

# Discrete Newton Algorithms for Budgeted Network Problems

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

Sauder School of Business, UBC; U. Rome Tor Vergata; Aachen



Aussois 2014



# Discrete Newton Algorithms for Budgeted Network Problems

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

Sauder School of Business, UBC; U. Rome Tor Vergata; Aachen



Aussois 2014

**S. Thomas McCormick**  
Sauder School of Business  
University of British Columbia

# Outline

## 1 Prelude

- A motivating example
- Interdiction curves
- Dual of interdiction
- Parametric curves

# Outline

## 1 Prelude

- A motivating example
- Interdiction curves
- Dual of interdiction
- Parametric curves

## 2 The Slope Problem

- What is it?
- Algorithms
- Newton- $B$

# Outline

## 1 Prelude

- A motivating example
- Interdiction curves
- Dual of interdiction
- Parametric curves

## 2 The Slope Problem

- What is it?
- Algorithms
- Newton- $B$

## 3 Maximum Robust Flow

- What is it?

# Outline

- 1 Prelude
  - A motivating example
  - Interdiction curves
  - Dual of interdiction
  - Parametric curves
- 2 The Slope Problem
  - What is it?
  - Algorithms
  - Newton- $B$
- 3 Maximum Robust Flow
  - What is it?
- 4 Multiple Parameters
  - Multi-parametric scheduling problems

# Outline

- 1 Prelude
  - A motivating example
  - Interdiction curves
  - Dual of interdiction
  - Parametric curves
- 2 The Slope Problem
  - What is it?
  - Algorithms
  - Newton- $B$
- 3 Maximum Robust Flow
  - What is it?
- 4 Multiple Parameters
  - Multi-parametric scheduling problems

# What is Network Interdiction?

Our motivating budgeted network problem is [network interdiction](#).



# What is Network Interdiction?

Our motivating budgeted network problem is **network interdiction**.

Consider an ordinary max flow network:



# What is Network Interdiction?

Our motivating budgeted network problem is **network interdiction**.

Now suppose that an **interdictor** can damage the network:



# What is Network Interdiction?

Our motivating budgeted network problem is **network interdiction**.

Now suppose that an **interdictor** can damage the network:



In general we have a **flow player** who wants to push as much post-damage flow as possible through the network ...

# What is Network Interdiction?

Our motivating budgeted network problem is **network interdiction**.

Now suppose that an **interdictor** can damage the network:



In general we have a **flow player** who wants to push as much post-damage flow as possible through the network ...

... and an **interdiction player** who wants to damage the network by removing arcs subject to a **budget** so as to minimize the post-removal flow.

# The Formal Network Interdiction Model

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

# The Formal Network Interdiction Model

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

# The Formal Network Interdiction Model

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

# The Formal Network Interdiction Model

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



# The Formal Network Interdiction Model

- We start with an ordinary max flow min cut network with source  $s$ , sink  $t$ , and capacities  $c$ . The capacity of cut  $S$  is  $\text{cap}_c(S)$ .
- We have a second non-negative datum on each arc:  $r_{ij}$  is the **removal cost** of destroying arc  $i \rightarrow j$ ; we could spend, e.g.,  $r_{ij}/2$  to reduce the capacity of  $i \rightarrow j$  to  $c_{ij}/2$ .
  - In Min Cut we assume that the removal cost of  $i \rightarrow j$  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  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , the interdiction value is 0.

## Removing arcs greedily

- Thus if  $B = 0$ , then the interdiction value is  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , 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  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , 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 a removal to a cut to remove more flow.

# Removing arcs greedily

- Thus if  $B = 0$ , then the interdiction value is  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , 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 a removal 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  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , 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 a removal 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: we could use a pairwise interchange argument.

# Removing arcs greedily

- Thus if  $B = 0$ , then the interdiction value is  $\text{cap}_c^*$ , the ordinary min cut value; for  $B \geq \text{cap}_r^*$ , 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 a removal 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: 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  $S$ .

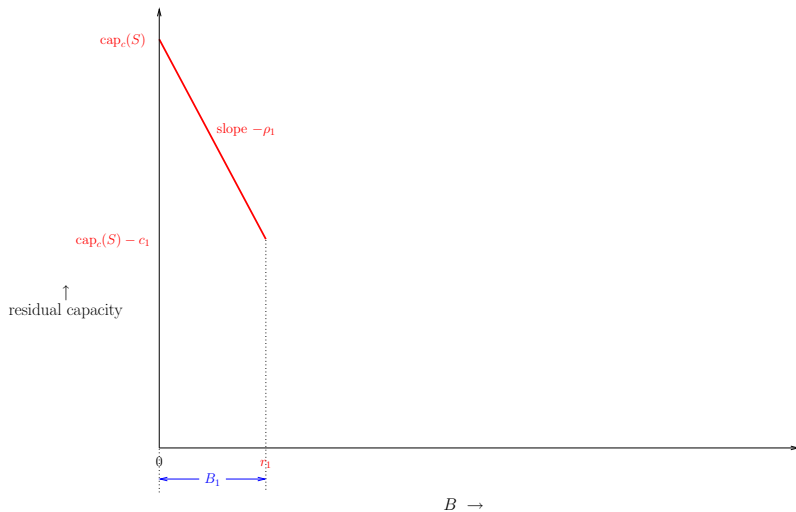
# The interdiction curve for a fixed cut $S$

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



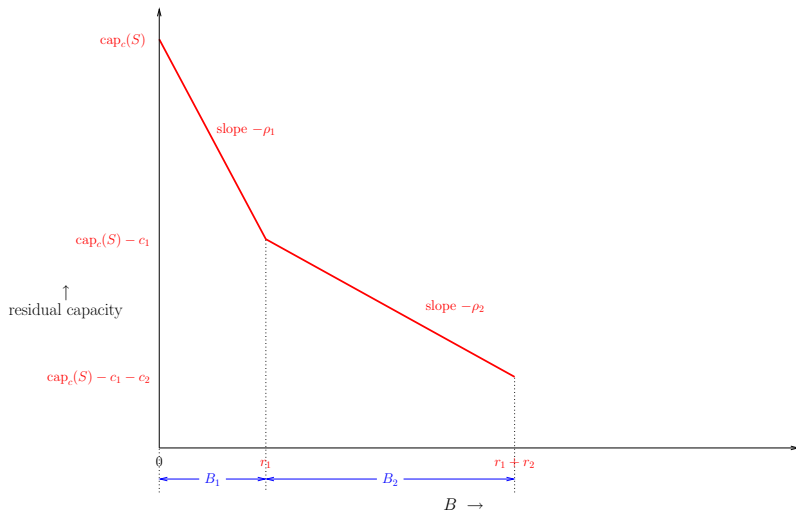
# The interdiction curve for a fixed cut $S$

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



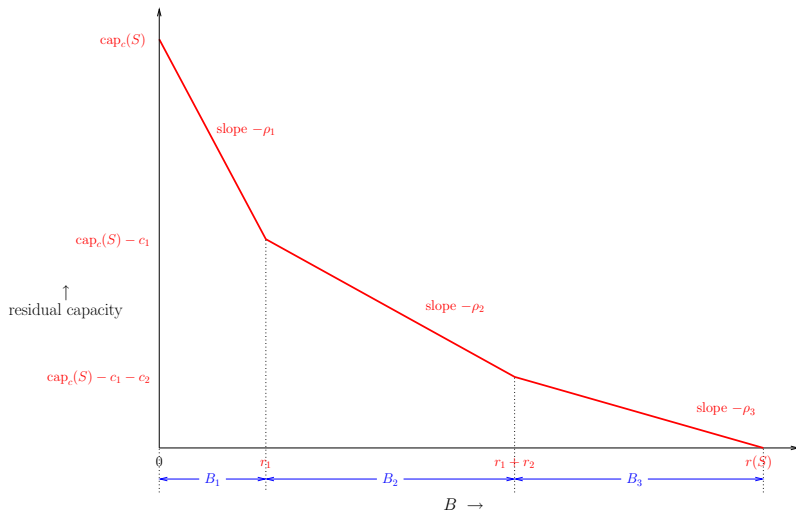
# The interdiction curve for a fixed cut $S$

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



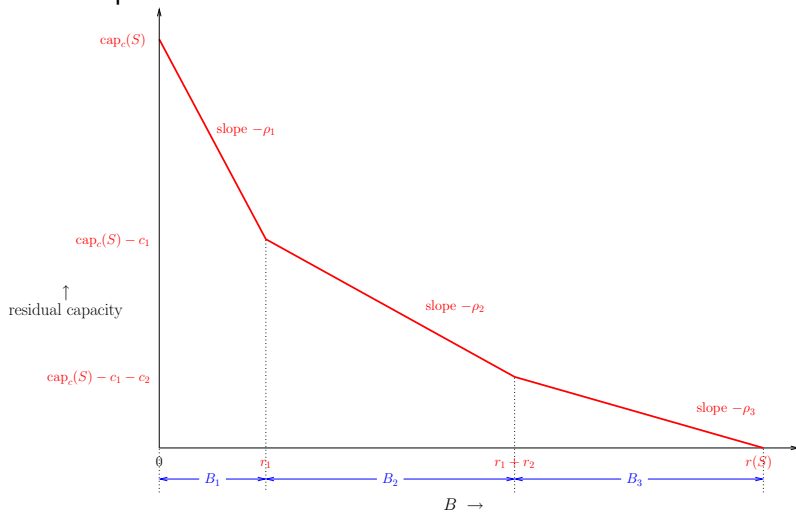
# The interdiction curve for a fixed cut $S$

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.

