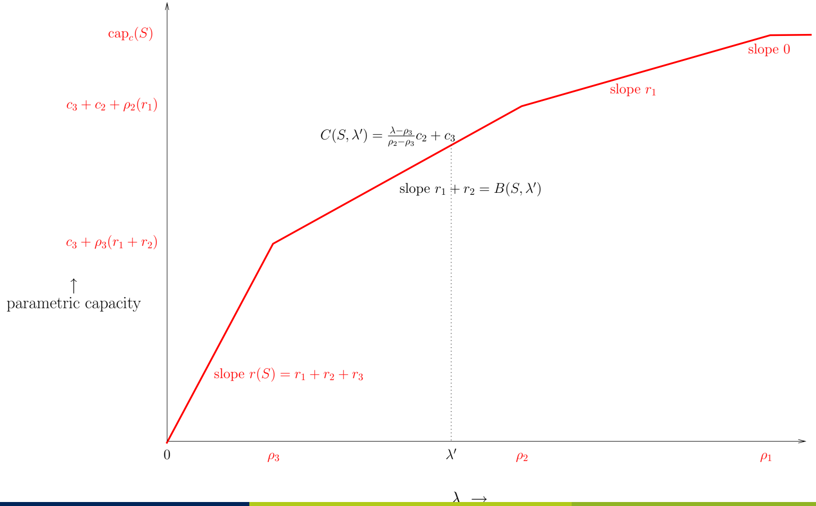
Parametric capacity of fixed cut S

The parametric capacity curve for S is piecewise linear concave.



Parametric capacity of fixed cut S

For a value λ' of λ we also get the local budget $B(S, \lambda')$ and local residual capacity $C(S, \lambda')$.



Conjugate duality between interdiction and parametric capacity for S

- Both curves are piecewise linear; the interdiction curve (residual capacity curve) is convex, the parametric capacity curve is concave.

Conjugate duality between interdiction and parametric capacity for S

- Both curves are piecewise linear; the interdiction curve (residual capacity curve) is convex, the parametric capacity curve is concave.
- Index the arcs of $\delta^+(S)$ in descending order of ρ_e. Then the breakpoints of S's interdiction curve are 0, r_1, $r_1 + r_2$, $r_1 + r_2 + r_3$, $\ldots$. The slopes of S's parametric capacity curve are $\ldots$, $r_1 + r_2 + r_3$, $r_1 + r_2$, r_1, 0.

Conjugate duality between interdiction and parametric capacity for S

- Both curves are piecewise linear; the interdiction curve (residual capacity curve) is convex, the parametric capacity curve is concave.
- Index the arcs of $\delta^+(S)$ in descending order of ρ_e. Then the breakpoints of S's interdiction curve are 0, r_1, $r_1 + r_2$, $r_1 + r_2 + r_3$, ... The slopes of S's parametric capacity curve are . . . , $r_1 + r_2 + r_3$, $r_1 + r_2$, r_1, 0.
- The slopes of S's interdiction curve are $-\rho_1$, $-\rho_2$, ... The breakpoints of S's parametric capacity curve are . . . , ρ_3, ρ_2, ρ_1.

Conjugate duality between interdiction and parametric capacity for S

- Both curves are piecewise linear; the interdiction curve (residual capacity curve) is convex, the parametric capacity curve is concave.
- Index the arcs of $\delta^+(S)$ in descending order of ρ_e. Then the breakpoints of S's interdiction curve are 0, r_1, $r_1 + r_2$, $r_1 + r_2 + r_3$, ... The slopes of S's parametric capacity curve are ..., $r_1 + r_2 + r_3$, $r_1 + r_2$, r_1, 0.
- The slopes of S's interdiction curve are $-\rho_1$, $-\rho_2$, ... The breakpoints of S's parametric capacity curve are ..., ρ_3, ρ_2, ρ_1.
- Thus breakpoints and slopes are interchanged between S's interdiction curve and its parametric capacity curve, though in reverse order and modulo a minus sign.

Conjugate duality between interdiction and parametric capacity for S

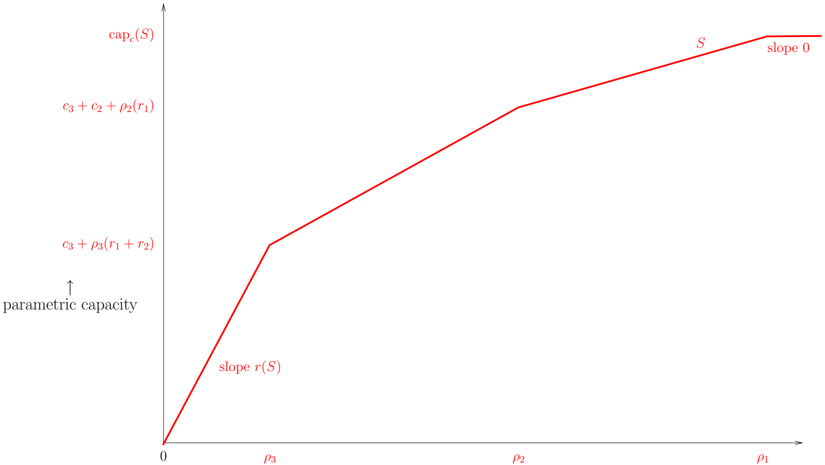
- Both curves are piecewise linear; the interdiction curve (residual capacity curve) is convex, the parametric capacity curve is concave.
- Index the arcs of $\delta^+(S)$ in descending order of ρ_e. Then the breakpoints of S's interdiction curve are 0, r_1, $r_1 + r_2$, $r_1 + r_2 + r_3$, ... The slopes of S's parametric capacity curve are ..., $r_1 + r_2 + r_3$, $r_1 + r_2$, r_1, 0.
- The slopes of S's interdiction curve are $-\rho_1$, $-\rho_2$, ... The breakpoints of S's parametric capacity curve are ..., ρ_3, ρ_2, ρ_1.
- Thus breakpoints and slopes are interchanged between S's interdiction curve and its parametric capacity curve, though in reverse order and modulo a minus sign.
- In the language of conjugate duality, this is equivalent to saying that the parametric capacity curve $\text{cap}(S, \lambda)$ is the negative of the conjugate dual of the interdiction curve for S, evaluated at $-\lambda$.

The overall parametric capacity curve: the λ-profile

Now overlay the parametric capacity curves for all S.

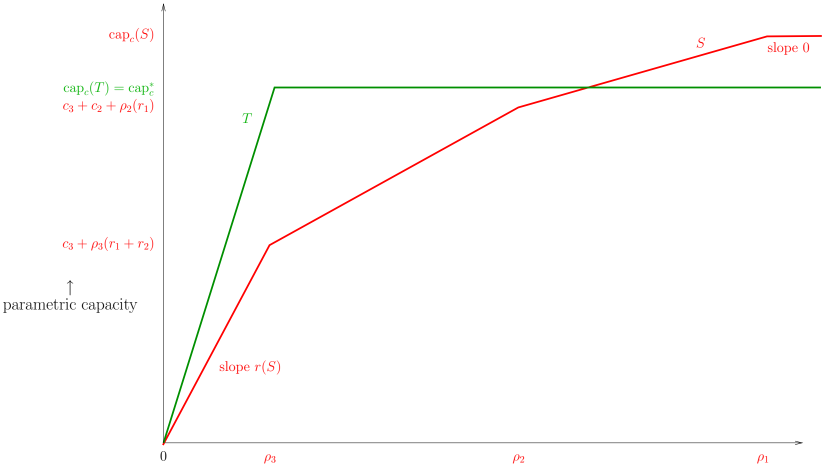
The overall parametric capacity curve: the λ-profile

Now overlay the parametric capacity curves for all S.



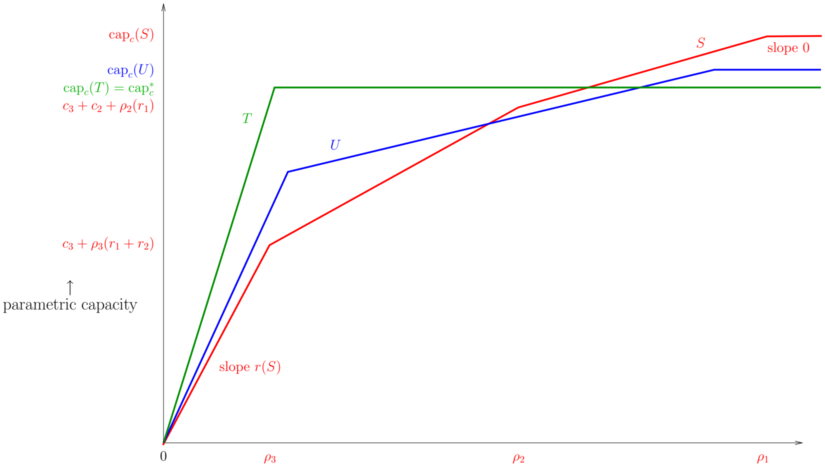
The overall parametric capacity curve: the λ-profile

Now overlay the parametric capacity curves for all S.



