

# Algorithms for the Unsplittable Flow Problem

Chandra Chekuri

Nitish Korula

Alina Ene

University of Illinois at Urbana-Champaign

Theory Seminar 2009

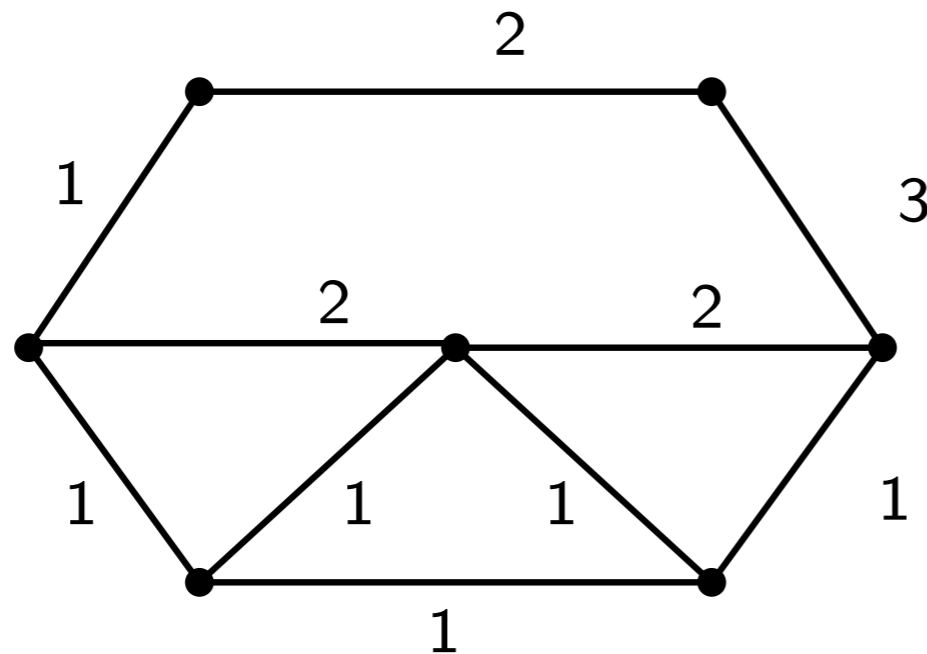
# Unsplittable Flow Problem (UFP)

# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge

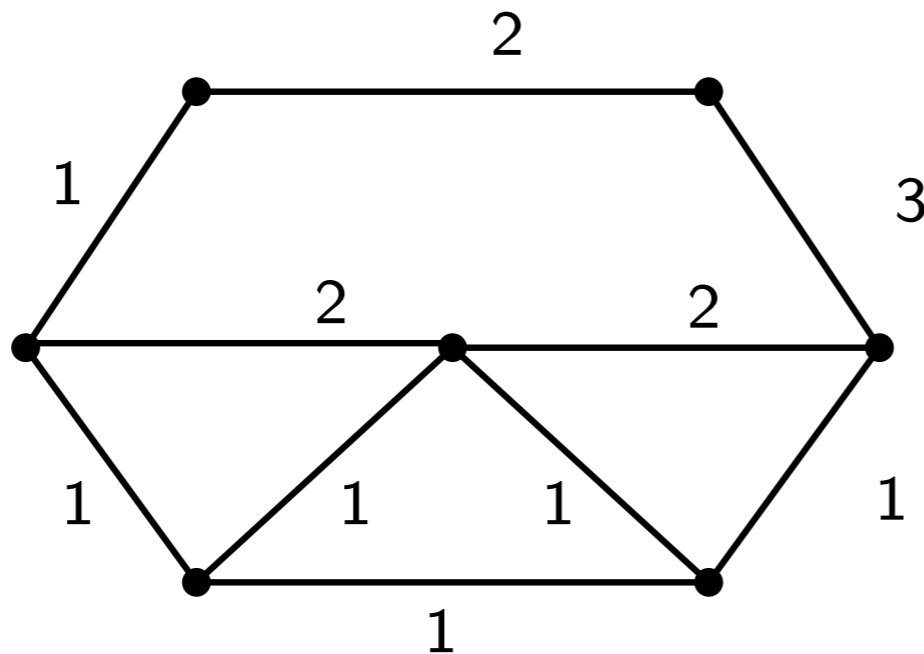
# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge



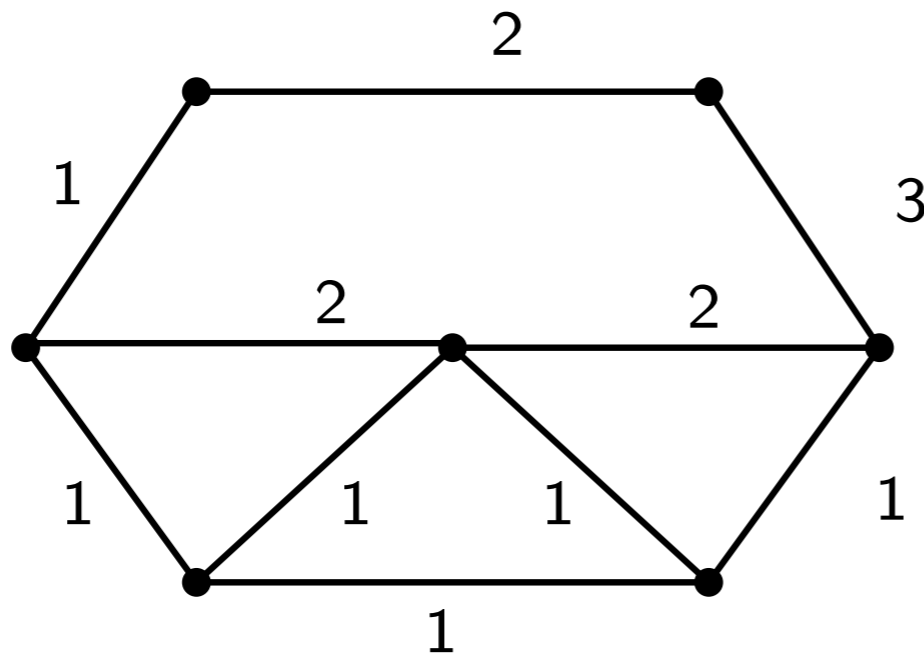
# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests