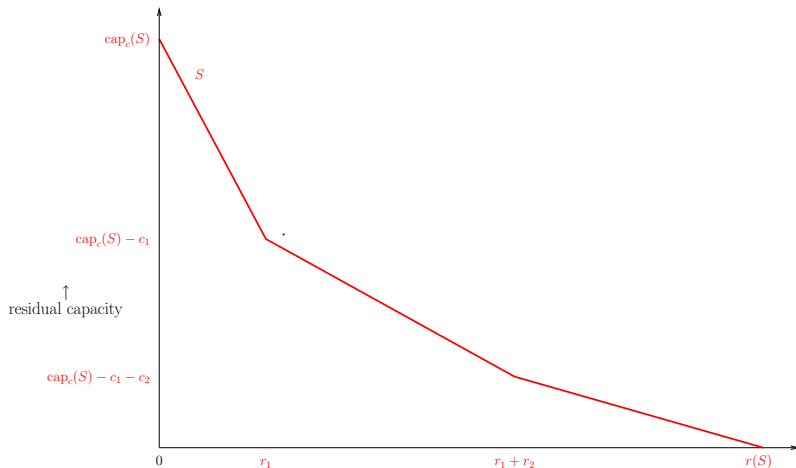


# The overall interdiction curve: the $B$ -profile

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

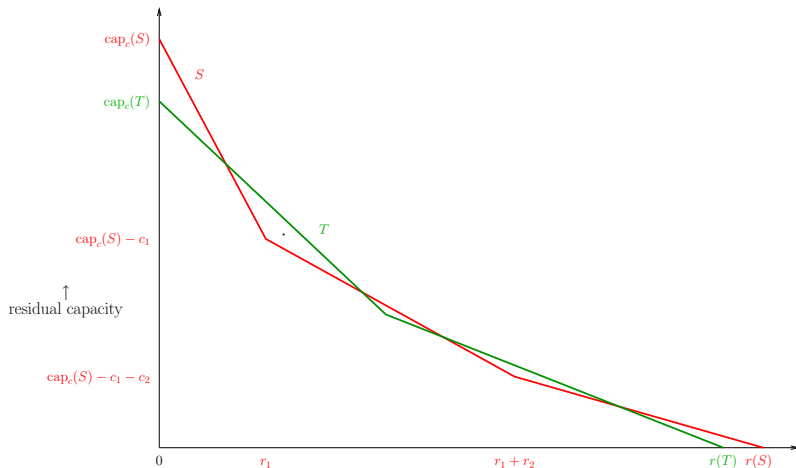
# The overall interdiction curve: the $B$ -profile

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



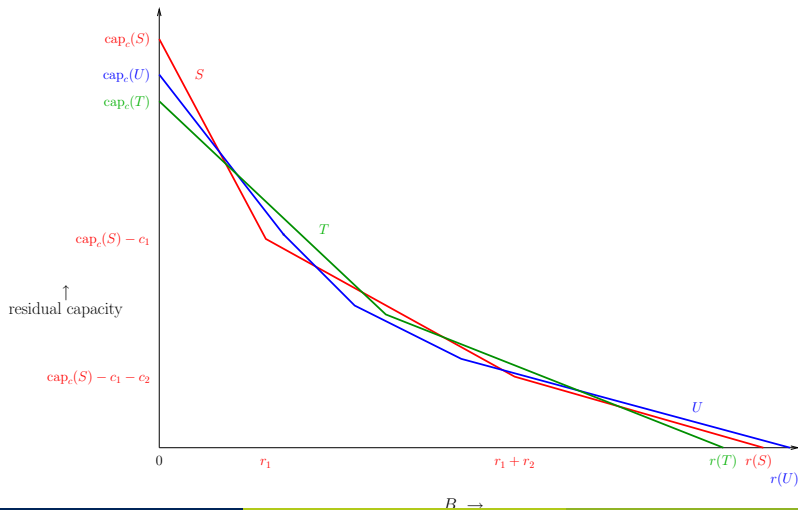
# The overall interdiction curve: the $B$ -profile

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



# The overall interdiction curve: the $B$ -profile

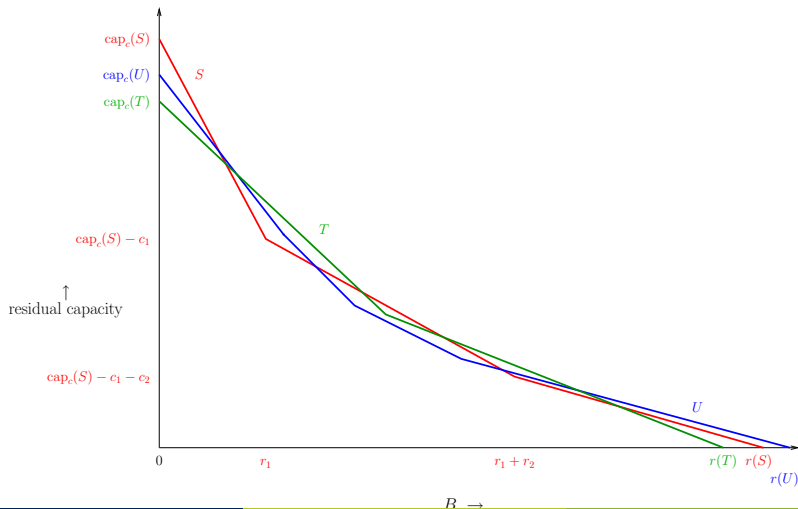
We overlay the cut-wise interdiction curves to get the overall curve.





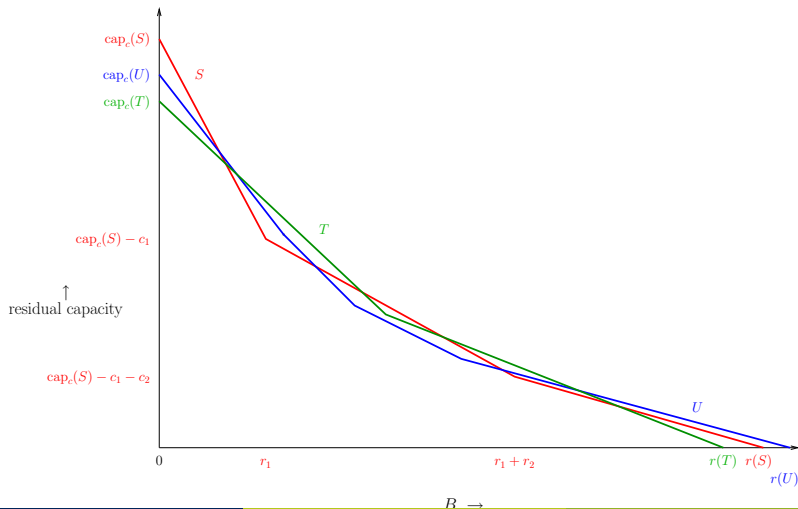
# The overall interdiction curve: the $B$ -profile

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.



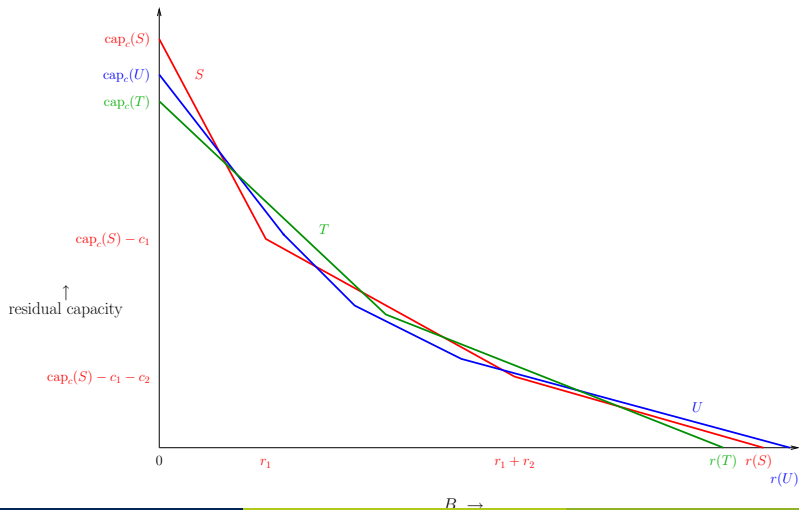
# The overall interdiction curve: the $B$ -profile

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



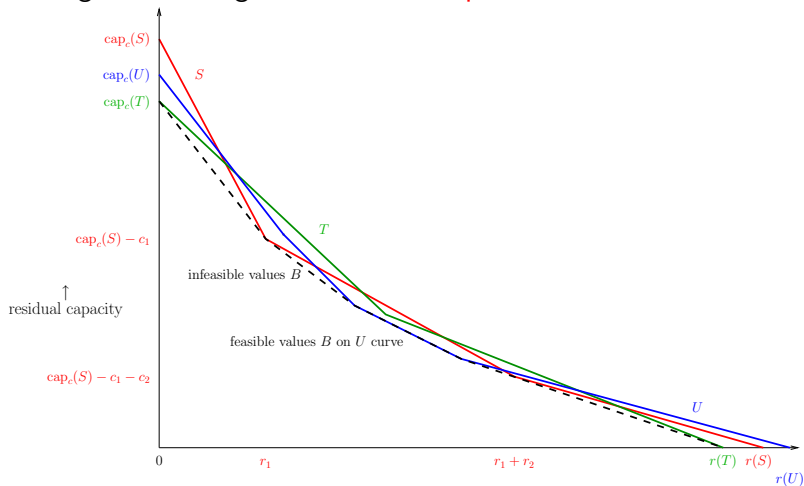
# The overall interdiction curve: the $B$ -profile