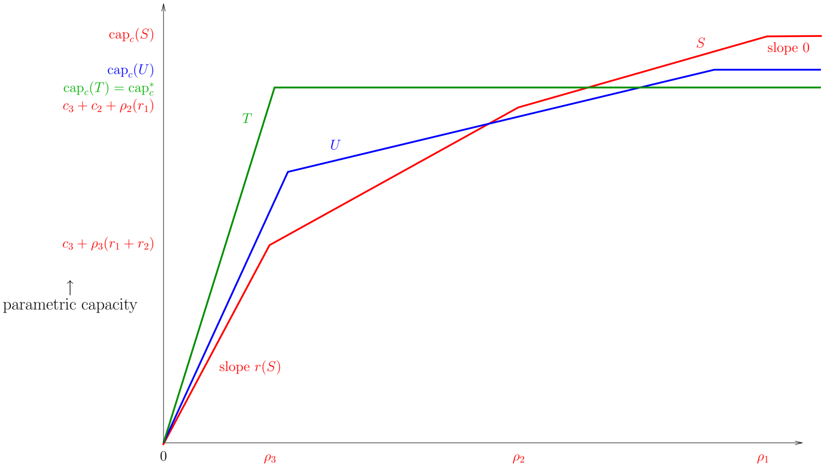
The overall parametric capacity curve: the λ-profile

Now overlay the parametric capacity curves for all S.



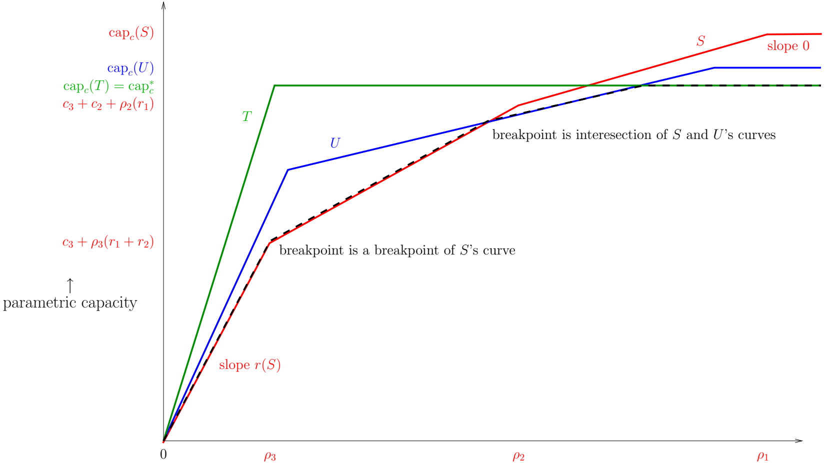
The overall parametric capacity curve: the λ-profile

For a fixed value of λ, we want to find the S whose parametric capacity at λ is minimum, so we just want the pointwise minimum of all these curves.



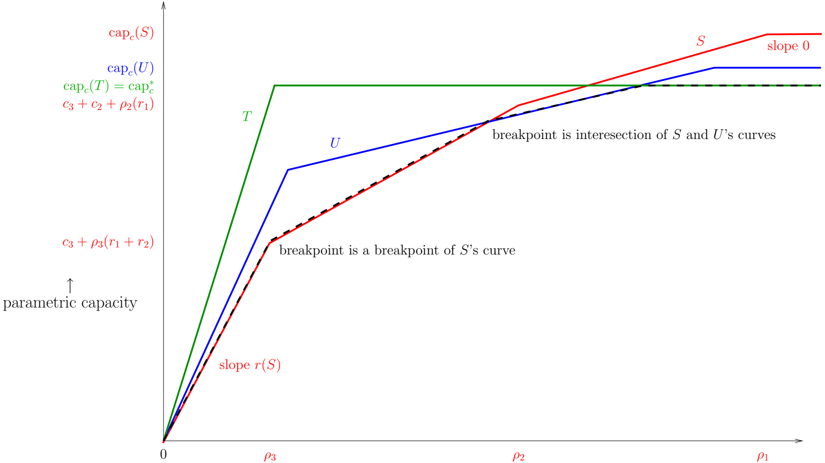
The overall parametric capacity curve: the λ-profile

For a fixed value of λ, we want to find the S whose parametric capacity at λ is minimum, so we just want the pointwise minimum of all these curves.



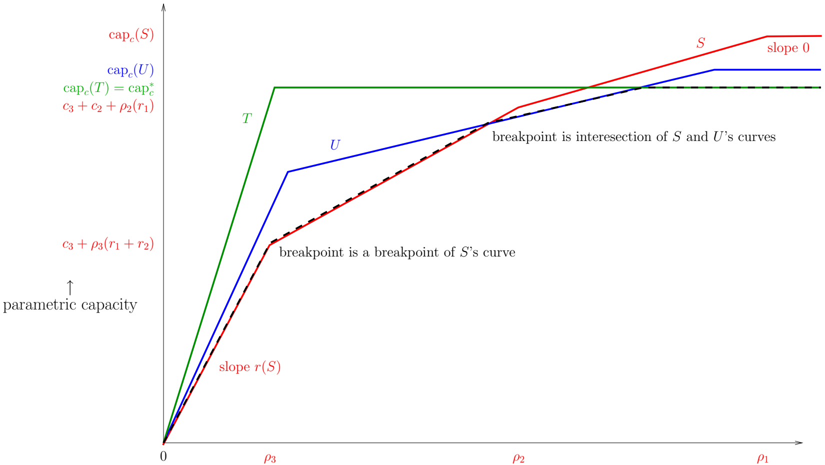
The overall parametric capacity curve: the λ-profile

Since the minimum of a bunch of concave curves is again concave, this time we do not need to linearize. We call this overall parametric capacity curve the λ-profile.



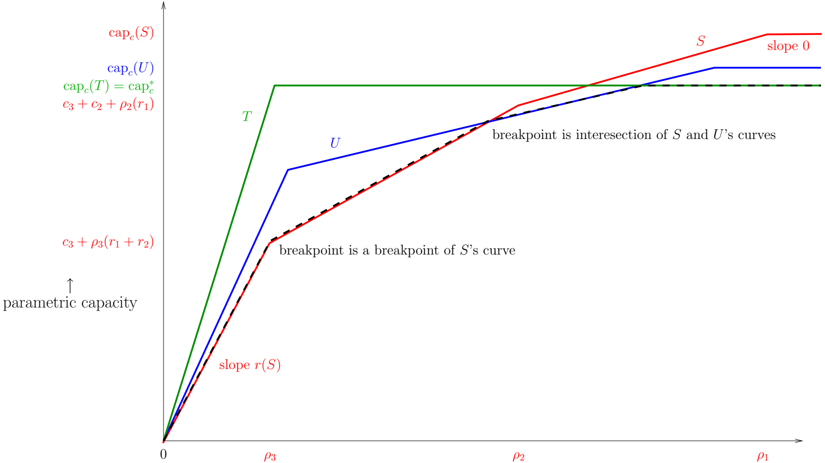
The overall parametric capacity curve: the λ-profile

We can compute things like $\mathrm{cap}^*(\lambda)$ easily using parametric min cut technology.



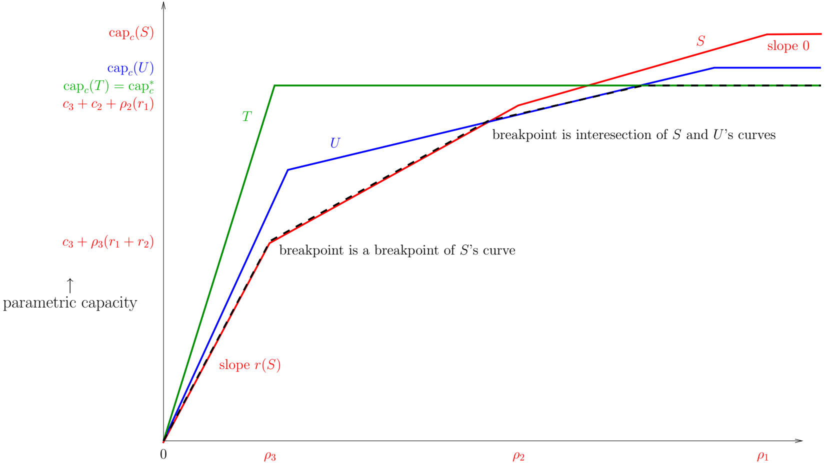
The overall parametric capacity curve: the λ-profile

We can show that the conjugate duality between S's interdiction and parametric capacity curves carries over to conjugate duality between the B-profile and the λ-profile.