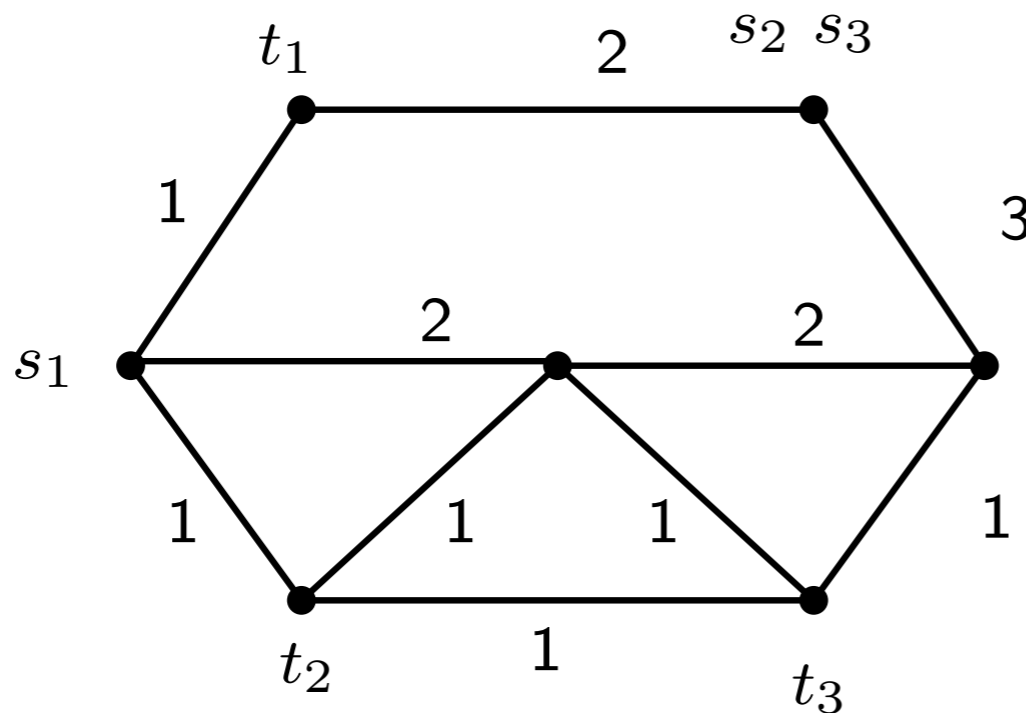
# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$



# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$



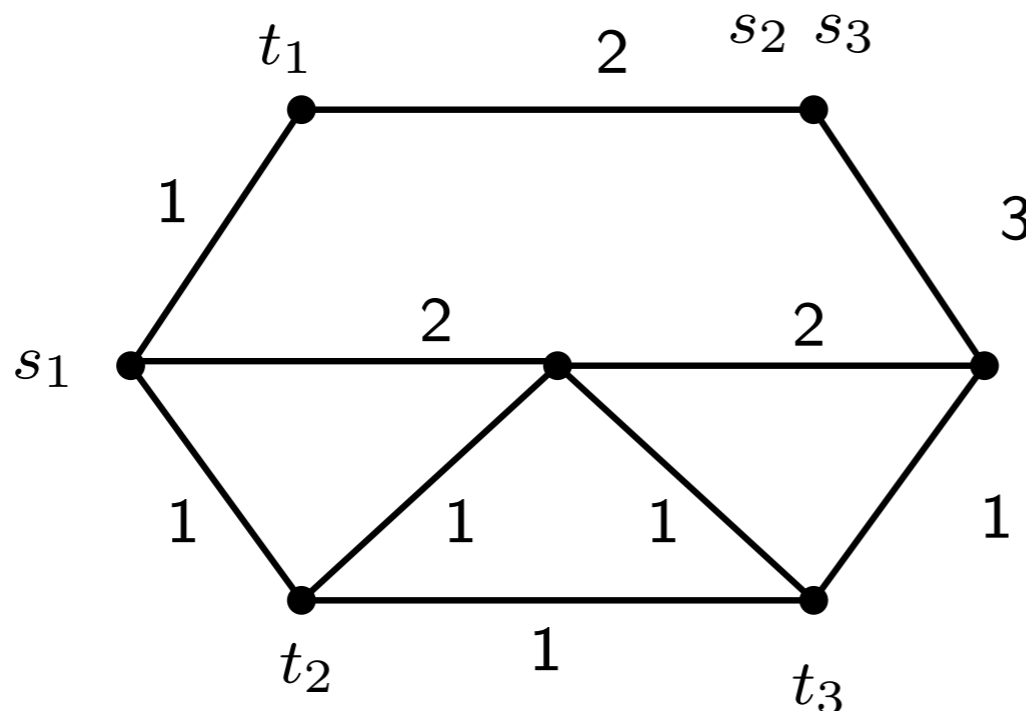
$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 1 \quad w_2 = 1$$

$$d_3 = 1 \quad w_3 = 1$$

# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$
- Route  $R_i$ : send  $d_i$  units of flow along a unique path from  $s_i$  to  $t_i$



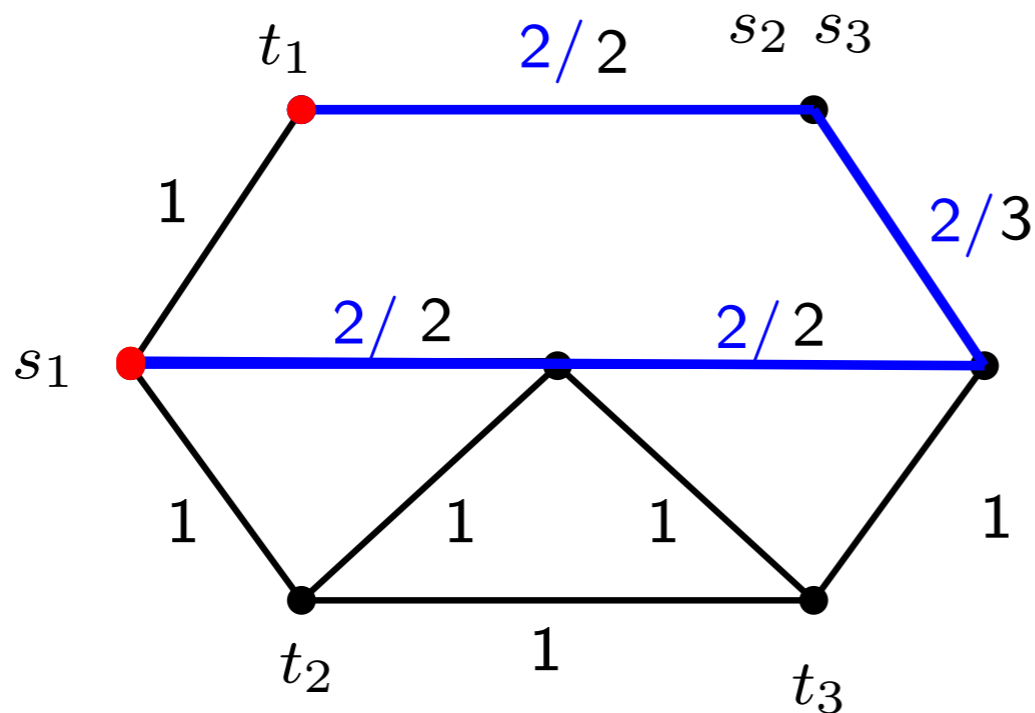
$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 1 \quad w_2 = 1$$

$$d_3 = 1 \quad w_3 = 1$$

# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$
- Route  $R_i$ : send  $d_i$  units of flow along a unique path from  $s_i$  to  $t_i$