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



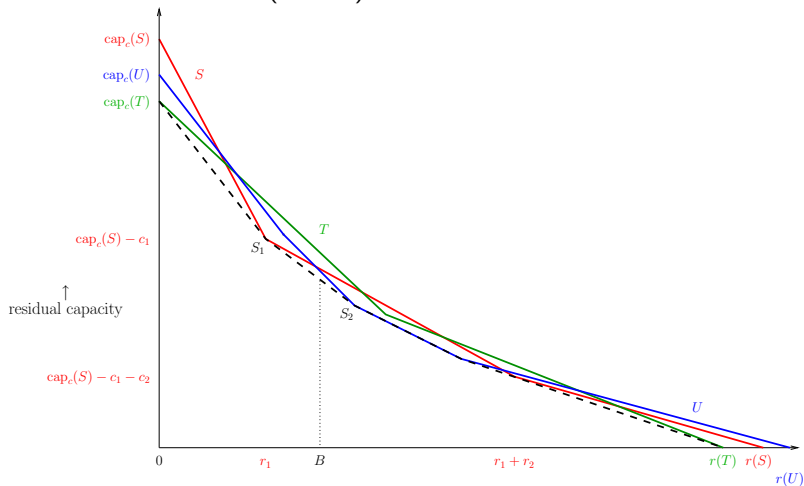
# The overall interdiction curve: the $B$ -profile

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



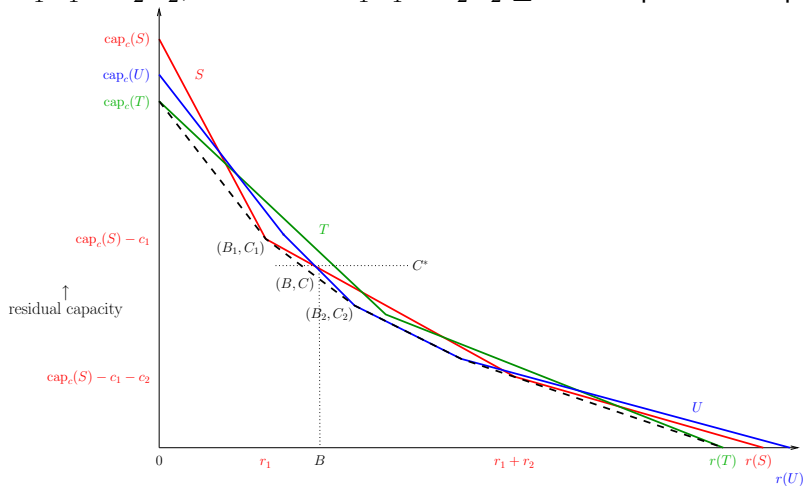
# The overall interdiction curve: the $B$ -profile

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



# 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  $\lambda$  s.t.  $B = \lambda_1 B_1 + \lambda_2 B_2$ ; define  $C = \lambda_1 C_1 + \lambda_2 C_2 \leq C^* = \text{opt. resid. capacity}$ .



# The linear program and its dual

- The normal min cut dual LP is

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

# The linear program and its dual

- The normal min cut dual LP is

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

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



# The linear program and its dual

- The normal min cut dual LP is

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

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

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

# The linear program and its dual

- The normal min cut dual LP is

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

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

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

- Prize-collecting with a budget in place of a penalty.

# The linear program and its dual

- Repeat the new LP with **dual variables**:

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

# The linear program and its dual

- Repeat the new LP with **dual variables**:

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

- When we “primalize” this interdiction dual LP we get new primal variable  $\lambda$  corresponding to the dual constraint  $\sum_{u \rightarrow v} r_{uv} z_{uv} \leq B$ , 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**:

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

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

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

# The linear program and its dual

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

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

# The linear program and its dual

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

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

- The two capacity constraints simplify into

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

a *parametric capacity* in the scalar parameter  $\lambda$ .

# The linear program and its dual

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

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

- The two capacity constraints simplify into

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

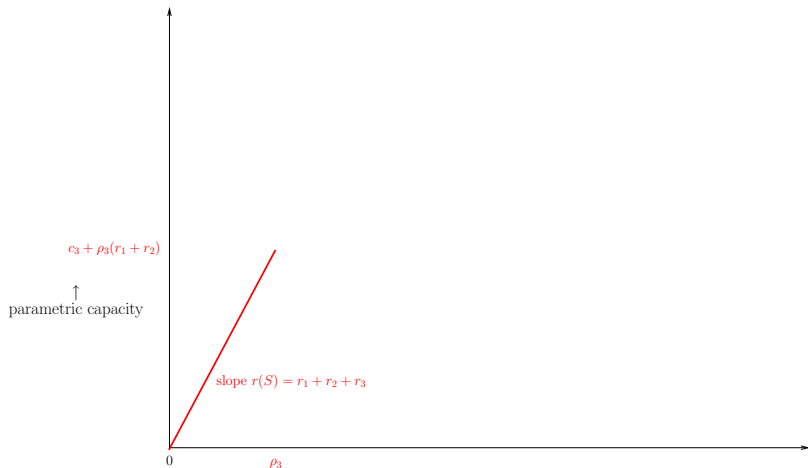
a *parametric capacity* in the scalar parameter  $\lambda$ .