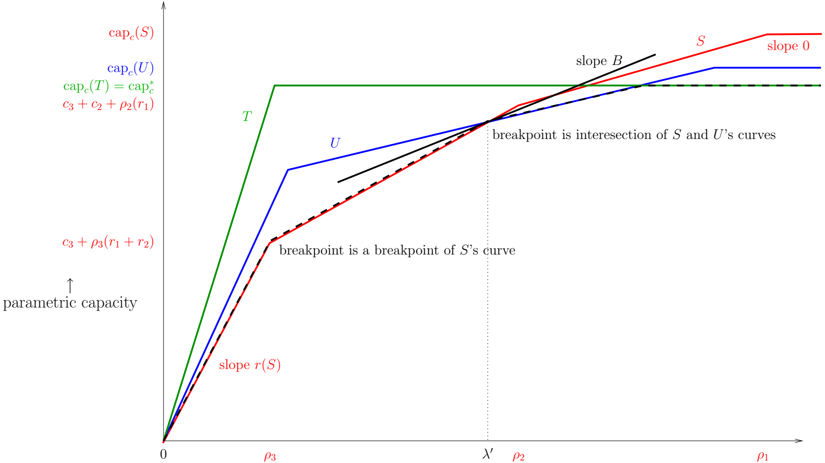
The overall parametric capacity curve: the λ-profile

Recall that to get our pseudo-approximation for a given B, we want to compute the two cuts S_1 and S_2 bracketing B on the B-profile.



The overall parametric capacity curve: the λ-profile

Conjugate duality implies that this is equivalent to finding a breakpoint λ' on the λ-profile whose adjacent slopes bracket B, here S and U; we also get $B_1 = B(S_1, \lambda')$, $C_1 = C(S_1, \lambda')$, $B_2 = B(S_2, \lambda')$, and $C_2 = C(S_2, \lambda')$.



Outline

1. Network Interdiction
   - What is it?
   - Interdiction curves
2. LP Duality
   - Dual of interdiction
3. Parametric Min Cut
   - Parametric curves
4. The Breakpoint Subproblem
   - What is it?
   - Algorithms
   - Discrete Newton
5. Multiple Parameters
   - What is it?
   - Scheduling problem
   - Multi-GGT

The key breakpoint subproblem

- Notice that any breakpoint $\hat{\lambda}$ of the λ-profile is defined by the intersection of a segment to its left coming from cut $S^-(\hat{\lambda})$ with local slope $\mathrm{sl}^-(\hat{\lambda})$, and a segment to its right coming from cut $S^+(\hat{\lambda})$ with local slope $\mathrm{sl}^+(\hat{\lambda})$, with $\mathrm{sl}^-(\hat{\lambda}) > \mathrm{sl}^+(\hat{\lambda})$ by concavity.

The key breakpoint subproblem

- Notice that any breakpoint $\hat{\lambda}$ of the λ-profile is defined by the intersection of a segment to its left coming from cut $S^-(\hat{\lambda})$ with local slope $\mathrm{sl}^-(\hat{\lambda})$, and a segment to its right coming from cut $S^+(\hat{\lambda})$ with local slope $\mathrm{sl}^+(\hat{\lambda})$, with $\mathrm{sl}^-(\hat{\lambda}) > \mathrm{sl}^+(\hat{\lambda})$ by concavity.
- The subproblem we now want to solve combinatorially: Given B, find breakpoint λ_B of the λ-profile such that $\mathrm{sl}^+(\lambda_B) \leq B \leq \mathrm{sl}^-(\lambda_B)$, along with the corresponding $S^-(\lambda_B)$ and $S^+(\lambda_B)$.

The key breakpoint subproblem

- Notice that any breakpoint $\hat{\lambda}$ of the λ-profile is defined by the intersection of a segment to its left coming from cut $S^-(\hat{\lambda})$ with local slope $\mathrm{sl}^-(\hat{\lambda})$, and a segment to its right coming from cut $S^+(\hat{\lambda})$ with local slope $\mathrm{sl}^+(\hat{\lambda})$, with $\mathrm{sl}^-(\hat{\lambda}) > \mathrm{sl}^+(\hat{\lambda})$ by concavity.
- The subproblem we now want to solve combinatorially: Given B, find breakpoint λ_B of the λ-profile such that $\mathrm{sl}^+(\lambda_B) \le B \le \mathrm{sl}^-(\lambda_B)$, along with the corresponding $S^-(\lambda_B)$ and $S^+(\lambda_B)$.
- A technical detail: Suppose I give you λ_B. Can you then use it to compute $S^-(\lambda_B)$ and $S^+(\lambda_B)$?

The key breakpoint subproblem

- Notice that any breakpoint $\hat{\lambda}$ of the λ-profile is defined by the intersection of a segment to its left coming from cut $S^-(\hat{\lambda})$ with local slope $\mathrm{sl}^-(\hat{\lambda})$, and a segment to its right coming from cut $S^+(\hat{\lambda})$ with local slope $\mathrm{sl}^+(\hat{\lambda})$, with $\mathrm{sl}^-(\hat{\lambda}) > \mathrm{sl}^+(\hat{\lambda})$ by concavity.
- The subproblem we now want to solve combinatorially: Given B, find breakpoint λ_B of the λ-profile such that $\mathrm{sl}^+(\lambda_B) \leq B \leq \mathrm{sl}^-(\lambda_B)$, along with the corresponding $S^-(\lambda_B)$ and $S^+(\lambda_B)$.
- A technical detail: Suppose I give you λ_B. Can you then use it to compute $S^-(\lambda_B)$ and $S^+(\lambda_B)$?
  - Yes: We can use a combination of Picard-Queyranne decomposition w.r.t. an optimal flow at λ_B, and *min* flow / *max* cut in the residual network to find them

The key breakpoint subproblem

- Notice that any breakpoint $\hat{\lambda}$ of the λ-profile is defined by the intersection of a segment to its left coming from cut $S^-(\hat{\lambda})$ with local slope $\mathrm{sl}^-(\hat{\lambda})$, and a segment to its right coming from cut $S^+(\hat{\lambda})$ with local slope $\mathrm{sl}^+(\hat{\lambda})$, with $\mathrm{sl}^-(\hat{\lambda}) > \mathrm{sl}^+(\hat{\lambda})$ by concavity.
- The subproblem we now want to solve combinatorially: Given B, find breakpoint λ_B of the λ-profile such that $\mathrm{sl}^+(\lambda_B) \leq B \leq \mathrm{sl}^-(\lambda_B)$, along with the corresponding $S^-(\lambda_B)$ and $S^+(\lambda_B)$.
- A technical detail: Suppose I give you λ_B. Can you then use it to compute $S^-(\lambda_B)$ and $S^+(\lambda_B)$?
  - Yes: We can use a combination of Picard-Queyranne decomposition w.r.t. an optimal flow at λ_B, and *min* flow / *max* cut in the residual network to find them
- So let's just concentrate on finding λ_B.

Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.

Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}^*_r$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.
2. Compute $\hat{\lambda} = (\lambda_L + \lambda_R)/2$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.

Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.
2. Compute $\hat{\lambda} = (\lambda_L + \lambda_R)/2$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.
3. If $B \in [\mathrm{sl}^+(\hat{\lambda}), \mathrm{sl}^-(\hat{\lambda})]$, then $\lambda_B = \hat{\lambda}$ and we can stop.

Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.
2. Compute $\hat{\lambda} = (\lambda_L + \lambda_R)/2$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.
3. If $B \in [\mathrm{sl}^+(\hat{\lambda}), \mathrm{sl}^-(\hat{\lambda})]$, then $\lambda_B = \hat{\lambda}$ and we can stop.
4. Otherwise, if $B < \mathrm{sl}^+(\hat{\lambda})$ then replace λ_L by $\hat{\lambda}$; else ($B > \mathrm{sl}^-(\hat{\lambda})$) replace λ_R by $\hat{\lambda}$ and go to 2.

Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.
2. Compute $\hat{\lambda} = (\lambda_L + \lambda_R)/2$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.
3. If $B \in [\mathrm{sl}^+(\hat{\lambda}), \mathrm{sl}^-(\hat{\lambda})]$, then $\lambda_B = \hat{\lambda}$ and we can stop.
4. Otherwise, if $B < \mathrm{sl}^+(\hat{\lambda})$ then replace λ_L by $\hat{\lambda}$; else $(B > \mathrm{sl}^-(\hat{\lambda}))$ replace λ_R by $\hat{\lambda}$ and go to 2.

- This runs in something like $\Theta(\log(nD))$ time, where D is the size of the data.

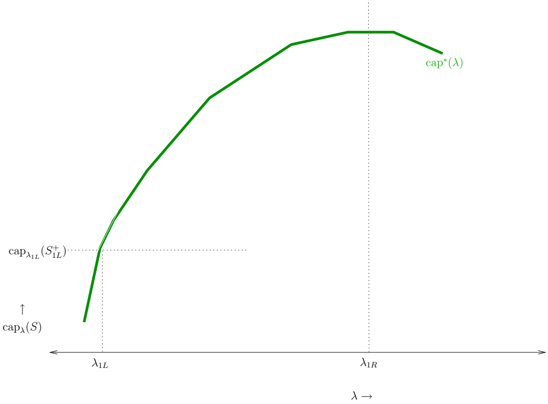
Binary search solves it

1. Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$; then all interesting values of λ are in $[\lambda_L, \lambda_R]$.
2. Compute $\hat{\lambda} = (\lambda_L + \lambda_R)/2$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.
3. If $B \in [\mathrm{sl}^+(\hat{\lambda}), \mathrm{sl}^-(\hat{\lambda})]$, then $\lambda_B = \hat{\lambda}$ and we can stop.
4. Otherwise, if $B < \mathrm{sl}^+(\hat{\lambda})$ then replace λ_L by $\hat{\lambda}$; else $(B > \mathrm{sl}^-(\hat{\lambda}))$ replace λ_R by $\hat{\lambda}$ and go to 2.