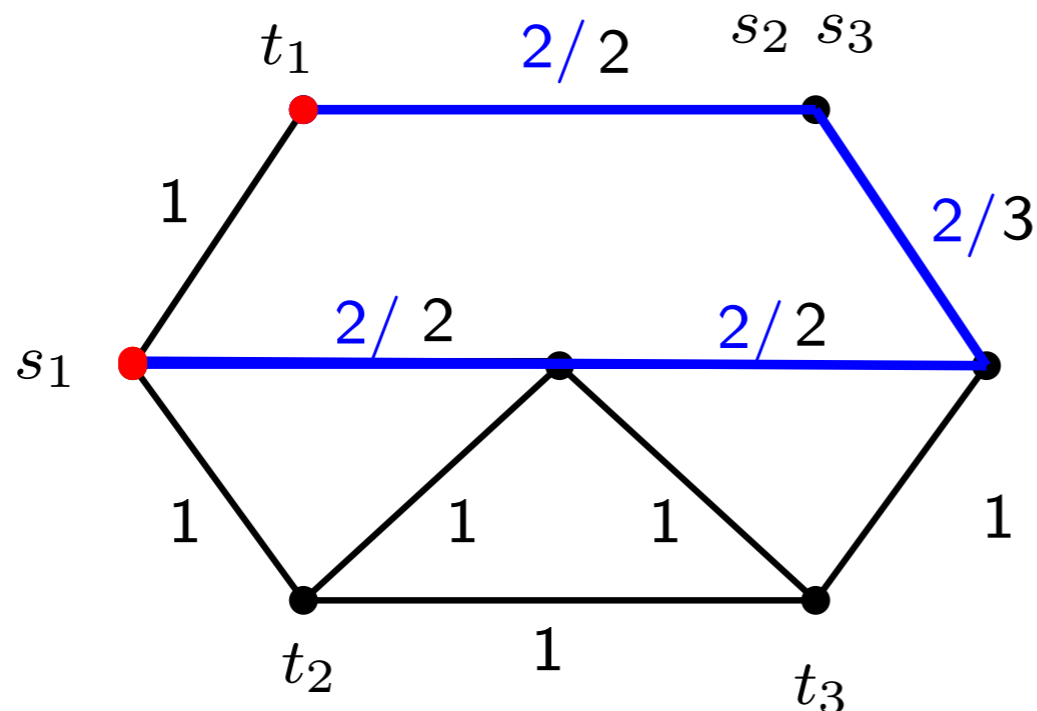
$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 1 \quad w_2 = 1$$

$$d_3 = 1 \quad w_3 = 1$$

# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$
- Route  $R_i$ : send  $d_i$  units of flow along a unique path from  $s_i$  to  $t_i$
- Find a feasible subset of requests with maximum profit



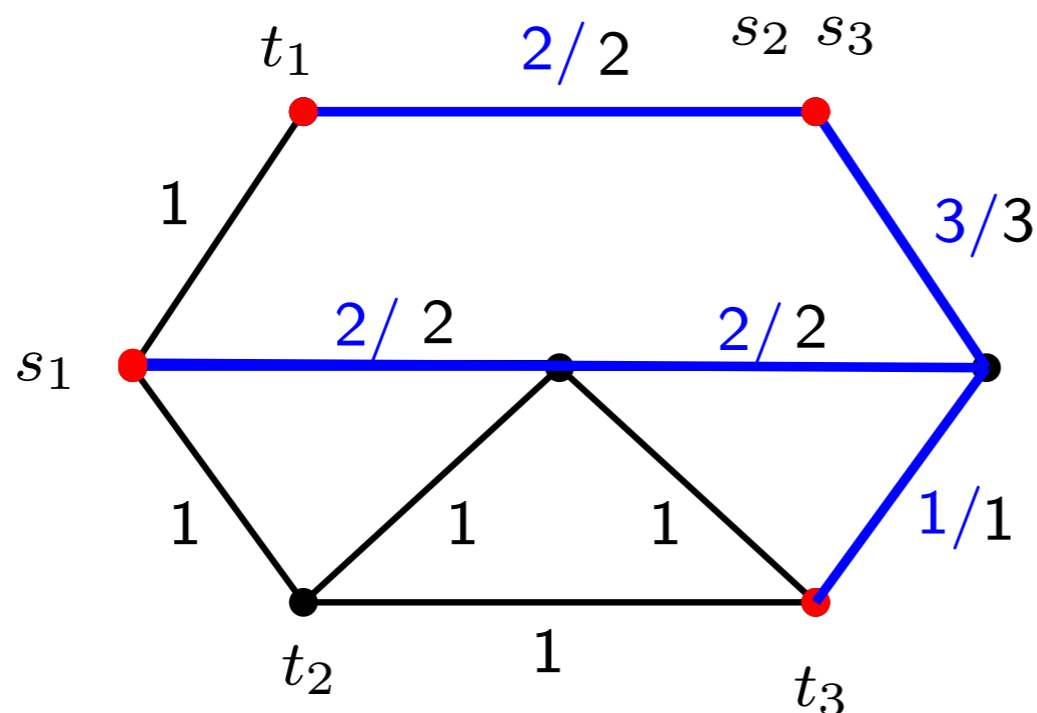
$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 1 \quad w_2 = 1$$

$$d_3 = 1 \quad w_3 = 1$$

# Unsplittable Flow Problem (UFP)

- Graph  $G$  (undirected/directed) with a capacity  $c_e$  on each edge
- Set  $\mathcal{R}$  of requests
- Request  $R_i$ : pair of vertices  $(s_i, t_i)$ , demand  $d_i$ , profit  $w_i$
- Route  $R_i$ : send  $d_i$  units of flow along a unique path from  $s_i$  to  $t_i$
- Find a feasible subset of requests with maximum profit



$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 1 \quad w_2 = 1$$

$$d_3 = 1 \quad w_3 = 1$$

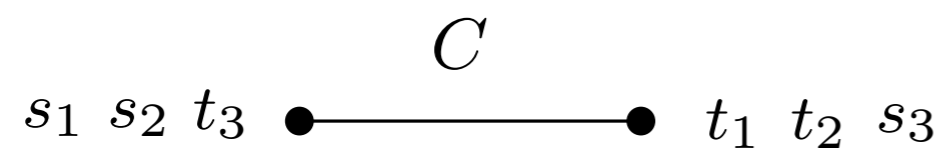
UFP is hard ...