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



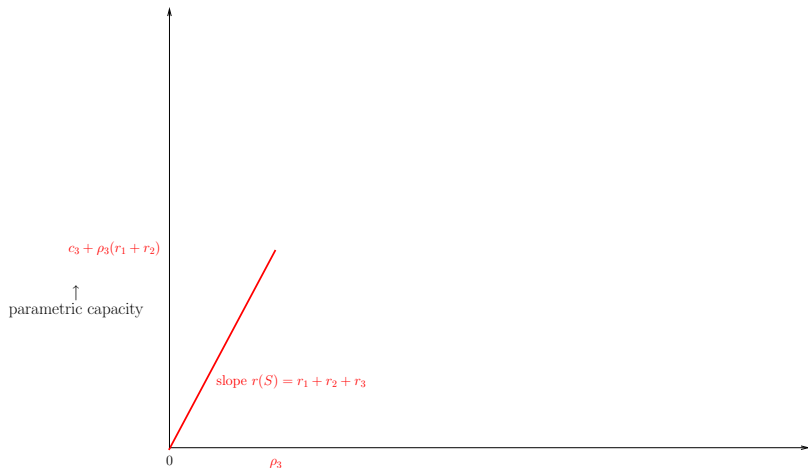
# Parametric capacity of fixed cut $S$

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



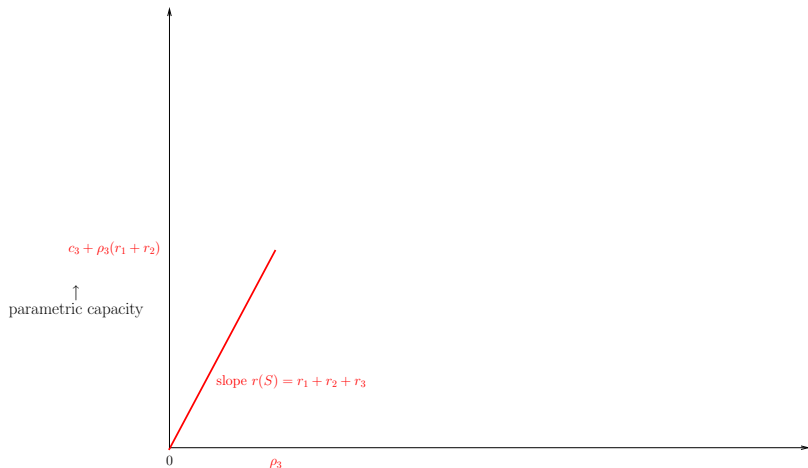
# Parametric capacity of fixed cut $S$

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



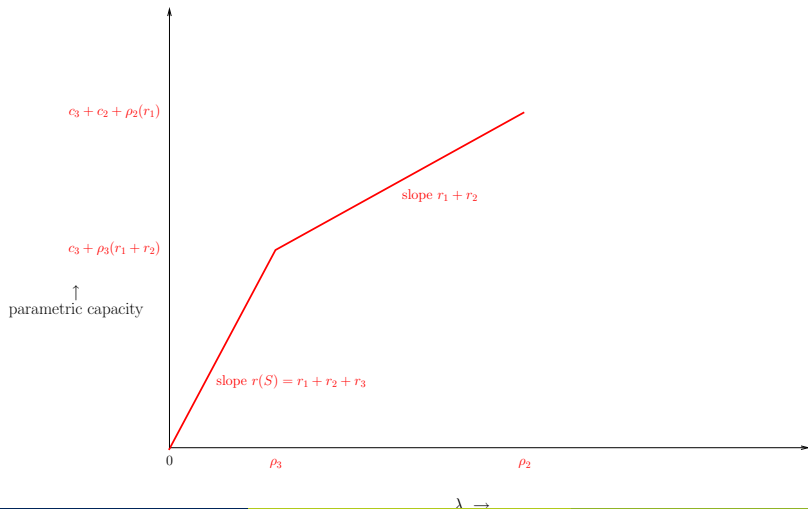
# Parametric capacity of fixed cut $S$

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



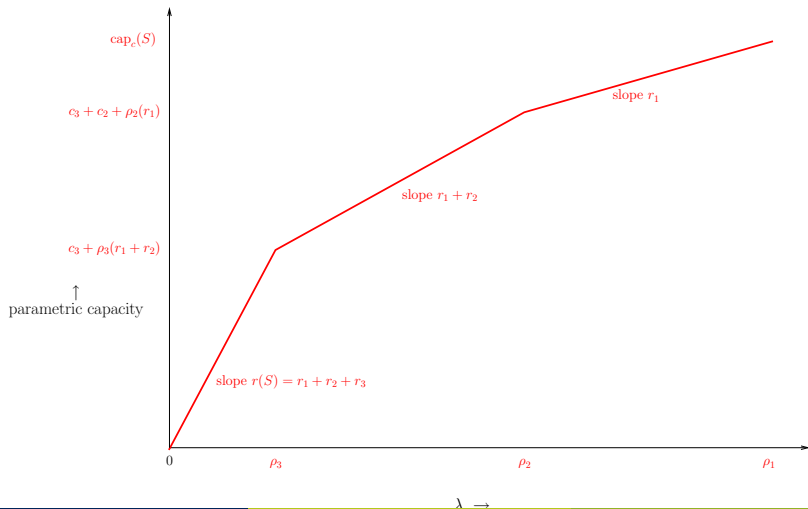
# Parametric capacity of fixed cut $S$

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



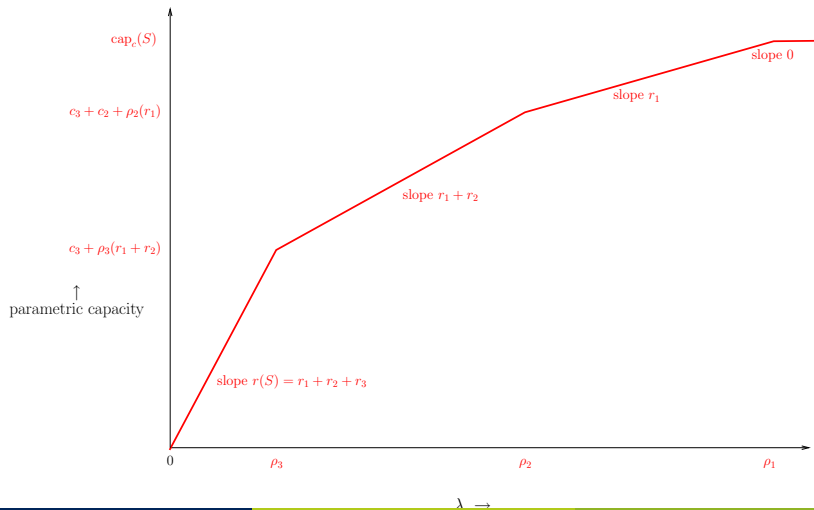
# Parametric capacity of fixed cut $S$

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



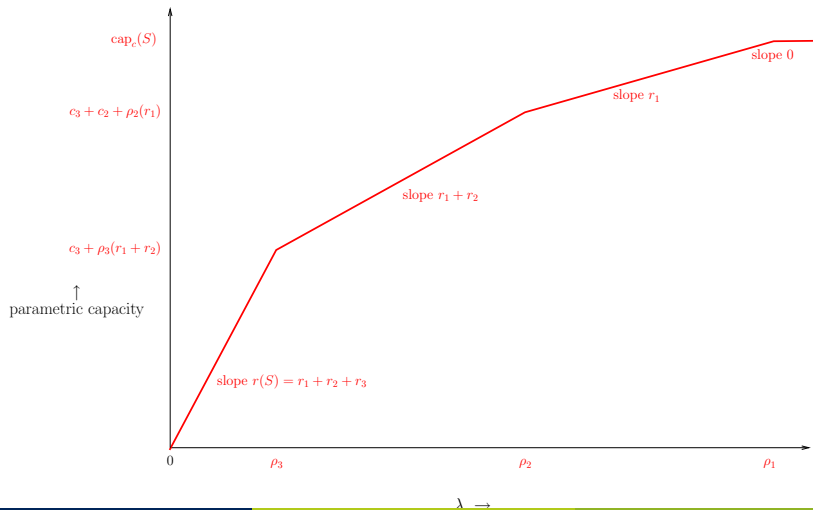
# Parametric capacity of fixed cut $S$

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



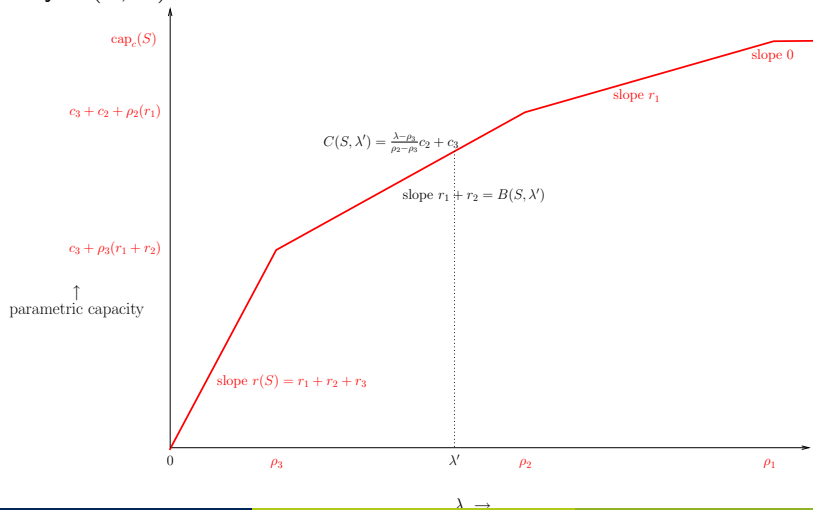
# Parametric capacity of fixed cut $S$

The parametric capacity curve for  $S$  is piecewise linear concave.



# Parametric capacity of fixed cut $S$

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



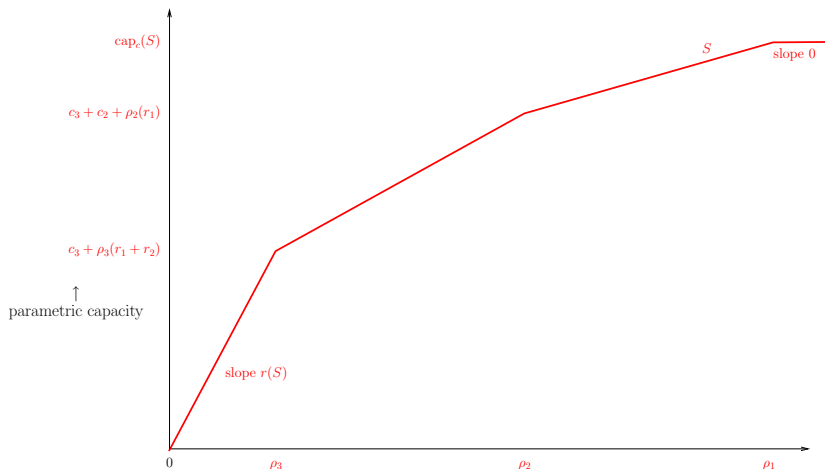


# The overall parametric capacity curve: the $\lambda$ -profile

Now overlay the parametric capacity curves for all  $S$ .

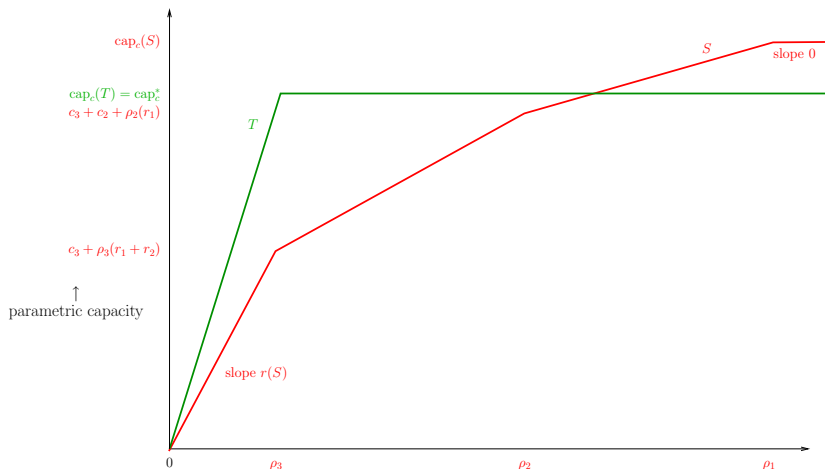
# The overall parametric capacity curve: the $\lambda$ -profile

Now overlay the parametric capacity curves for all  $S$ .



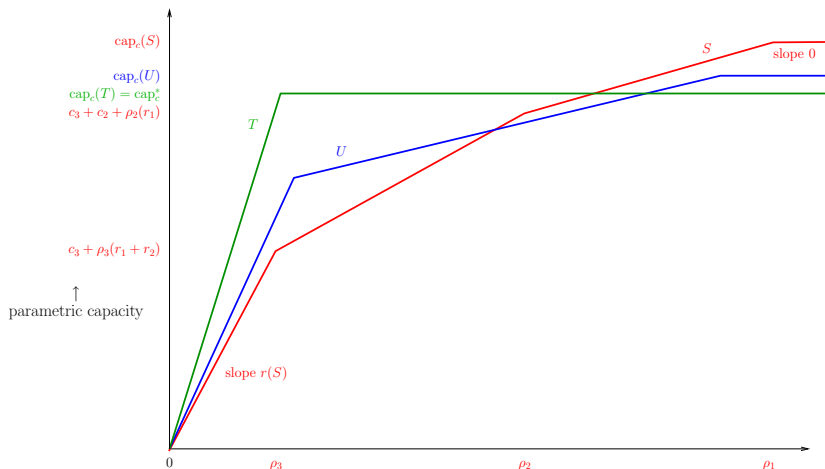
# The overall parametric capacity curve: the $\lambda$ -profile

Now overlay the parametric capacity curves for all  $S$ .