- This runs in something like $\Theta(\log(nD))$ time, where D is the size of the data.
- Can we do better?

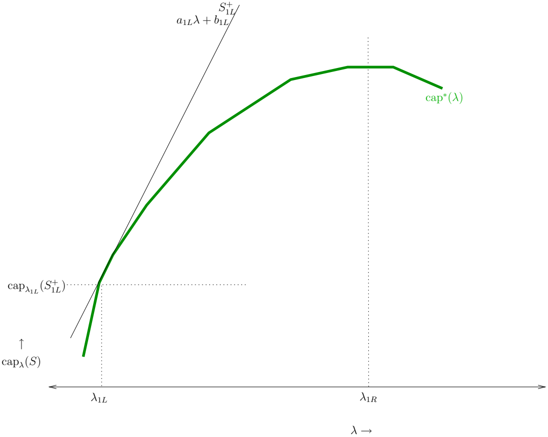
Discrete Newton gives a better algorithm

Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$ as before. Denote $\mathrm{sl}^+(\lambda_L)$ by sl_L^+ and $\mathrm{sl}^-(\lambda_R)$ by sl_R^-.



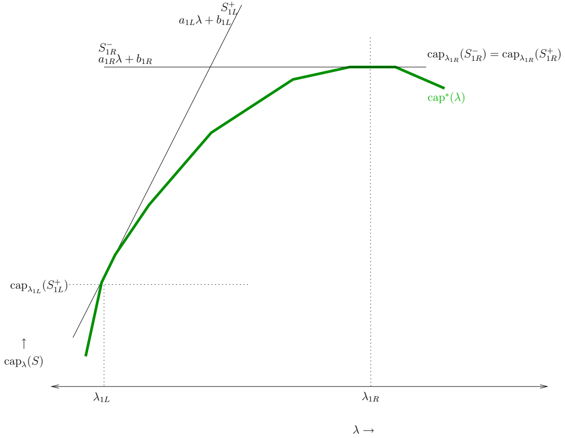
Discrete Newton gives a better algorithm

Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}_r^*$ as before. Denote $\mathrm{sl}^+(\lambda_L)$ by sl_L^+ and $\mathrm{sl}^-(\lambda_R)$ by sl_R^-.



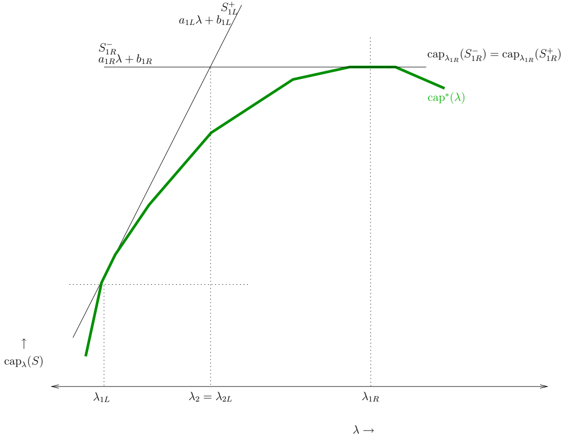
Discrete Newton gives a better algorithm

Set $\lambda_L = 0$ and $\lambda_R = \mathrm{cap}^*_r$ as before. Denote $\mathrm{sl}^+(\lambda_L)$ by sl^+_L and $\mathrm{sl}^-(\lambda_R)$ by sl^-_R.



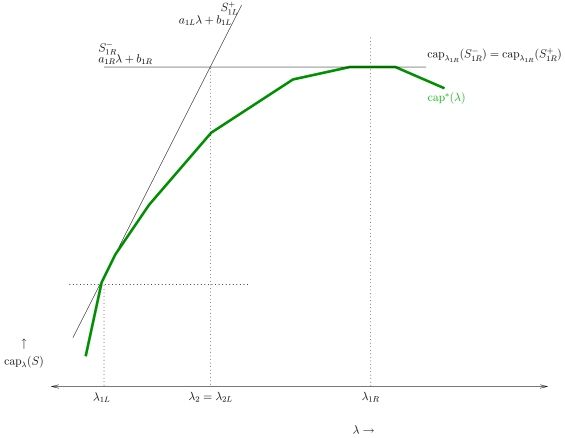
Discrete Newton gives a better algorithm

Compute $\hat{\lambda}$ as the intersection of the line of slope sl_L^+ through $(\lambda_L, \mathrm{cap}^*(\lambda_L))$, and the line of slope sl_R^- through $(\lambda_R, \mathrm{cap}^*(\lambda_R))$, a max flow w.r.t. $\hat{\lambda}$, and $\mathrm{sl}^-(\hat{\lambda})$ and $\mathrm{sl}^+(\hat{\lambda})$.



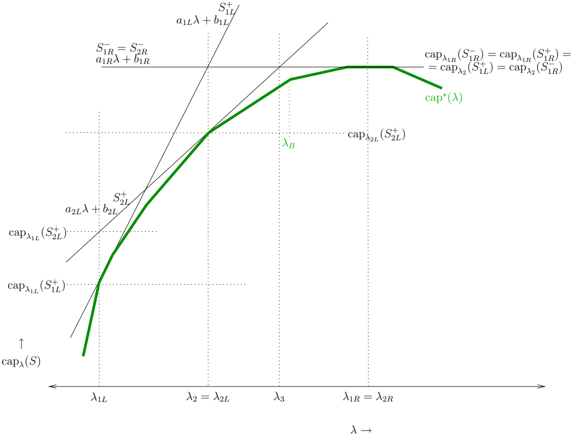
Discrete Newton gives a better algorithm

If $B \in [\mathrm{sl}^+(\hat{\lambda}), \mathrm{sl}^-(\hat{\lambda})]$, then $\lambda_B = \hat{\lambda}$ and we can stop.



Discrete Newton gives a better algorithm

Otherwise, if $B < \mathrm{sl}^+(\hat\lambda)$ then replace λ_L by $\hat\lambda$; else ($B > \mathrm{sl}^-(\hat\lambda)$) replace λ_R by $\hat\lambda$ and go to 2.



Defining gaps

- How can we analyze the running time of this Newton-B algorithm?

Defining gaps

- How can we analyze the running time of this Newton-B algorithm?
- Let's think in terms of lines of slope B. Let L^* denote the line of slope B through the (as-yet unknown) point $(\lambda_B, \mathrm{cap}^*(\lambda_B))$. This L^* is the highest possible line of slope B through any point of the λ-profile.

Defining gaps

- How can we analyze the running time of this Newton-B algorithm?
- Let's think in terms of lines of slope B. Let L^* denote the line of slope B through the (as-yet unknown) point $(\lambda_B, \text{cap}^*(\lambda_B))$. This L^* is the highest possible line of slope B through any point of the λ-profile.
- Thus the line of slope B through, e.g., $(\lambda_L, \text{cap}^*(\lambda_L))$ lies below L^*.

Defining gaps

- How can we analyze the running time of this Newton-B algorithm?
- Let's think in terms of lines of slope B. Let L^* denote the line of slope B through the (as-yet unknown) point $(\lambda_B, \mathrm{cap}^*(\lambda_B))$. This L^* is the highest possible line of slope B through any point of the λ-profile.
- Thus the line of slope B through, e.g., $(\lambda_L, \mathrm{cap}^*(\lambda_L))$ lies below L^*.
- Since the lines defining $\hat{\lambda}$ are tangents to the λ-profile, their intersection must lie above L^*, and so the line of slope B through this intersection point lies above L^*.