# UFP is hard ...

- Knapsack

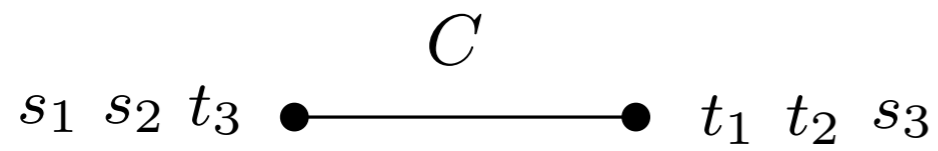
# UFP is hard ...

- Knapsack



# UFP is hard ...

- Knapsack

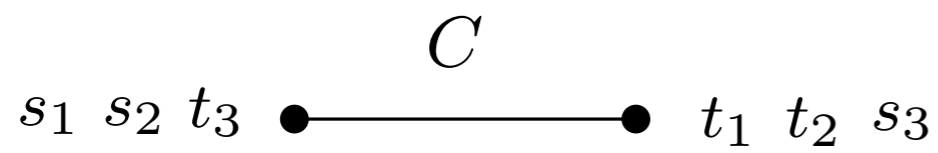


$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

# UFP is hard ...

- Knapsack



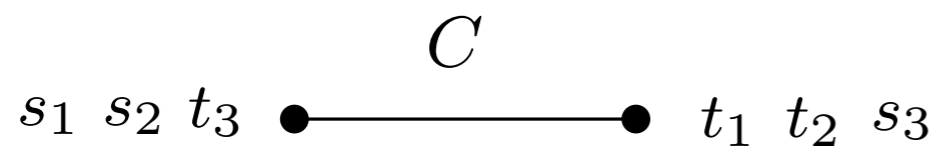
$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation

# UFP is hard ...

- Knapsack



$d_i$ : size of  $i$ -th item

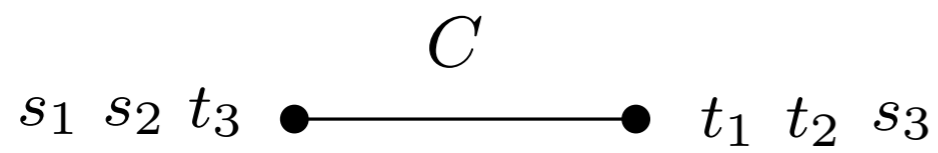
$w_i$ : profit of  $i$ -th item

- Resource Allocation



# UFP is hard ...

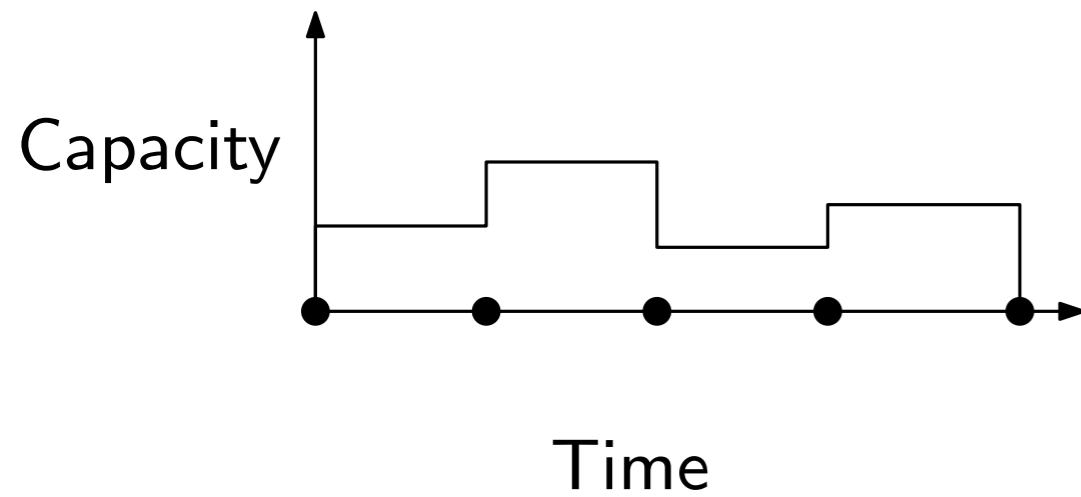
- Knapsack



$d_i$ : size of  $i$ -th item

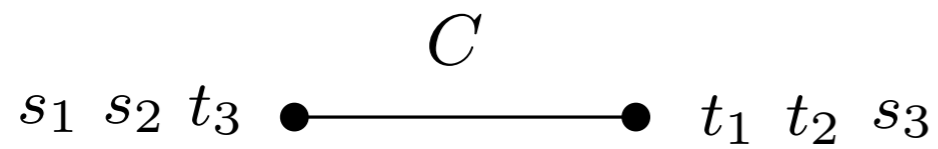
$w_i$ : profit of  $i$ -th item

- Resource Allocation



# UFP is hard ...

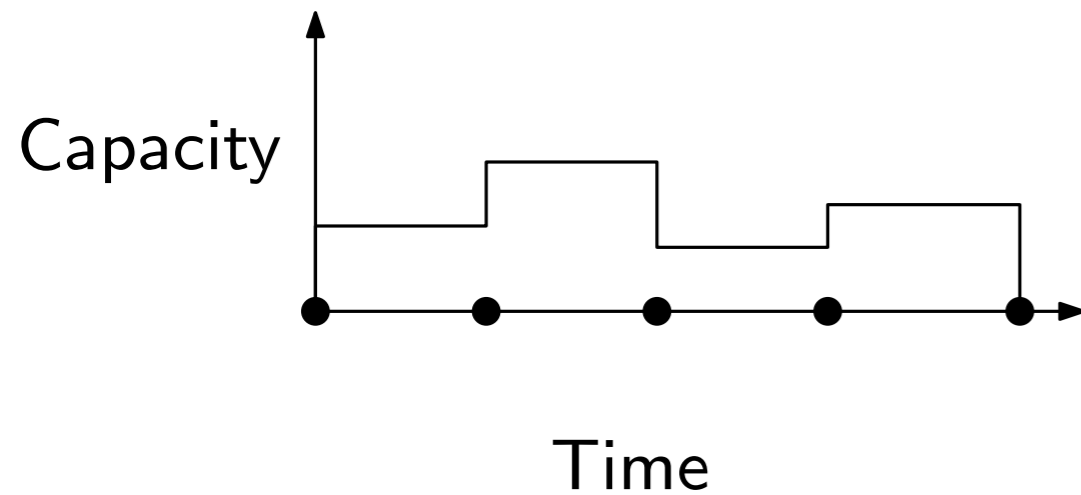
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