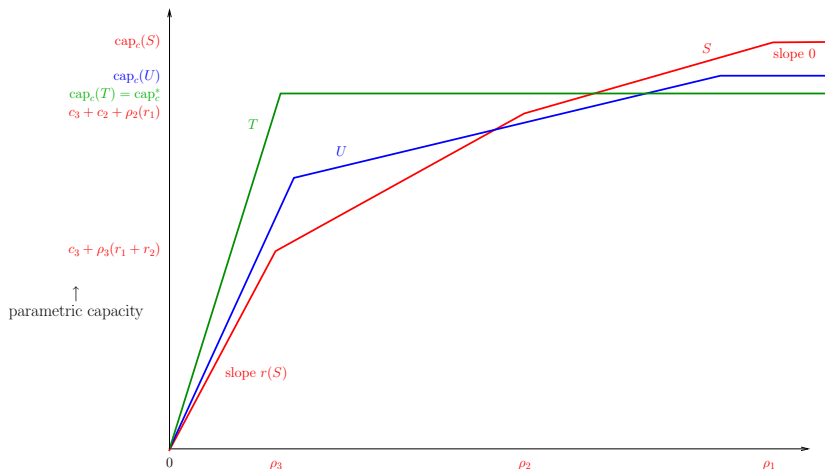
# The overall parametric capacity curve: the $\lambda$ -profile

Now overlay the parametric capacity curves for all  $S$ .



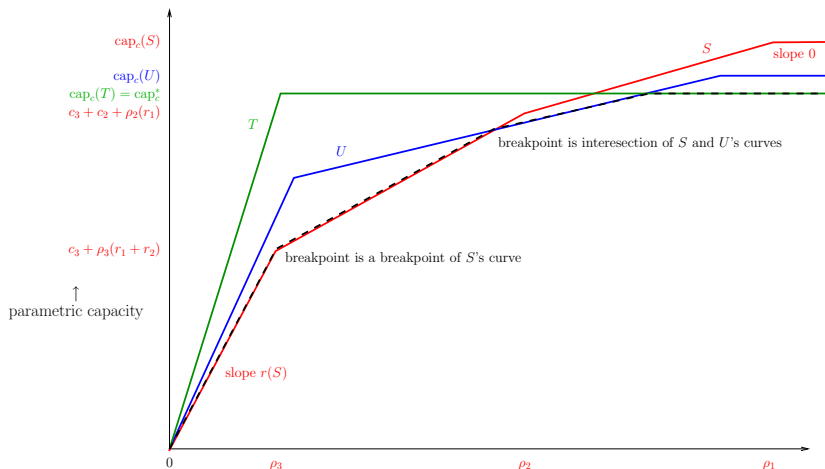
# The overall parametric capacity curve: the $\lambda$ -profile

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



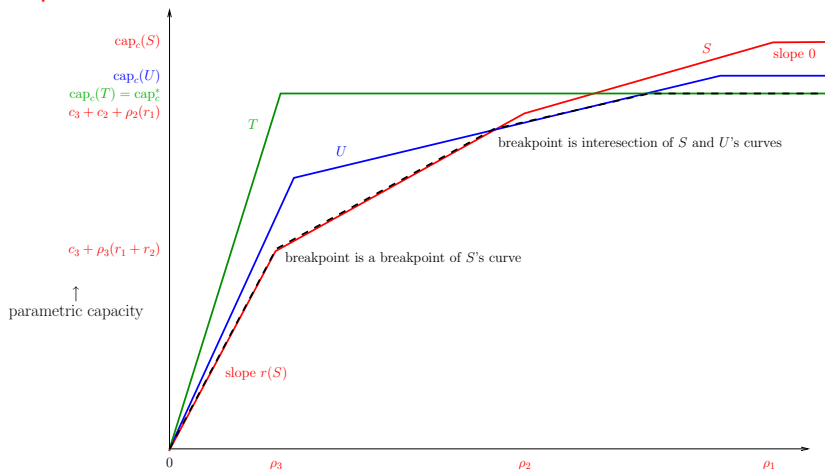
# The overall parametric capacity curve: the $\lambda$ -profile

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



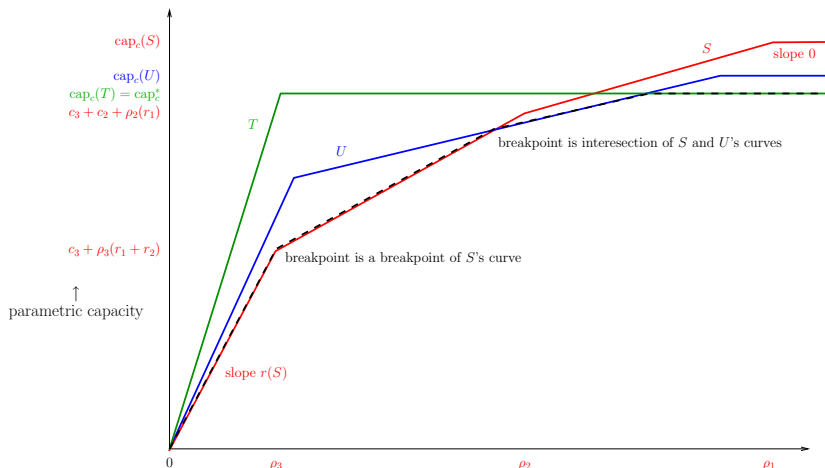
# The overall parametric capacity curve: the $\lambda$ -profile

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  $\lambda$ -profile.



# The overall parametric capacity curve: the $\lambda$ -profile

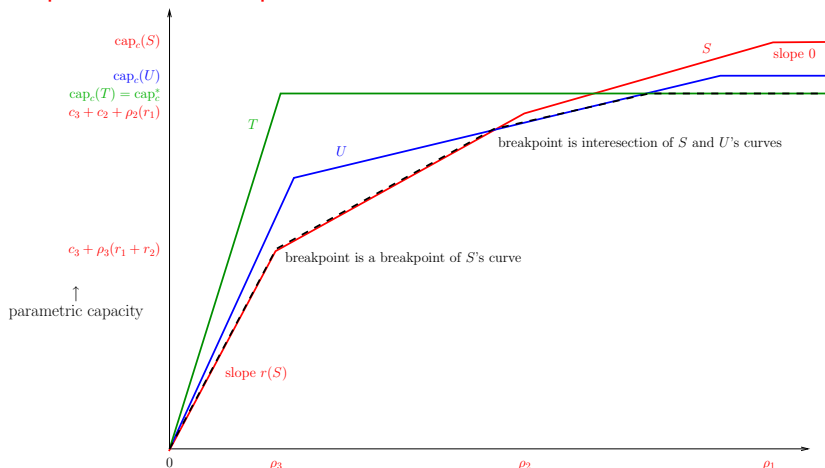
We can compute things like  $\text{cap}^*(\lambda)$  easily using parametric min cut technology.





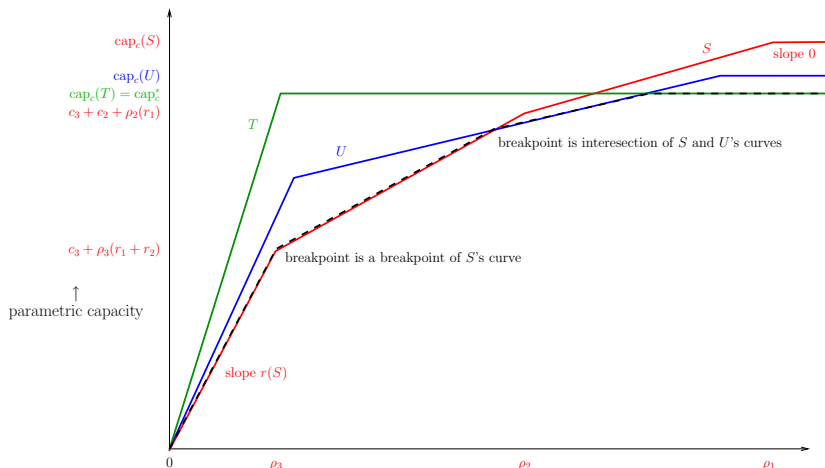
# The overall parametric capacity curve: the $\lambda$ -profile

There is **conjugate duality** between  $S$ 's interdiction and parametric capacity curves; convex duality shows this carries over to **conjugate duality between the  $B$ -profile and the  $\lambda$ -profile**.