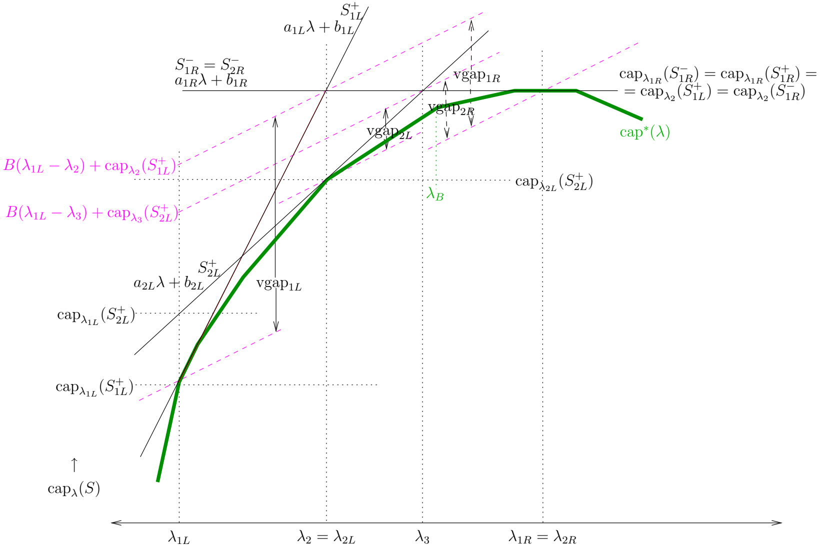
Defining gaps

- How can we analyze the running time of this Newton-B algorithm?
- Let's think in terms of lines of slope B. Let L^* denote the line of slope B through the (as-yet unknown) point $(\lambda_B, \mathrm{cap}^*(\lambda_B))$. This L^* is the highest possible line of slope B through any point of the λ-profile.
- Thus the line of slope B through, e.g., $(\lambda_L, \mathrm{cap}^*(\lambda_L))$ lies below L^*.
- Since the lines defining $\hat{\lambda}$ are tangents to the λ-profile, their intersection must lie above L^*, and so the line of slope B through this intersection point lies above L^*.
- Define vgap_L to be the vertical distance between the line of slope B through the intersection point, and the line of slope B through $(\lambda_L, \mathrm{cap}^*(\lambda_L))$, and similarly for vgap_R.

Defining gaps

- How can we analyze the running time of this Newton-B algorithm?
- Let's think in terms of lines of slope B. Let L^* denote the line of slope B through the (as-yet unknown) point $(\lambda_B, \mathrm{cap}^*(\lambda_B))$. This L^* is the highest possible line of slope B through any point of the λ-profile.
- Thus the line of slope B through, e.g., $(\lambda_L, \mathrm{cap}^*(\lambda_L))$ lies below L^*.
- Since the lines defining $\hat{\lambda}$ are tangents to the λ-profile, their intersection must lie above L^*, and so the line of slope B through this intersection point lies above L^*.
- Define vgap_L to be the vertical distance between the line of slope B through the intersection point, and the line of slope B through $(\lambda_L, \mathrm{cap}^*(\lambda_L))$, and similarly for vgap_R.
- Also define slgap_L to be $\mathrm{sl}_L^+ - B$ and slgap_R to be $B - \mathrm{sl}_R^-$.

vgap illustrated



The key inequality

- We use primes to denote new values. When $\hat{\lambda}$ becomes the new λ_L then the key inequality is

$$\frac{\mathrm{vgap}'_L}{\mathrm{vgap}_L} + \frac{\mathrm{slgap}'_L}{\mathrm{slgap}_L} < 1 \tag{1}$$

The key inequality

- We use primes to denote new values. When $\hat{\lambda}$ becomes the new λ_L then the key inequality is

$$\frac{\mathrm{vgap}'_L}{\mathrm{vgap}_L} + \frac{\mathrm{slgap}'_L}{\mathrm{slgap}_L} < 1 \tag{1}$$

- This immediately implies that at each iteration, one of vgap_L, vgap_R, slgap_L, or slgap_R is cut down by a factor of at least 2. Thus Newton-B is never worse than Binary Search.

The key inequality

- We use primes to denote new values. When $\hat{\lambda}$ becomes the new λ_L then the key inequality is

$$\frac{\mathrm{vgap}'_L}{\mathrm{vgap}_L} + \frac{\mathrm{slgap}'_L}{\mathrm{slgap}_L} < 1 \tag{1}$$

- This immediately implies that at each iteration, one of vgap_L, vgap_R, slgap_L, or slgap_R is cut down by a factor of at least 2. Thus Newton-B is never worse than Binary Search.
- (1) was originally proved in Mc+Ervolina '94. Then Rote, and Radzik '92–'98 showed that Newton-B is sometimes faster than Binary Search, and has a strongly polynomial bound.

The key inequality

- We use primes to denote new values. When $\hat{\lambda}$ becomes the new λ_L then the key inequality is

$$\frac{\mathrm{vgap}'_L}{\mathrm{vgap}_L} + \frac{\mathrm{slgap}'_L}{\mathrm{slgap}_L} < 1 \tag{1}$$

- This immediately implies that at each iteration, one of vgap_L, vgap_R, slgap_L, or slgap_R is cut down by a factor of at least 2. Thus Newton-B is never worse than Binary Search.
- (1) was originally proved in Mc+Ervolina '94. Then Rote, and Radzik '92–'98 showed that Newton-B is sometimes faster than Binary Search, and has a strongly polynomial bound.
  - The better weakly polynomial bound is $O\left(\frac{\log(nD)}{1+\log\log(nD)-\log\log n}\right)$.

The key inequality

- We use primes to denote new values. When $\hat{\lambda}$ becomes the new λ_L then the key inequality is

$$\frac{\mathrm{vgap}'_L}{\mathrm{vgap}_L} + \frac{\mathrm{slgap}'_L}{\mathrm{slgap}_L} < 1 \tag{1}$$

- This immediately implies that at each iteration, one of vgap_L, vgap_R, slgap_L, or slgap_R is cut down by a factor of at least 2. Thus Newton-B is never worse than Binary Search.
- (1) was originally proved in Mc+Ervolina '94. Then Rote, and Radzik '92–'98 showed that Newton-B is sometimes faster than Binary Search, and has a strongly polynomial bound.
  - The better weakly polynomial bound is $O\left(\frac{\log(nD)}{1+\log\log(nD)-\log\log n}\right)$.
  - Sometimes there is an $O(m)$ bound on the number of iterations.

Some implications

- We didn't use much network structure in this analysis.

Some implications

- We didn't use much network structure in this analysis.
- Thus we could define an interdiction version of any capacitated problem that can be formulated as an LP.

Some implications

- We didn't use much network structure in this analysis.
- Thus we could define an interdiction version of any capacitated problem that can be formulated as an LP.
  - Primalizing the LP would give a conjugate dual parametric problem that we could then solve via Newton-B.

Some implications

- We didn't use much network structure in this analysis.
- Thus we could define an interdiction version of any capacitated problem that can be formulated as an LP.
  - Primalizing the LP would give a conjugate dual parametric problem that we could then solve via Newton-B.
  - The Burch et al pseudo-approximation framework carries through also.

Some implications

- We didn't use much network structure in this analysis.
- Thus we could define an interdiction version of any capacitated problem that can be formulated as an LP.
    - Primalizing the LP would give a conjugate dual parametric problem that we could then solve via Newton-B.
    - The Burch et al pseudo-approximation framework carries through also.
    - We are in the process of identifying other such problems.

Some implications

- We didn't use much network structure in this analysis.
- Thus we could define an interdiction version of any capacitated problem that can be formulated as an LP.
  - Primalizing the LP would give a conjugate dual parametric problem that we could then solve via Newton-B.
  - The Burch et al pseudo-approximation framework carries through also.
  - We are in the process of identifying other such problems.
- Indeed, this Newton-B algorithm and its analysis works for any concave (or convex) function, even continuous ones.

Outline

Multiple budgets equals multiple parameters

- A natural generalization is when there are multiple ways to destroy capacity, at different costs.

Multiple budgets equals multiple parameters

- A natural generalization is when there are multiple ways to destroy capacity, at different costs.
- It should be clear via duality that this would turn into a parametric min cut problem with multiple parameters.

Multiple budgets equals multiple parameters

- A natural generalization is when there are multiple ways to destroy capacity, at different costs.
- It should be clear via duality that this would turn into a parametric min cut problem with multiple parameters.
- Now we'd be trying to find a point on the parametric surface whose local derivatives bracket the given budgets in the coordinate directions.

Multiple budgets equals multiple parameters

- A natural generalization is when there are multiple ways to destroy capacity, at different costs.
- It should be clear via duality that this would turn into a parametric min cut problem with multiple parameters.
- Now we'd be trying to find a point on the parametric surface whose local derivatives bracket the given budgets in the coordinate directions.
- As before we could solve this via LP, but we'd prefer a combinatorial algorithm.

Multiple budgets equals multiple parameters

- A natural generalization is when there are multiple ways to destroy capacity, at different costs.
- It should be clear via duality that this would turn into a parametric min cut problem with multiple parameters.
- Now we'd be trying to find a point on the parametric surface whose local derivatives bracket the given budgets in the coordinate directions.
- As before we could solve this via LP, but we'd prefer a combinatorial algorithm.
- Interdiction already gets complicated with two parameters, so let's consider a simpler multiple parameter scheduling problem instead.

Chen's '94 scheduling problem

- We again start with a usual max flow network. We think of the nodes j such that $s \rightarrow j \in A$ as jobs, and we denote c_{sj} by p_j, the processing time of job j.

Chen's '94 scheduling problem

- We again start with a usual max flow network. We think of the nodes j such that $s \to j \in A$ as jobs, and we denote c_{sj} by p_j, the processing time of job j.
- If a max flow in the network saturates all of these job arcs, then we are happy and the problem goes away.