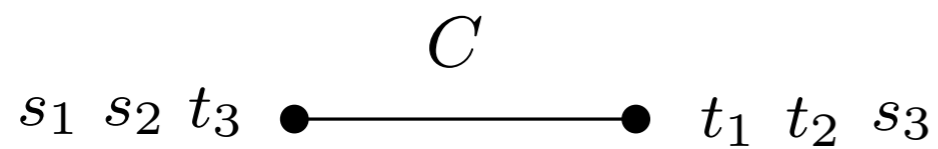
- Resource Allocation



$[s_i, t_i)$ : span of  $i$ -th task

# UFP is hard ...

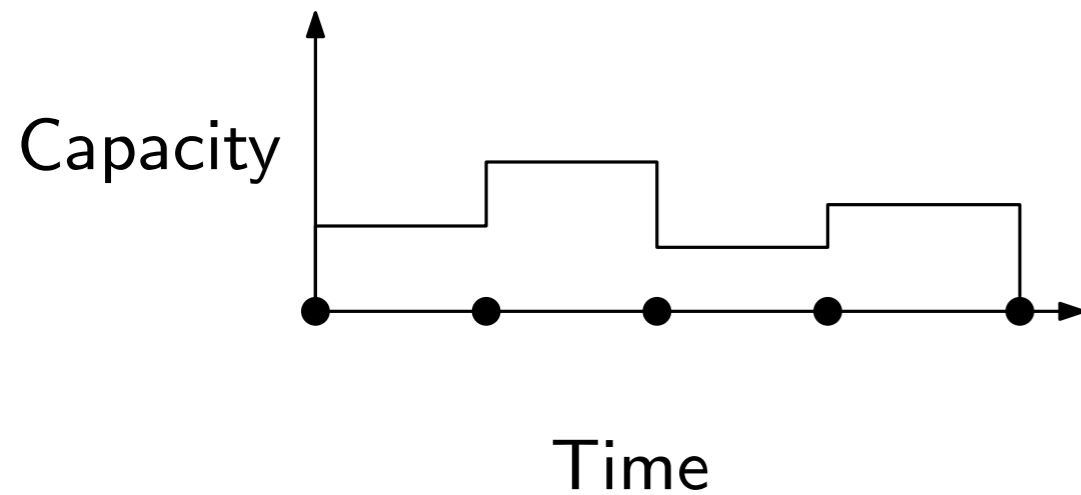
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation

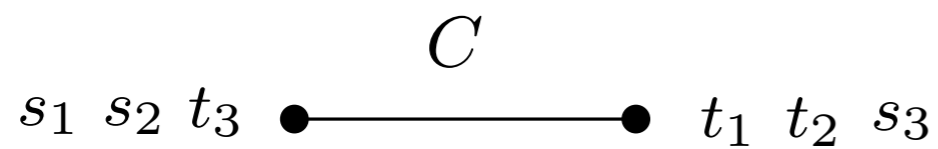


$[s_i, t_i)$ : span of  $i$ -th task

$d_i$ : demand of  $i$ -th task

# UFP is hard ...

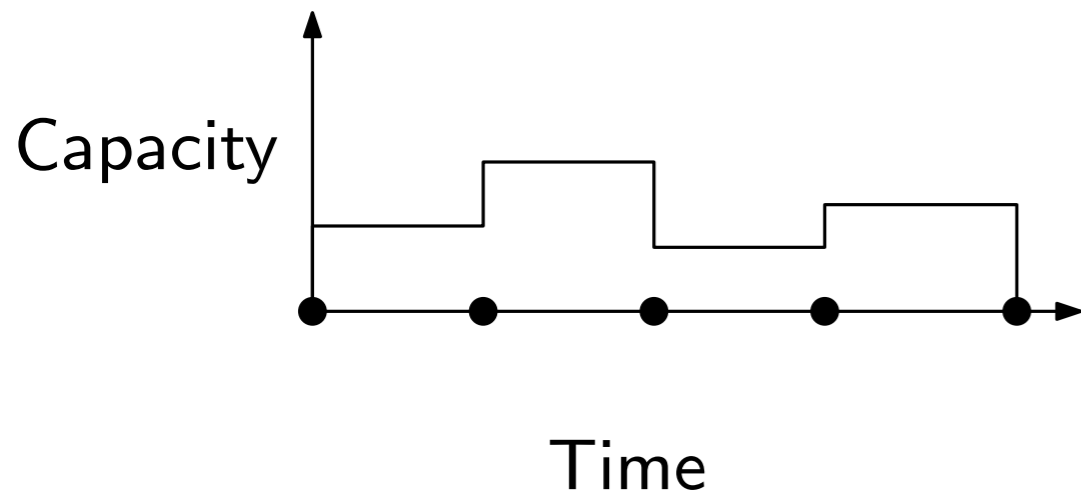
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation