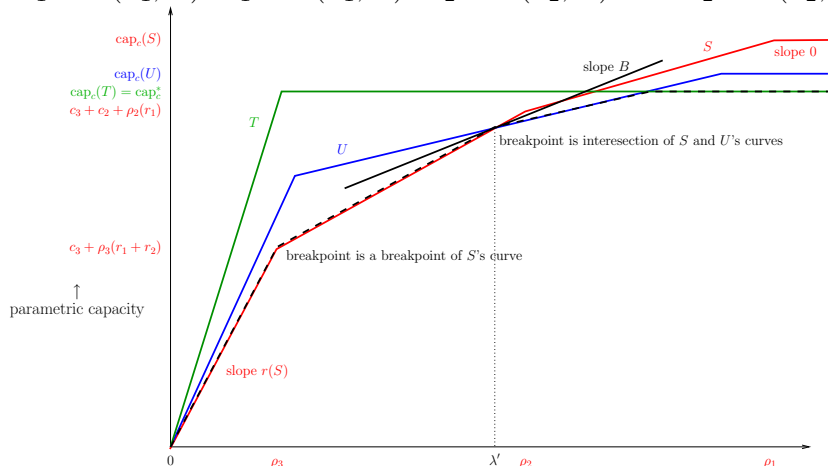
# The overall parametric capacity curve: the $\lambda$ -profile

The Burch et al. pseudo-approximation framework asks: for a given  $B$ , compute the two cuts  $S_1$  and  $S_2$  bracketing  $B$  on the  $B$ -profile.



# The overall parametric capacity curve: the $\lambda$ -profile

Conjugate duality implies that this is equivalent to finding a breakpoint  $\lambda'$  on the  $\lambda$ -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 Prelude
  - A motivating example
  - Interdiction curves
  - Dual of interdiction
  - Parametric curves
- 2 The Slope Problem
  - What is it?
  - Algorithms
  - Newton- $B$
- 3 Maximum Robust Flow
  - What is it?
- 4 Multiple Parameters
  - Multi-parametric scheduling problems

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.



# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call **Newton- $B$** .

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call **Newton- $B$** .
- This will give fast, combinatorial algorithms for:

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call **Newton- $B$** .
- This will give fast, combinatorial algorithms for:
  - 1 Network Interdiction (and more general interdiction problems).

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call *Newton- $B$* .
- This will give fast, combinatorial algorithms for:
  - ① Network Interdiction (and more general interdiction problems).
  - ② Reverse Network Interdiction (optimally buy capacity).

# The key Slope Problem

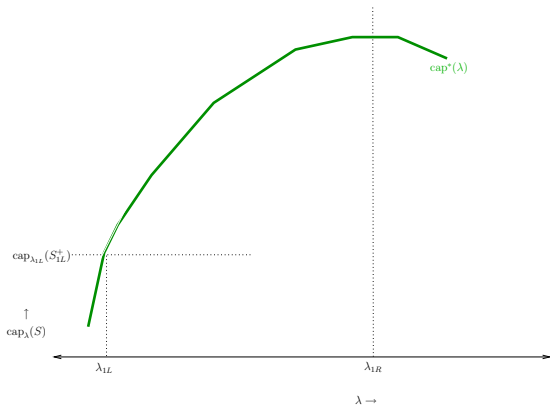
- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call *Newton- $B$* .
- This will give fast, combinatorial algorithms for:
  - 1 Network Interdiction (and more general interdiction problems).
  - 2 Reverse Network Interdiction (optimally buy capacity).
  - 3 Maximum Robust Flow (see later).

# The key Slope Problem

- We are given a parametric curve such as the  $\lambda$ -profile  $\text{cap}^*(\lambda)$  and a budget  $B$ .
- **The Slope Problem:** Find a breakpoint of the curve such that the slopes of the two adjacent segments bracket  $B$ .
- If we can solve the Slope Problem then we get a pseudo-approx. for Network Interdiction from a framework by Burch et al.
  - Burch et al. already had an LP-based way to solve the Slope Problem, but we want a **combinatorial** algorithm.
- We will use a discrete version of Newton's Algorithm that we call *Newton- $B$* .
- This will give fast, combinatorial algorithms for:
  - 1 Network Interdiction (and more general interdiction problems).
  - 2 Reverse Network Interdiction (optimally buy capacity).
  - 3 Maximum Robust Flow (see later).
  - 4 Various scheduling problems with multiple budgets.

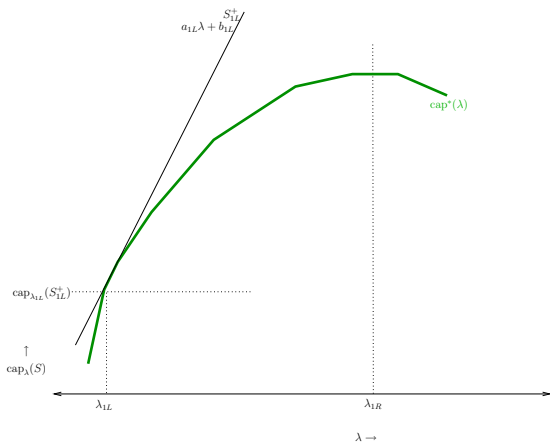
# Newton- $B$ gives a better algorithm

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



# Newton-B gives a better algorithm

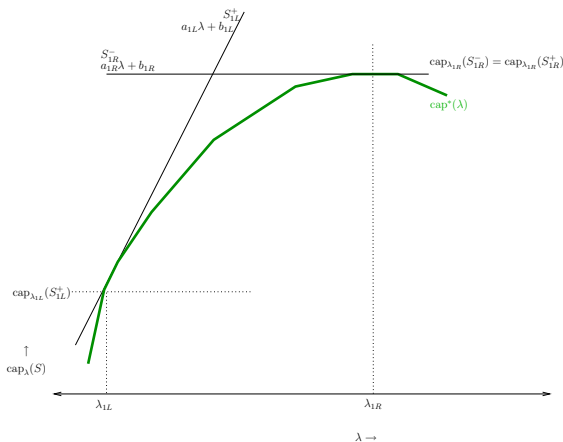
Set  $\lambda_L = 0$  and  $\lambda_R = \text{cap}_r^*$  as in Binary Search. Denote  $\text{sl}^+(\lambda_L)$  by  $\text{sl}_L^+$  and  $\text{sl}^-(\lambda_R)$  by  $\text{sl}_R^-$ .





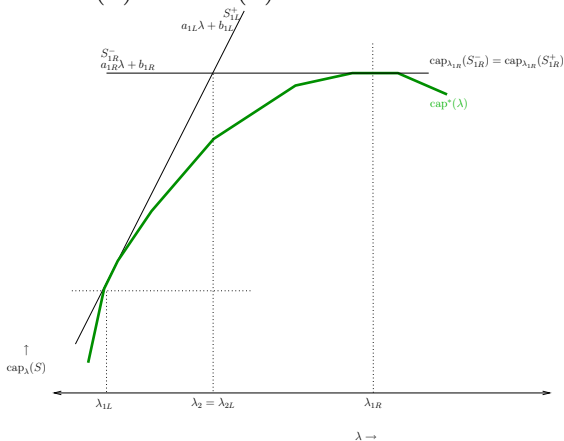
# Newton-B gives a better algorithm

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



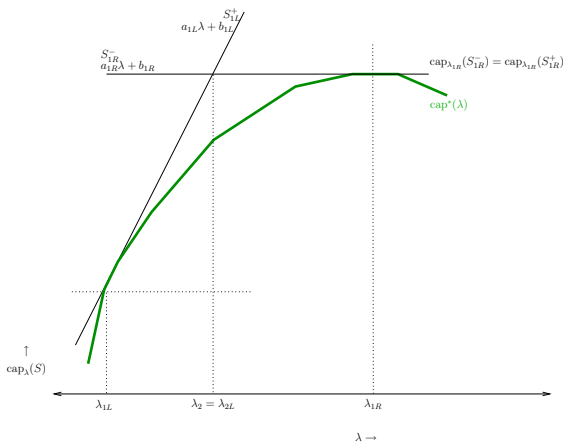
# Newton-B gives a better algorithm

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



# Newton- $B$ gives a better algorithm

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





# The key inequality

- The analysis flows from a key inequality that says that each iteration cuts down the gap between our current slope and our target slope by a factor of at least 2, or the gap between our current vertical position and an upper bound on the vertical position of the optimal breakpoint by a factor of at least 2. Thus Newton- $B$  is never worse than Binary Search.

# The key inequality

- The analysis flows from a key inequality that says that each iteration cuts down the gap between our current slope and our target slope by a factor of at least 2, or the gap between our current vertical position and an upper bound on the vertical position of the optimal breakpoint by a factor of at least 2. Thus Newton- $B$  is never worse than Binary Search.
- This was originally proved in Mc+Evolina '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

- The analysis flows from a key inequality that says that each iteration cuts down the gap between our current slope and our target slope by a factor of at least 2, or the gap between our current vertical position and an upper bound on the vertical position of the optimal breakpoint by a factor of at least 2. Thus Newton- $B$  is never worse than Binary Search.
- This 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

- The analysis flows from a key inequality that says that each iteration cuts down the gap between our current slope and our target slope by a factor of at least 2, or the gap between our current vertical position and an upper bound on the vertical position of the optimal breakpoint by a factor of at least 2. Thus Newton- $B$  is never worse than Binary Search.
- This 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  $O\left(\frac{\log(nD)}{1+\log \log(nD)-\log \log n}\right)$ .
  - There is also an  $O(m^2 \log^2 n)$  bound on the number of iterations.



# The key inequality

- The analysis flows from a key inequality that says that each iteration cuts down the gap between our current slope and our target slope by a factor of at least 2, or the gap between our current vertical position and an upper bound on the vertical position of the optimal breakpoint by a factor of at least 2. Thus Newton- $B$  is never worse than Binary Search.
- This 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  $O\left(\frac{\log(nD)}{1+\log \log(nD)-\log \log n}\right)$ .
  - There is also an  $O(m^2 \log^2 n)$  bound on the number of iterations.
- Thus Newton- $B$  is never worse than Binary Search, and has better weakly and strongly polynomial bounds.