Chen's '94 scheduling problem

- We again start with a usual max flow network. We think of the nodes j such that $s \to j \in A$ as jobs, and we denote c_{sj} by p_j, the processing time of job j.
- If a max flow in the network saturates all of these job arcs, then we are happy and the problem goes away.
- So assume instead that there is some non-trivial min cut. We want to *outsource* some of the processing of jobs until there exists a max flow saturating the residual processing time of every job.

Chen's '94 scheduling problem

- We again start with a usual max flow network. We think of the nodes j such that $s \to j \in A$ as jobs, and we denote c_{sj} by p_j, the processing time of job j.
- If a max flow in the network saturates all of these job arcs, then we are happy and the problem goes away.
- So assume instead that there is some non-trivial min cut. We want to *outsource* some of the processing of jobs until there exists a max flow saturating the residual processing time of every job.
- Initially assume that if we pay $\$\lambda$, we reduce p_j to $\max(0, p_j - a_j\lambda)$ (where $a_j \geq 0$ is given for each j).

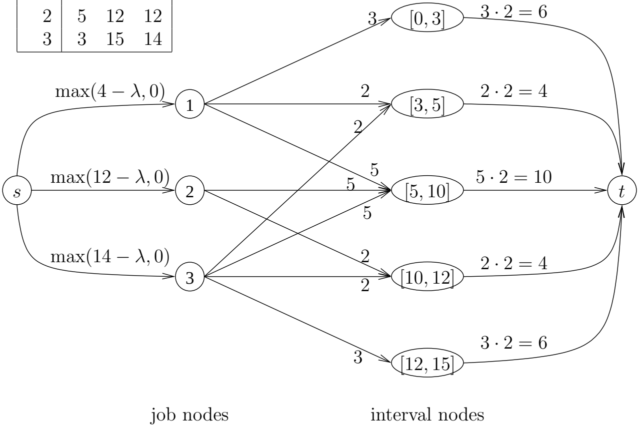
Chen's '94 scheduling problem

- We again start with a usual max flow network. We think of the nodes j such that $s \to j \in A$ as jobs, and we denote c_{sj} by p_j, the processing time of job j.
- If a max flow in the network saturates all of these job arcs, then we are happy and the problem goes away.
- So assume instead that there is some non-trivial min cut. We want to *outsource* some of the processing of jobs until there exists a max flow saturating the residual processing time of every job.
- Initially assume that if we pay $\$\lambda$, we reduce p_j to $\max(0, p_j - a_j\lambda)$ (where $a_j \geq 0$ is given for each j).
- Now we want to minimize λ such that there exists a flow saturating all residual job arcs.

Chen's scheduling problem: example

Here is a specific instance of this type of scheduling problem.

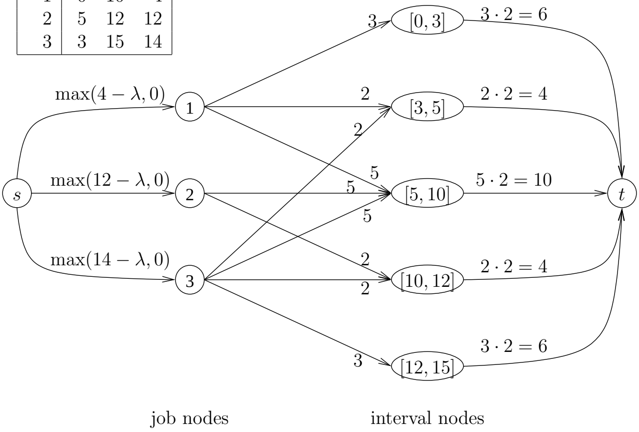
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

Here we have jobs 1, 2, 3 that we are scheduling on two machines.

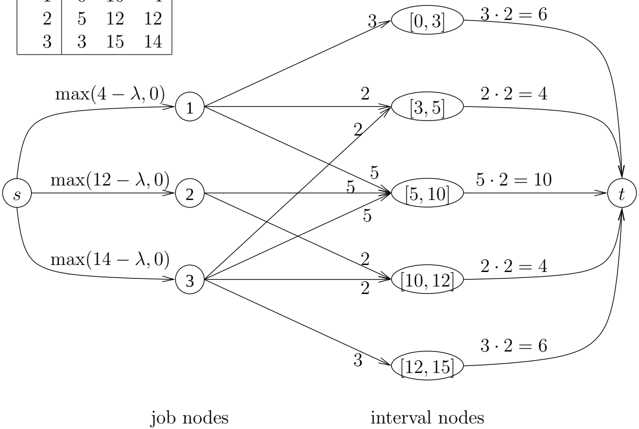
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

Job 1 is available during $[0, 10]$; 2 during $[5, 12]$; 3 during $[3, 15]$.

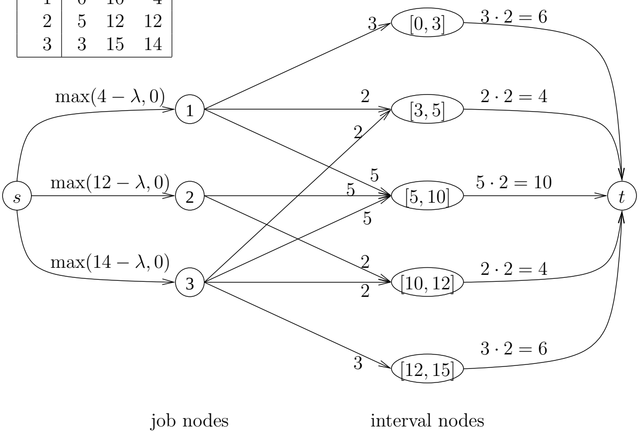
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

These time slots divide the total time into the five time intervals on the right.

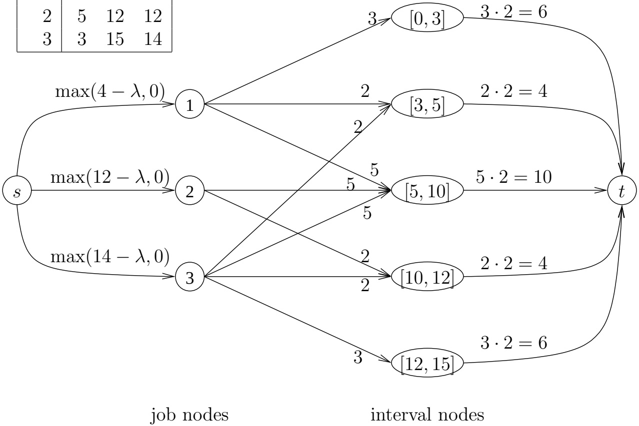
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

The capacity into an interval is its width; the capacity out of an interval is (# of machines) times its width.

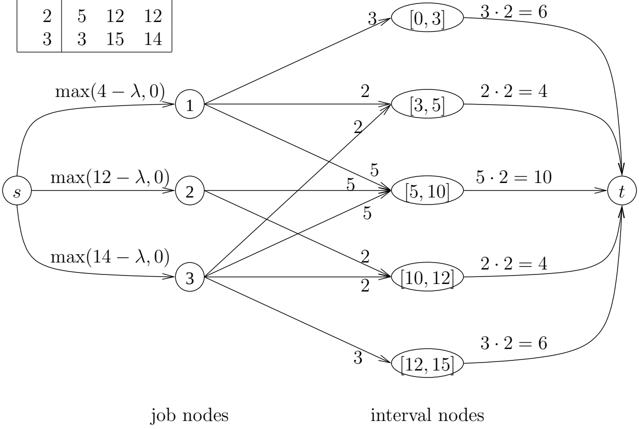
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

At $\lambda = 0$ there is no flow saturating s since, e.g., the total capacity out of $2 = 2 + 5 < 14 =$ required flow into 2.

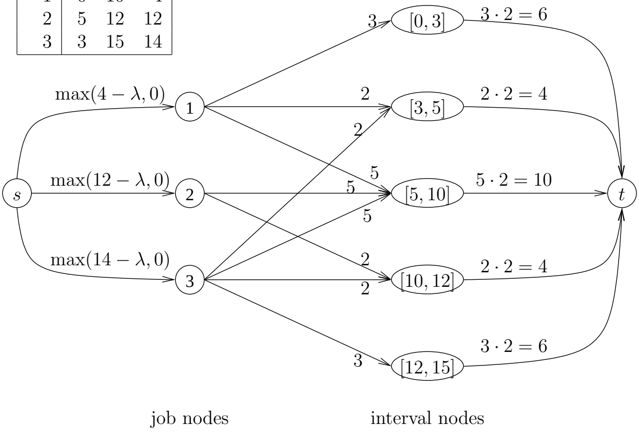
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

At $\lambda = 0$ the Min Cut is determined by jobs 2 & 3 requiring $12 + 14 = 26$ units, but having access to only 19 units of capacity, a gap of 7 units.