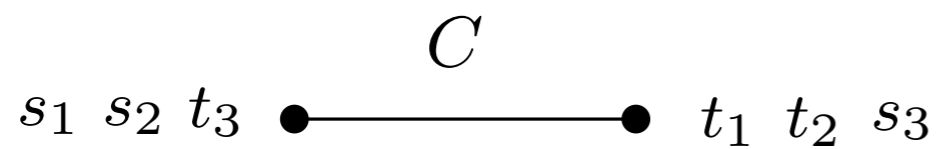
$[s_i, t_i)$ : span of  $i$ -th task

$d_i$ : demand of  $i$ -th task

$w_i$ : profit of  $i$ -th task

# UFP is hard ...

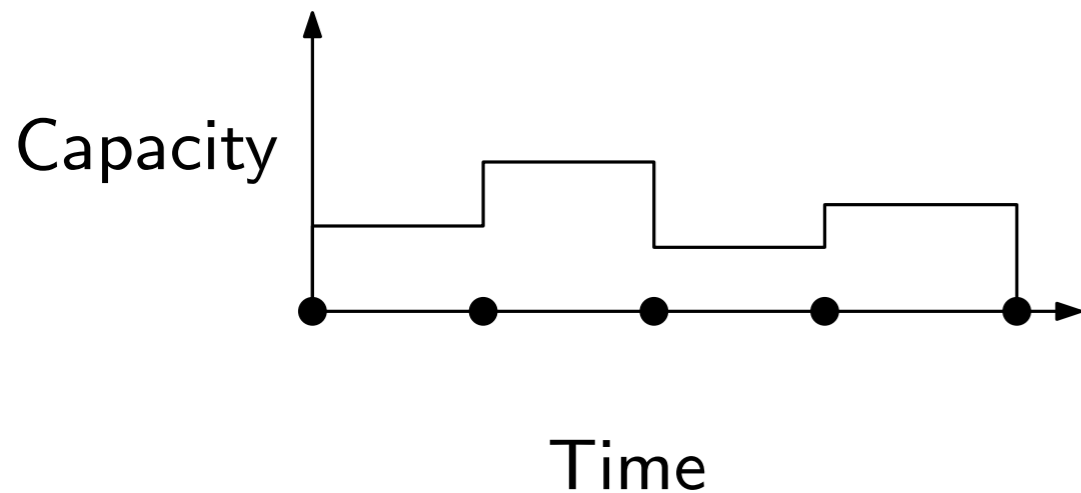
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation



$[s_i, t_i)$ : span of  $i$ -th task

$d_i$ : demand of  $i$ -th task

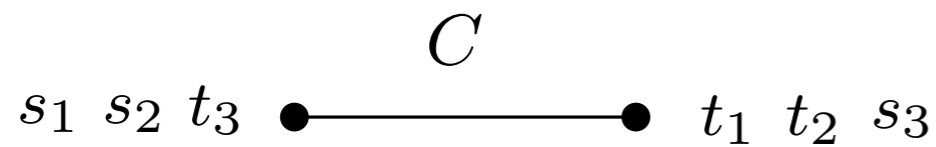
$w_i$ : profit of  $i$ -th task

- Edge Disjoint Paths

$$d_i = w_i = 1, c_e = 1$$

# UFP is hard ...

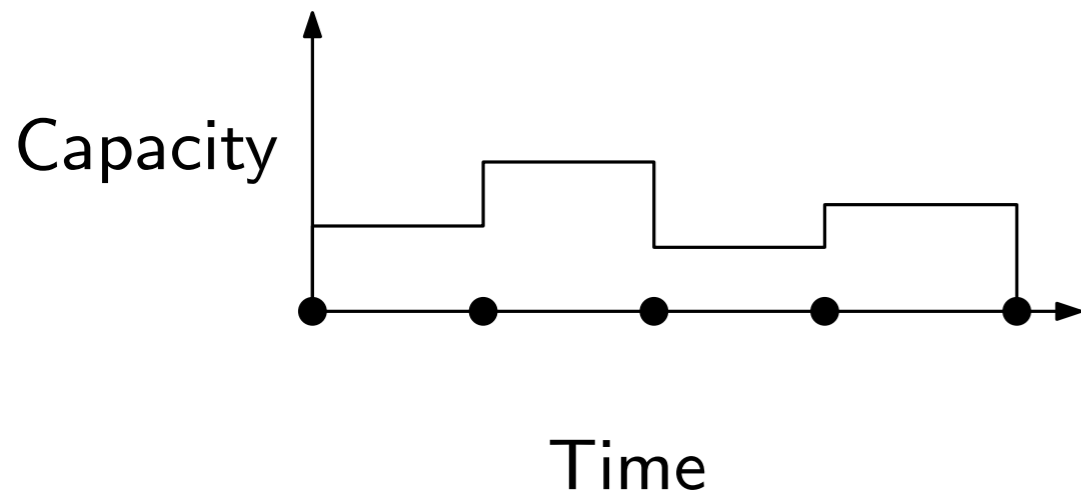
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation



$[s_i, t_i)$ : span of  $i$ -th task

$d_i$ : demand of  $i$ -th task

$w_i$ : profit of  $i$ -th task

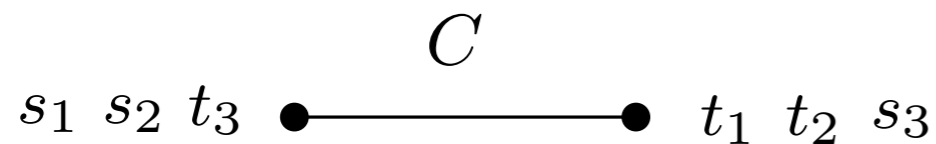
- Edge Disjoint Paths

$$d_i = w_i = 1, c_e = 1$$

packing is  
hard

# UFP is hard ...

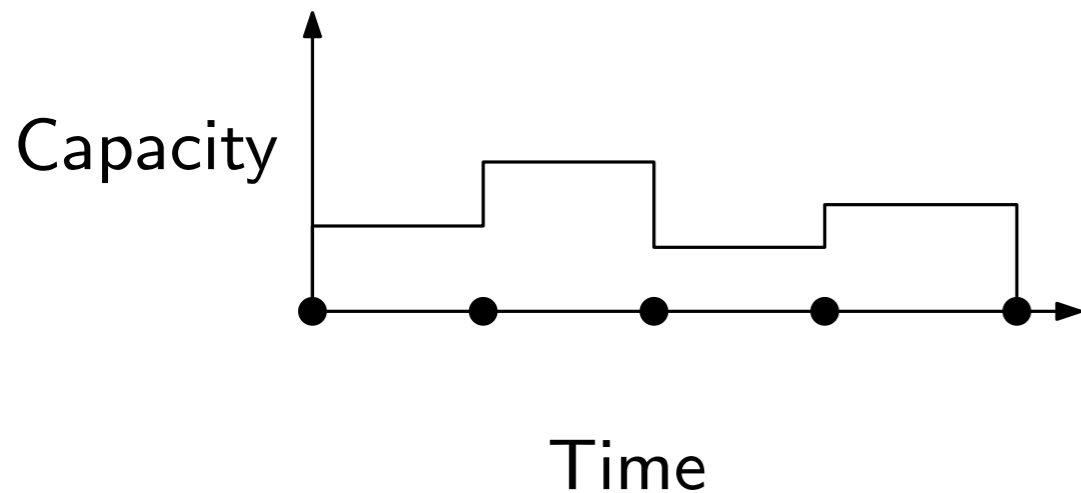
- Knapsack



$d_i$ : size of  $i$ -th item

$w_i$ : profit of  $i$ -th item

- Resource Allocation



$[s_i, t_i)$ : span of  $i$ -th task

$d_i$ : demand of  $i$ -th task

$w_i$ : profit of  $i$ -th task

- Edge Disjoint Paths

$$d_i = w_i = 1, c_e = 1$$

packing is  
hard

routing is  
hard

LP-based approaches?