## 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; we can, e.g., either get an under-budget cut whose greedy partial deletion gives at most twice the optimal residual flow, or a cut whose greedy deletion costs at most  $2B$  but which gives at most the optimal residual flow.

## 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 can, e.g., either get an under-budget cut whose greedy partial deletion gives at most twice the optimal residual flow, or a cut whose greedy deletion costs at most  $2B$  but which gives at most the optimal residual flow.
  - 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 can, e.g., either get an under-budget cut whose greedy partial deletion gives at most twice the optimal residual flow, or a cut whose greedy deletion costs at most  $2B$  but which gives at most the optimal residual flow.
  - 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

- 1 Prelude
  - A motivating example
  - Interdiction curves
  - Dual of interdiction
  - Parametric curves
- 2 The Slope Problem
  - What is it?
  - Algorithms
  - Newton- $B$
- 3 Maximum Robust Flow
  - What is it?
- 4 Multiple Parameters
  - Multi-parametric scheduling problems



# Interdiction and Robust Flows

- Who moves first?

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF)** problem:

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$**  and **fractional deletion** is allowed.

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$  and fractional deletion** is allowed.
    - Thus interdictor's set of strategies is  $\mathcal{X} = \{\mu_e \in [0, 1]^E \mid r^T \mu \leq B\}$ .

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$  and fractional deletion** is allowed.
    - Thus interdictor's set of strategies is  $\mathcal{X} = \{\mu_e \in [0, 1]^E \mid r^T \mu \leq B\}$ .
    - The flow player **is not allowed to re-route**  $f$  after the interdictor acts.



# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$  and fractional deletion** is allowed.
    - Thus interdictor's set of strategies is  $\mathcal{X} = \{\mu_e \in [0, 1]^E \mid r^T \mu \leq B\}$ .
    - The flow player **is not allowed to re-route**  $f$  after the interdictor acts.
    - She **specifies a path decomposition** for  $f$ , so her set of strategies is:
 
$$\mathcal{F} = \{(f_P)_{P \in \mathcal{P}} \mid \sum_{P \in \mathcal{P}: e \in P} f_P \leq c_e, \forall e \in E\}.$$

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$**  and **fractional deletion** is allowed.
    - Thus interdictor's set of strategies is  $\mathcal{X} = \{\mu_e \in [0, 1]^E \mid r^T \mu \leq B\}$ .
    - The flow player **is not allowed to re-route**  $f$  after the interdictor acts.
    - She **specifies a path decomposition** for  $f$ , so her set of strategies is:  $\mathcal{F} = \{(f_P)_{P \in \mathcal{P}} \mid \sum_{P \in \mathcal{P}: e \in P} f_P \leq c_e, \forall e \in E\}$ .
    - Thus: if the flow player chooses a path-flow  $f \in \mathcal{F}$  and the interdictor  $\mu \in \mathcal{X}$ , then, for each  $P \in \mathcal{P}$ , the amount of flow deleted by the interdictor is  $f_P \cdot \max_{e \in P} \mu_e$

# Interdiction and Robust Flows

- Who moves first?
  - Network interdiction: **interdictor** moves **first**, then the flow player acts
  - Robust flow: **flow player** moves **first**, then the interdictor acts
- The **Maximum Robust Flow (MRF) problem**:
  - Choose a flow  $f$  to maximize the amount of  $f$  reaching the sink  $t$ , no matter how an interdictor acts when:
    - The interdictor has **budget  $B$**  and **fractional deletion** is allowed.
    - Thus interdictor's set of strategies is  $\mathcal{X} = \{\mu_e \in [0, 1]^E \mid r^T \mu \leq B\}$ .
    - The flow player **is not allowed to re-route**  $f$  after the interdictor acts.
    - She **specifies a path decomposition** for  $f$ , so her set of strategies is:
 
$$\mathcal{F} = \{(f_P)_{P \in \mathcal{P}} \mid \sum_{P \in \mathcal{P}: e \in P} f_P \leq c_e, \forall e \in E\}.$$
    - Thus: if the flow player chooses a path-flow  $f \in \mathcal{F}$  and the interdictor  $\mu \in \mathcal{X}$ , then, for each  $P \in \mathcal{P}$ , the amount of flow deleted by the interdictor is  $f_P \cdot \max_{e \in P} \mu_e$
  - Choose a **path-flow  $f \in \mathcal{F}$**  of maximum robust value  $\text{Rval}(f) := \text{val}(f) - z(f)$ , where
 
$$z(f) := \max_{\mu \in \mathcal{X}} (\sum_{P \in \mathcal{P}} f_P \cdot \max_{e \in P} \mu_e)$$
 is the maximum amount of  $f$  the interdictor can delete using a budget of  $B$

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];
  - 3 there exists a pseudo-approximation algorithm for any given integer  $B$  [Bertsimas, Nasrabadi, and Stiller 2013]

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];
  - 3 there exists a pseudo-approximation algorithm for any given integer  $B$  [Bertsimas, Nasrabadi, and Stiller 2013]
- Our contributions:



# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];
  - 3 there exists a pseudo-approximation algorithm for any given integer  $B$  [Bertsimas, Nasrabadi, and Stiller 2013]
- Our contributions:
  - 2a The problem is hard when  $B = 2$ , even if **fractional interdiction** is allowed;

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];
  - 3 there exists a pseudo-approximation algorithm for any given integer  $B$  [Bertsimas, Nasrabadi, and Stiller 2013]
- Our contributions:
  - 2a The problem is hard when  $B = 2$ , even if **fractional interdiction** is allowed;
  - 2b There exists a pseudo-approximation algorithm, even when **removal costs are non-uniform** (and fractional interdiction is not allowed).

# The Maximum Robust Flow problem

- MRF has been studied with **uniform removal cost** ( $r_e = 1 \forall e \in E$ ) **and integral interdiction**, i.e., when the set of strategies for the interdictor is  $\mathcal{X} = \{\mu_e \in \{0, 1\}^E \mid \sum_{e \in E} \mu_e \leq B\}$ . In this case:
  - 1 the problem is easy when  $B = 1$  [Aneja, Nair, Chandrasekaran, 2001];
  - 2 the problem is hard when  $B = 2$  [Du, Chandrasekaran, 2007];
  - 3 there exists a pseudo-approximation algorithm for any given integer  $B$  [Bertsimas, Nasrabadi, and Stiller 2013]
- Our contributions:
  - 2a The problem is hard when  $B = 2$ , even if **fractional interdiction** is allowed;
  - 2b There exists a pseudo-approximation algorithm, even when **removal costs are non-uniform** (and fractional interdiction is not allowed).
    - Algorithm: compute any path decomposition of the optimal solution to the Slope Problem

# Algorithm for the Maximum Robust Flow problem

- The maximum robust flow problem: choose a path-flow  $f \in \mathcal{F}$  of maximum robust value  $\text{Rval}(f)$

# Algorithm for the Maximum Robust Flow problem

- The **maximum robust flow problem**: choose a path-flow  $f \in \mathcal{F}$  of maximum robust value  $\text{Rval}(f)$
- A **pseudo-approximation algorithm**: Let  $\bar{x}, \bar{\lambda}$  be an optimal solution to the Slope Problem

$$\begin{array}{ll}
 \max_{x, \lambda \geq 0} & \text{val}(x) - B\lambda \\
 \text{s.t.} & \text{conservation} \\
 & x_e \leq \min\{u_e, r_e \lambda\} \quad \forall e \in E
 \end{array}$$

and let  $\bar{f} = \{\bar{f}_P, P \in \mathcal{P}\}$  be a path-decomposition of  $\bar{x}$ . Also, let  $f^*$  be a flow with maximum robust value  $\text{Rval}(f^*)$ . Then the following hold:

where  $\rho = 1 - (1 - \frac{1}{k})^k$  and  $k = \max_{P \in \mathcal{P}} |E(P)|$ .

# Algorithm for the Maximum Robust Flow problem

- The maximum robust flow problem: choose a path-flow  $f \in \mathcal{F}$  of maximum robust value  $\text{Rval}(f)$
- A pseudo-approximation algorithm: Let  $\bar{x}, \bar{\lambda}$  be an optimal solution to the Slope Problem

$$\begin{array}{ll}
 \max_{x, \lambda \geq 0} & \text{val}(x) - B\lambda \\
 \text{s.t.} & \text{conservation} \\
 & x_e \leq \min\{u_e, r_e \lambda\} \quad \forall e \in E
 \end{array}$$

and let  $\bar{f} = \{\bar{f}_P, P \in \mathcal{P}\}$  be a path-decomposition of  $\bar{x}$ . Also, let  $f^*$  be a flow with maximum robust value  $\text{Rval}(f^*)$ . Then the following hold:

$$(i) \quad \text{Rval}(\bar{f}) \geq \text{val}(\bar{x}) - B\bar{\lambda}$$

where  $\rho = 1 - (1 - \frac{1}{k})^k$  and  $k = \max_{P \in \mathcal{P}} |E(P)|$ .