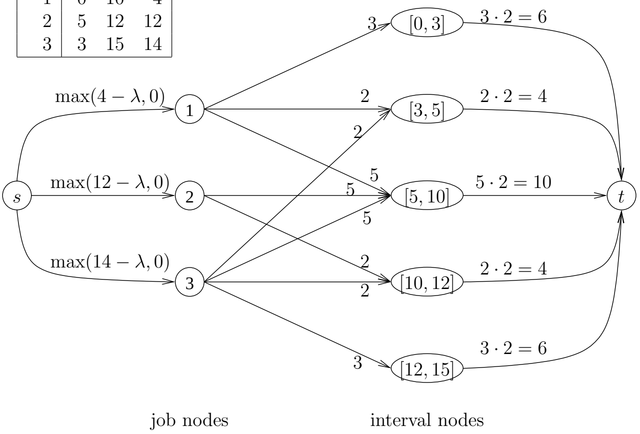
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

Thus we need to increase λ to at least $7/2 = 3.5$ to become feasible.

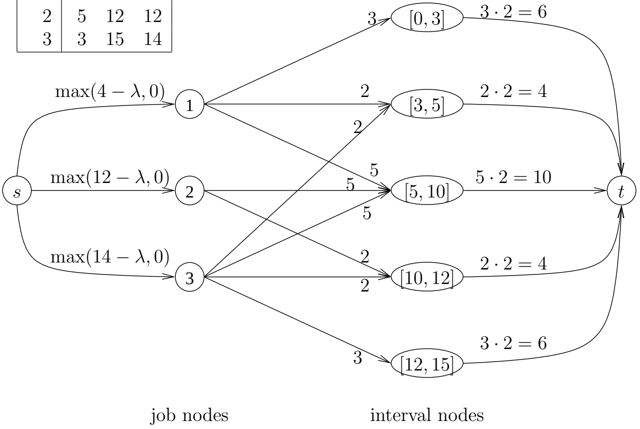
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

At $\lambda = 3.5$ there is still a gap at 2: it requires 8.5 units, but has access to only 7 units, so λ increases from 3.5 to 5, and now feasible.

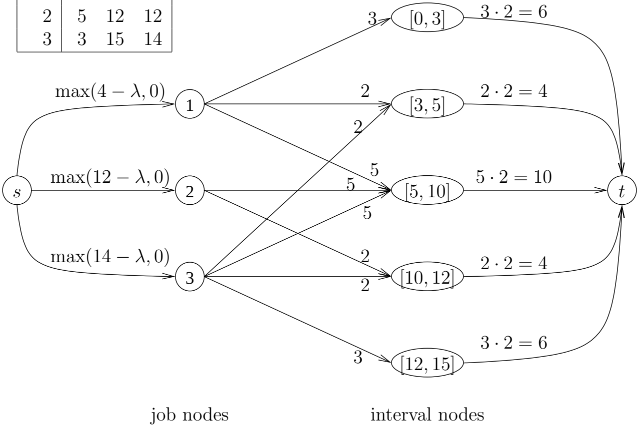
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

This Newton-type algorithm uses $O(\#\text{ jobs})$ iterations.

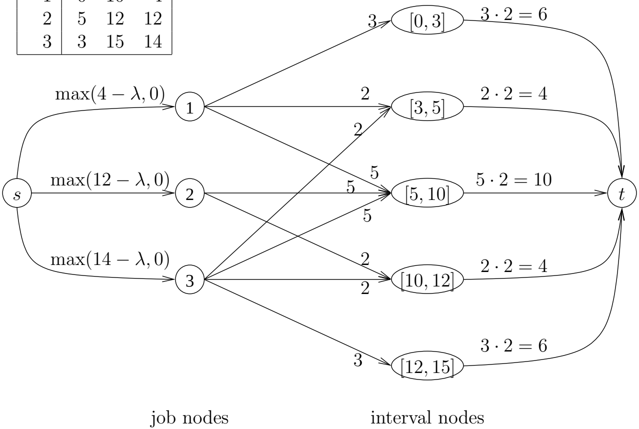
job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



Chen's scheduling problem: example

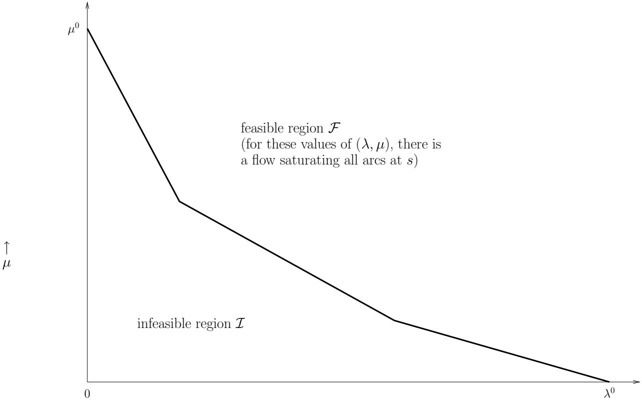
But it can be done in $O(1)$ MFs via Gallo-Grigoriadis-Tarjan (GGT) '89 parametric min cut.

job	r_j	d_j	p_j
1	0	10	4
2	5	12	12
3	3	15	14



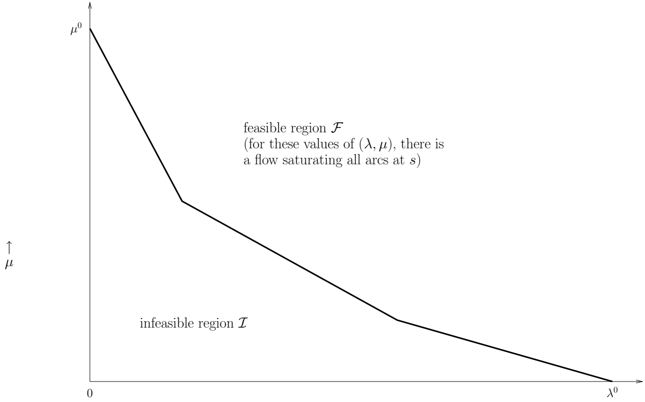
Two-parameter Chen

Suppose now that there are two ways to outsource, λ and μ such that if we pay $\$\lambda + \$\mu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu)$.
In the (λ, μ) plane there is a piecewise linear convex curve separating feasible points from infeasible ones.



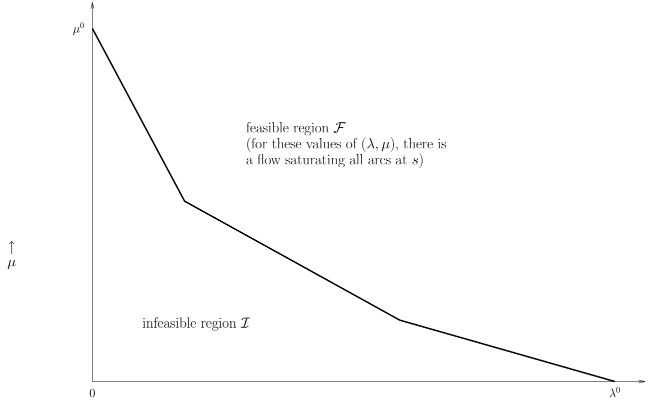
Two-parameter Chen

Suppose now that there are two ways to outsource, λ and μ such that if we pay $\$\lambda + \$\mu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu)$.
For cut S with $D \subseteq \delta^+(S) \cap \delta^+(\{s\})$, the constraints defining this region have the form $\lambda a(D) + \mu b(D) \geq p(D) - c(\delta^+(S))$.



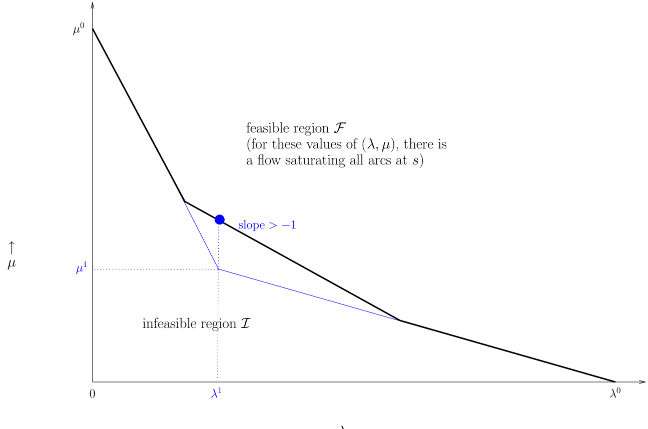
Two-parameter Chen

Suppose now that there are two ways to outsource, λ and μ such that if we pay $\$\lambda + \$\mu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu)$.
We want to find a breakpoint of this curve whose local slopes bracket slope -1.