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

- variable  $x_i$ : indicates whether request  $R_i$  is selected

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

- variable  $x_i$ : indicates whether request  $R_i$  is selected
- capacity constraints: the total demand of selected requests on each edge is at most the capacity

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

- variable  $x_i$ : indicates whether request  $R_i$  is selected
- capacity constraints: the total demand of selected requests on each edge is at most the capacity
- $P_i$ : unique path between  $s_i$  and  $t_i$

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

- variable  $x_i$ : indicates whether request  $R_i$  is selected
- capacity constraints: the total demand of selected requests on each edge is at most the capacity
- $P_i$ : unique path between  $s_i$  and  $t_i$

**Standard LP**     $\max \sum_i w_i x_i$

$$\begin{array}{llll} \sum_{i: e \in P_i} d_i x_i & \leq & c_e & (\forall e \in E(G)) \\ x_i & \in & [0, 1] & (\forall i \in \{1, \dots, k\}) \end{array}$$

# LP-based approaches?

Standard LP relaxation for UFP on paths and trees

- variable  $x_i$ : indicates whether request  $R_i$  is selected
- capacity constraints: the total demand of selected requests on each edge is at most the capacity
- $P_i$ : unique path between  $s_i$  and  $t_i$

**Standard LP**

$$\max \sum_i w_i x_i$$

$$\sum_{i: e \in P_i} d_i x_i \leq c_e \quad (\forall e \in E(G))$$

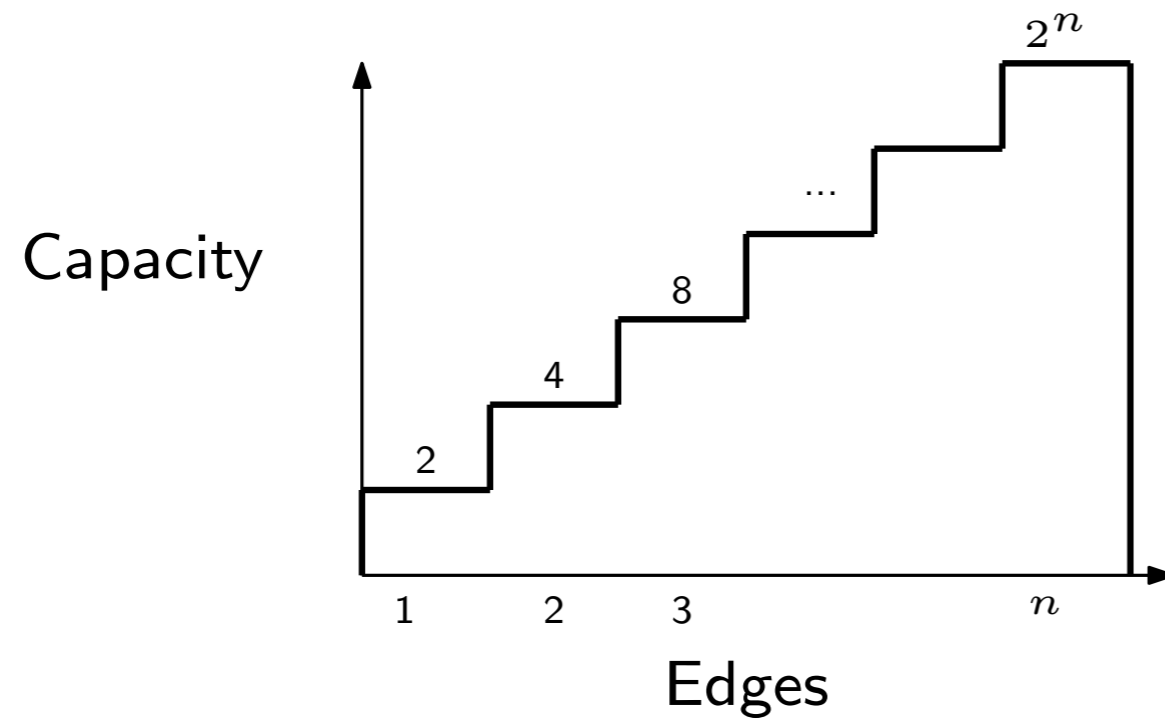
$$x_i \in [0, 1] \quad (\forall i \in \{1, \dots, k\})$$

**Lemma:** The standard LP has  $\Omega(n)$  integrality gap. [Chakrabarti et al., 2002]

LP-based approaches?

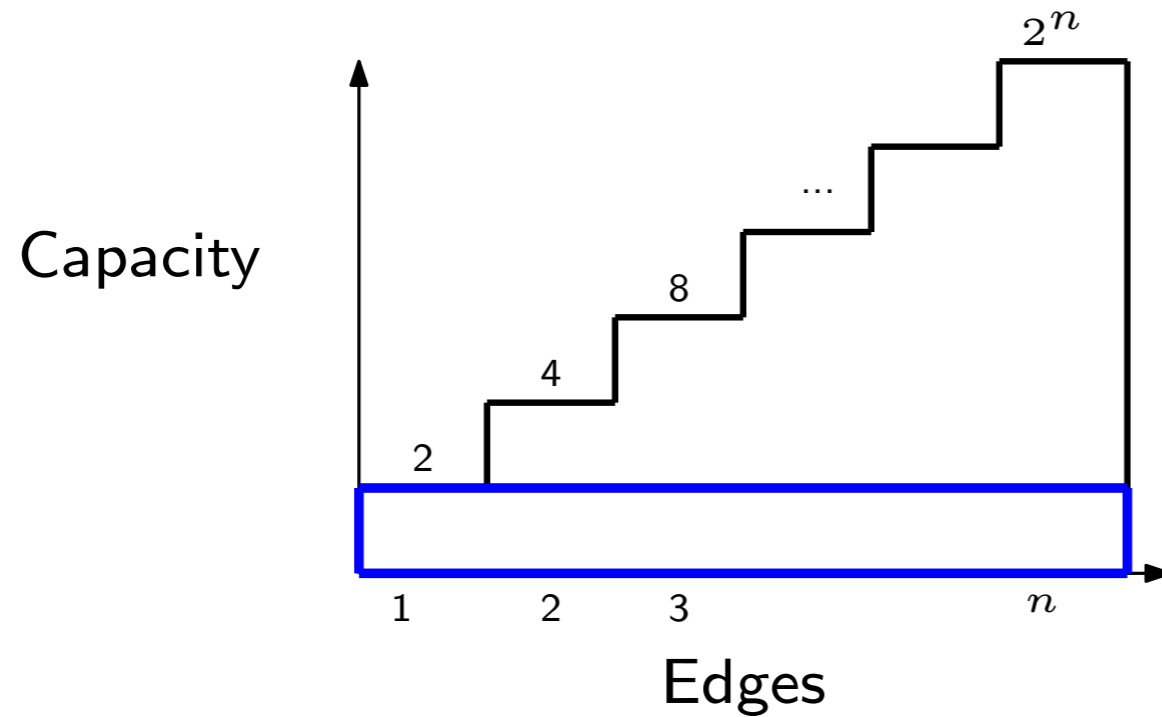
# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.