# Algorithm for the Maximum Robust Flow problem

- The maximum robust flow problem: choose a path-flow  $f \in \mathcal{F}$  of maximum robust value  $\text{Rval}(f)$
- A pseudo-approximation algorithm: Let  $\bar{x}, \bar{\lambda}$  be an optimal solution to the Slope Problem

$$\begin{array}{ll} \max_{x, \lambda \geq 0} & \text{val}(x) - B\lambda \\ \text{s.t.} & \text{conservation} \\ & x_e \leq \min\{u_e, r_e \lambda\} \quad \forall e \in E \end{array}$$

and let  $\bar{f} = \{\bar{f}_P, P \in \mathcal{P}\}$  be a path-decomposition of  $\bar{x}$ . Also, let  $f^*$  be a flow with maximum robust value  $\text{Rval}(f^*)$ . Then the following hold:

- (i)  $\text{Rval}(\bar{f}) \geq \text{val}(\bar{x}) - B\bar{\lambda}$
- (ii)  $\text{Rval}(f^*) - \text{Rval}(\bar{f}) \leq \frac{1-\rho}{\rho}(\text{val}(f^*) - \text{Rval}(f^*))$

where  $\rho = 1 - (1 - \frac{1}{k})^k$  and  $k = \max_{P \in \mathcal{P}} |E(P)|$ .

# Outline

- 1 Prelude
  - A motivating example
  - Interdiction curves
  - Dual of interdiction
  - Parametric curves
- 2 The Slope Problem
  - What is it?
  - Algorithms
  - Newton- $B$
- 3 Maximum Robust Flow
  - What is it?
- 4 Multiple Parameters
  - Multi-parametric scheduling problems



## 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  $s \rightarrow j \in A$  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  $s \rightarrow j \in A$  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

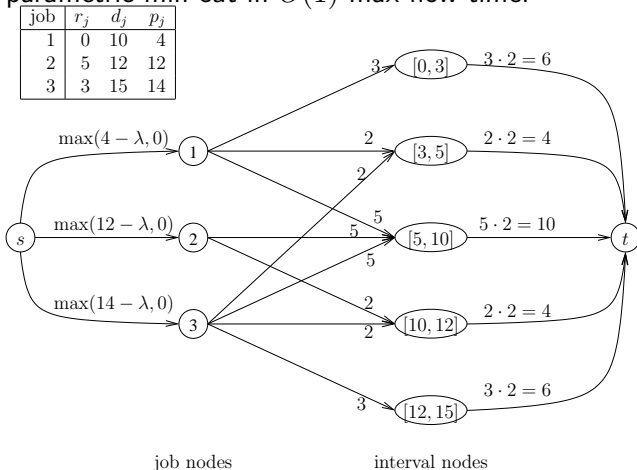
- We again start with a usual max flow network. We think of the nodes  $j$  such that  $s \rightarrow j \in A$  as **jobs**, and we denote  $c_{sj}$  by  $p_j$ , the **processing time** of job  $j$ .
- If a max flow in the network saturates all of these job arcs, then we are happy and the problem goes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

# Chen's scheduling problem: example

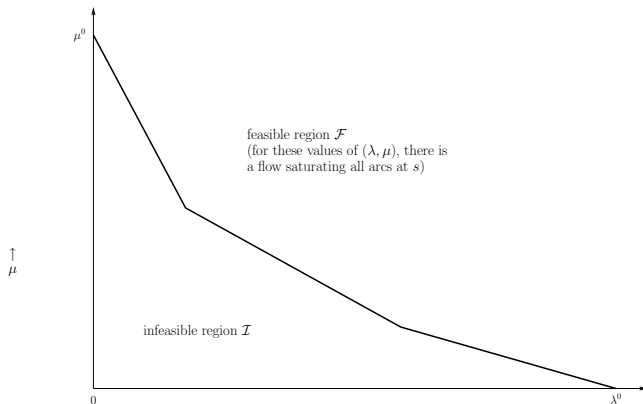
This single-parameter version can be solved using Gallo-Grigoriadis-Tarjan (GGT) '89 parametric min cut in  $O(1)$  max flow time.



# Two-parameter Chen

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

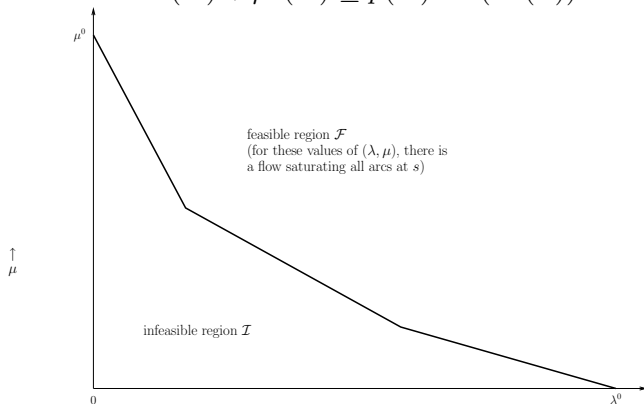
In the  $(\lambda, \mu)$  plane there is a piecewise linear convex curve separating feasible points from infeasible ones.



# Two-parameter Chen

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

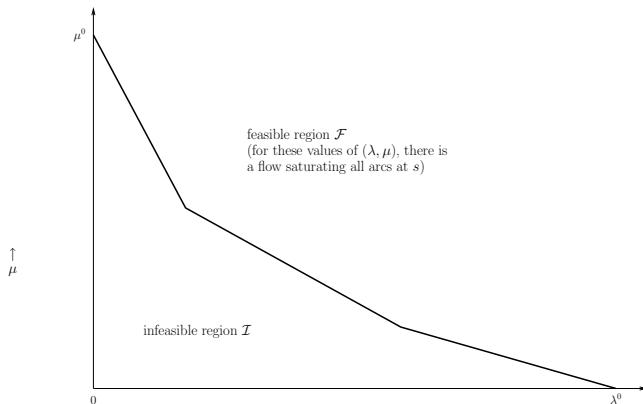
For node subset  $S$  with  $D \subseteq \delta^-(S) \cap \delta^+(\{s\})$ , the constraints defining this region have the form  $\lambda a(D) + \mu b(D) \geq p(D) - c(\delta^+(S))$ .



# Two-parameter Chen

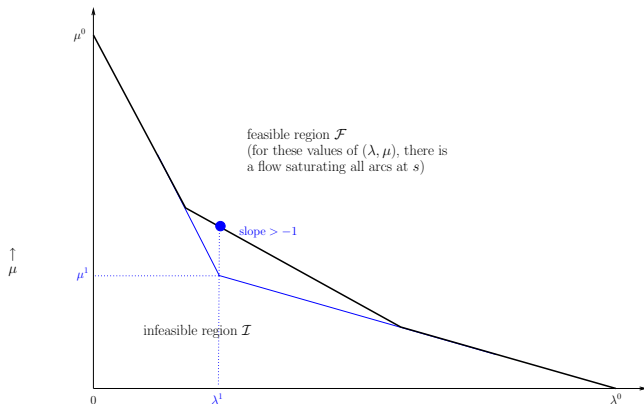
Suppose now that there are two ways to outsource,  $\lambda$  and  $\mu$  such that if we pay  $\$ \lambda + \$ \mu$ , we reduce  $p_j$  to  $\max(0, p_j - a_j \lambda - b_j \mu)$ .

We want to find a breakpoint of this curve whose local slopes bracket slope  $-1$ .



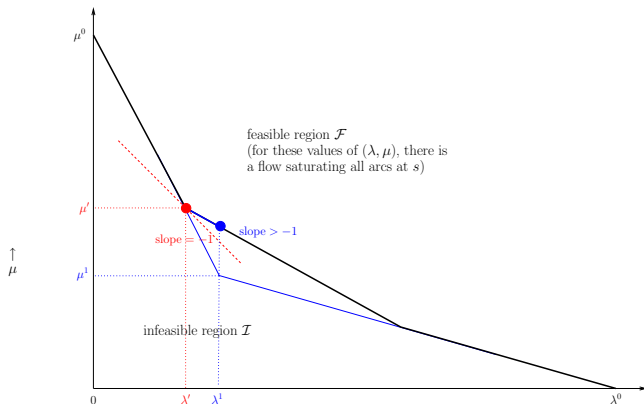
# Two-parameter Chen

Suppose now that there are two ways to outsource,  $\lambda$  and  $\mu$  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,  $\lambda$  and  $\mu$  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,  $\lambda$ ,  $\mu$ , and  $\nu$  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,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.

## Three-parameter Chen

- Suppose now that there are three ways to outsource,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.
- As we vary  $\nu$ , these 2-parameter solutions trace out a piecewise linear curve in the  $\nu$  direction.

## Three-parameter Chen

- Suppose now that there are three ways to outsource,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.
- As we vary  $\nu$ , these 2-parameter solutions trace out a piecewise linear curve in the  $\nu$  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,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.
- As we vary  $\nu$ , these 2-parameter solutions trace out a piecewise linear curve in the  $\nu$  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,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.
- As we vary  $\nu$ , these 2-parameter solutions trace out a piecewise linear curve in the  $\nu$  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,  $\lambda$ ,  $\mu$ , and  $\nu$  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  $\nu$  this is a 2-parameter problem we know how to solve.
- As we vary  $\nu$ , these 2-parameter solutions trace out a piecewise linear curve in the  $\nu$  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?

Any questions?

Questions?

Comments?



# Thank you

On behalf of all attendees, we thank the organizers:

# Thank you

On behalf of all attendees, we thank the organizers:

François Margot

# Thank you

On behalf of all attendees, we thank the organizers:

François Margot

and especially the the organizers who are retiring this year

Laurence Wolsey

# Thank you

On behalf of all attendees, we thank the organizers:

François Margot

and especially the the organizers who are retiring this year

Laurence Wolsey

and

Denis Naddef