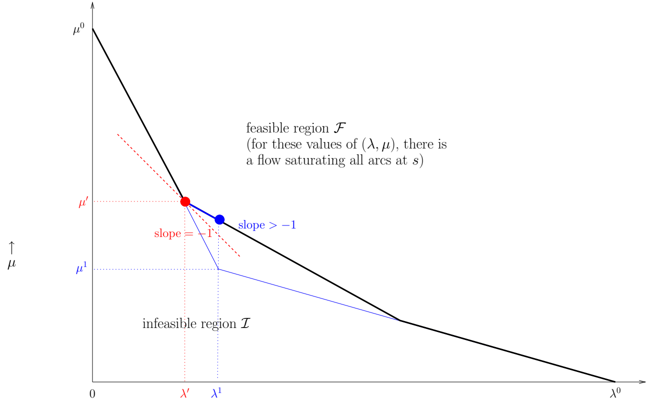
Two-parameter Chen

Suppose now that there are two ways to outsource, λ and μ such that if we pay $\$\lambda + \$\mu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu)$.
We know how to do this: Newton-B.



Two-parameter Chen

Suppose now that there are two ways to outsource, λ and μ such that if we pay $\$\lambda + \$\mu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu)$.
We know how to do this: Newton-B.



Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.
- As we vary ν, these 2-parameter solutions trace out a piecewise linear curve in the ν direction.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.
- As we vary ν, these 2-parameter solutions trace out a piecewise linear curve in the ν direction.
- We want to find a breakpoint on this curve whose local slopes bracket -1.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.
- As we vary ν, these 2-parameter solutions trace out a piecewise linear curve in the ν direction.
- We want to find a breakpoint on this curve whose local slopes bracket -1.
- Again, we know how to do this via a recursive application of Newton-B.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.
- As we vary ν, these 2-parameter solutions trace out a piecewise linear curve in the ν direction.
- We want to find a breakpoint on this curve whose local slopes bracket -1.
- Again, we know how to do this via a recursive application of Newton-B.
- This generalizes to any fixed number of parameters.

Three-parameter Chen

- Suppose now that there are three ways to outsource, λ, μ, and ν such that if we pay $\$\lambda + \$\mu + \$\nu$, we reduce p_j to $\max(0, p_j - a_j\lambda - b_j\mu - d_j\nu)$.
- For any fixed value of ν this is a 2-parameter problem we know how to solve.
- As we vary ν, these 2-parameter solutions trace out a piecewise linear curve in the ν direction.
- We want to find a breakpoint on this curve whose local slopes bracket -1.
- Again, we know how to do this via a recursive application of Newton-B.
- This generalizes to any fixed number of parameters.
- Open Question: LP is polynomial even when the number of parameters is not fixed. Can we get a combinatorial algorithm then?

Multi-parameter GGT

- Chen with ≥ 2 fits into a framework of Topkis '78 concerning parametric submodular optimization over a(n algebraic) lattice (here the lattice is $\mathbb{R}^2_+$ with $\leq$):

Multi-parameter GGT

- Chen with ≥ 2 fits into a framework of Topkis '78 concerning parametric submodular optimization over a(n algebraic) lattice (here the lattice is $\mathbb{R}^2_+$ with $\leq$):
  - The objective $\mathrm{cap}(S, \lambda, \mu)$ is submodular in S for each fixed (λ, μ).

Multi-parameter GGT

- Chen with ≥ 2 fits into a framework of Topkis '78 concerning parametric submodular optimization over a(n algebraic) lattice (here the lattice is $\mathbb{R}^2_+$ with $\leq$):
  - The objective $\text{cap}(S, \lambda, \mu)$ is submodular in S for each fixed (λ, μ).
  - It also satisfies Increasing Differences: for all $S \subseteq T$ and $(\lambda', \mu') \geq (\lambda, \mu)$,

$$\text{cap}(T, \lambda, \mu) - \text{cap}(T, \lambda', \mu') \leq \text{cap}(S, \lambda, \mu) - \text{cap}(S, \lambda', \mu').$$

Multi-parameter GGT

- Chen with ≥ 2 fits into a framework of Topkis '78 concerning parametric submodular optimization over a(n algebraic) lattice (here the lattice is $\mathbb{R}^2_+$ with $\leq$):
  - The objective $\text{cap}(S, \lambda, \mu)$ is submodular in S for each fixed (λ, μ).
  - It also satisfies Increasing Differences: for all $S \subseteq T$ and $(\lambda', \mu') \geq (\lambda, \mu)$,

$$\text{cap}(T, \lambda, \mu) - \text{cap}(T, \lambda', \mu') \leq \text{cap}(S, \lambda, \mu) - \text{cap}(S, \lambda', \mu').$$

- **Thm (Topkis):** With these two properties, if $(\lambda', \mu') \geq (\lambda, \mu)$ then $S^*(\lambda, \mu) \subseteq S^*(\lambda', \mu')$.

Multi-parameter GGT

- Chen with ≥ 2 fits into a framework of Topkis '78 concerning parametric submodular optimization over a(n algebraic) lattice (here the lattice is $\mathbb{R}^2_+$ with $\leq$):
  - The objective $\text{cap}(S, \lambda, \mu)$ is submodular in S for each fixed (λ, μ).
  - It also satisfies Increasing Differences: for all $S \subseteq T$ and $(\lambda', \mu') \geq (\lambda, \mu)$,

$$\text{cap}(T, \lambda, \mu) - \text{cap}(T, \lambda', \mu') \leq \text{cap}(S, \lambda, \mu) - \text{cap}(S, \lambda', \mu').$$

- **Thm (Topkis):** With these two properties, if $(\lambda', \mu') \geq (\lambda, \mu)$ then $S^*(\lambda, \mu) \subseteq S^*(\lambda', \mu')$.
- **Corollary:** In general, min cuts are non-decreasing along any chain in the lattice; for our 2-parameter scheduling problem, min cuts are increasing along any non-decreasing curve (chain) in $\mathbb{R}^2_+$.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
  - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
  - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.
    - This is *false* in general; Carstensen, and Mulmuley give examples where the parametric curve has an exponential number of min cuts.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
  - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.
    - This is *false* in general; Carstensen, and Mulmuley give examples where the parametric curve has an exponential number of min cuts.
    - But Discrete Newton works even in the general case.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
  - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.
    - This is *false* in general; Carstensen, and Mulmuley give examples where the parametric curve has an exponential number of min cuts.
    - But Discrete Newton works even in the general case.
  - True for our scheduling network, which is why we can use GGT.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
    - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.
        - This is *false* in general; Carstensen, and Mulmuley give examples where the parametric curve has an exponential number of min cuts.
        - But Discrete Newton works even in the general case.
    - True for our scheduling network, which is why we can use GGT.
- There are other 1-parameter cases where Topkis's structural result applies, see Arai, Ueno, Kajitani; Mc.; Fleischer; Fleischer, Iwata (SFM); Nagano (SFM); Scutellà; Milgrom and Shannon; Granot, Mc., Queyranne, Tardella.

GGT and Topkis in general

- For 1-parameter Min Cut, Increasing (or Decreasing) Differences is satisfied when only arcs at the source s (or sink t) are parametrized, and the capacities are monotone in the parameter.
  - Here monotone min cuts implies that min cuts are nested, and so there are only $O(n)$ min cuts.
    - This is *false* in general; Carstensen, and Mulmuley give examples where the parametric curve has an exponential number of min cuts.
    - But Discrete Newton works even in the general case.
  - True for our scheduling network, which is why we can use GGT.
- There are other 1-parameter cases where Topkis's structural result applies, see Arai, Ueno, Kajitani; Mc.; Fleischer; Fleischer, Iwata (SFM); Nagano (SFM); Scutellà; Milgrom and Shannon; Granot, Mc., Queyranne, Tardella.
  - In all these cases except Milgrom and Shannon we can also get the GGT-style result that min cuts for all values of the parameter can be computed in $O(1)$ Min Cuts (SFM) time.

Multi-parameter GGT?

- **Corollary:** For our 2-parameter scheduling problem, min cuts are increasing along any non-decreasing curve (chain) in $\mathbb{R}^2_+$.

Multi-parameter GGT?

- **Corollary:** For our 2-parameter scheduling problem, min cuts are increasing along any non-decreasing curve (chain) in $\mathbb{R}^2_+$.
- Cases with two or more parameters still fall into Topkis's framework, but it is not clear so far even what the structure of these cuts looks like.

Multi-parameter GGT?

- **Corollary:** For our 2-parameter scheduling problem, min cuts are increasing along any non-decreasing curve (chain) in $\mathbb{R}^2_+$.
- Cases with two or more parameters still fall into Topkis's framework, but it is not clear so far even what the structure of these cuts looks like.
- Open Question: When capacities are (piecewise) linear, how many different min cuts can we have over all (λ, μ)?

Multi-parameter GGT?

- **Corollary:** For our 2-parameter scheduling problem, min cuts are increasing along any non-decreasing curve (chain) in $\mathbb{R}^2_+$.
- Cases with two or more parameters still fall into Topkis's framework, but it is not clear so far even what the structure of these cuts looks like.
- Open Question: When capacities are (piecewise) linear, how many different min cuts can we have over all (λ, μ)?
- Open Question: How quickly can we compute min cuts in the 2-parameter case?

Any questions?

Questions?

Comments?