# LP-based approaches?

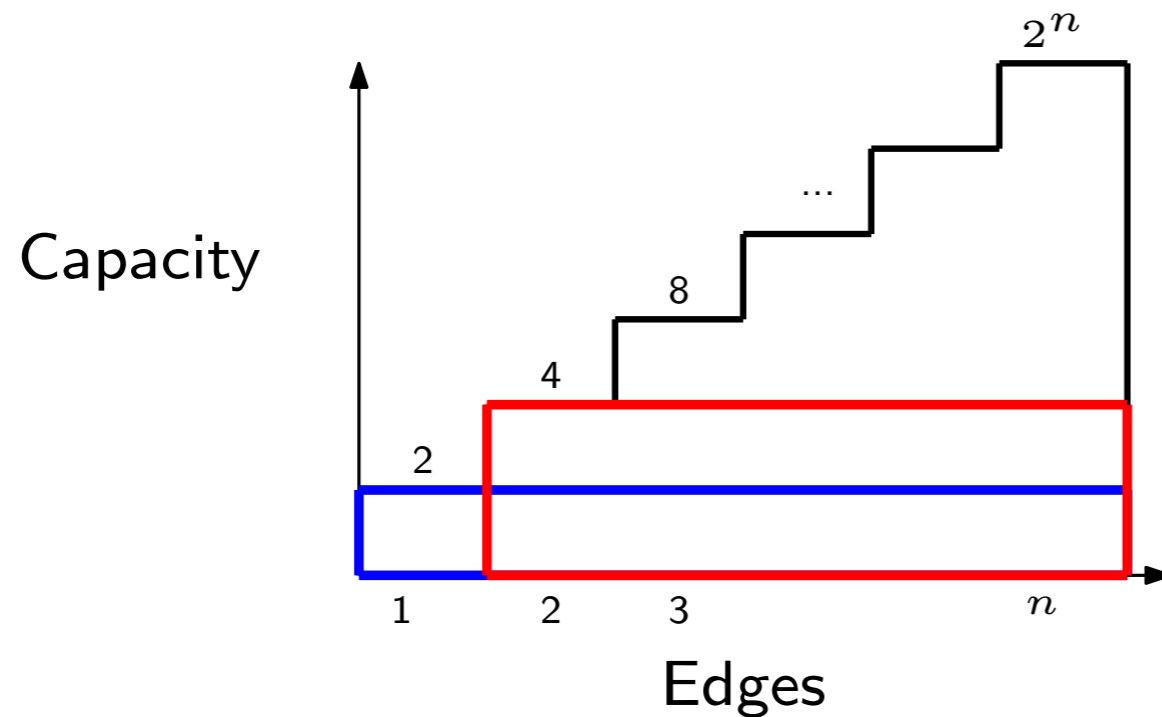
The standard LP has  $\Omega(n)$  integrality gap.



$$d_1 = 2 \quad w_1 = 1$$

# LP-based approaches?

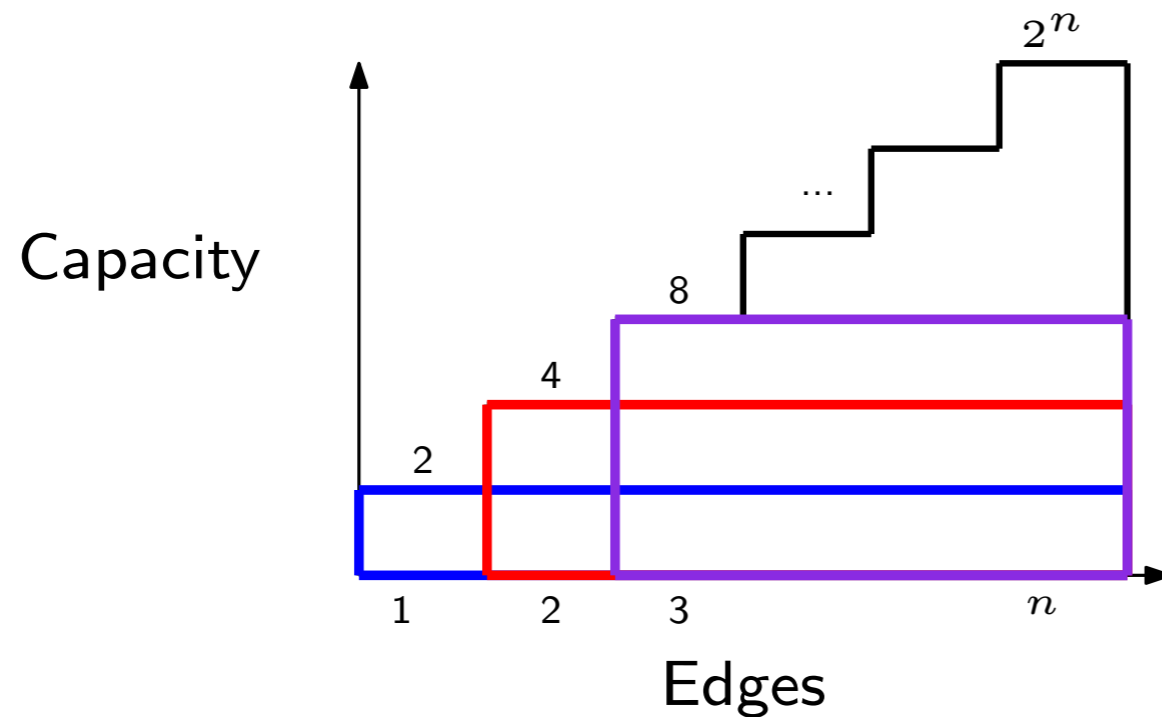
The standard LP has  $\Omega(n)$  integrality gap.



$$\begin{array}{ll} d_1 = 2 & w_1 = 1 \\ d_2 = 4 & w_2 = 1 \end{array}$$

# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.



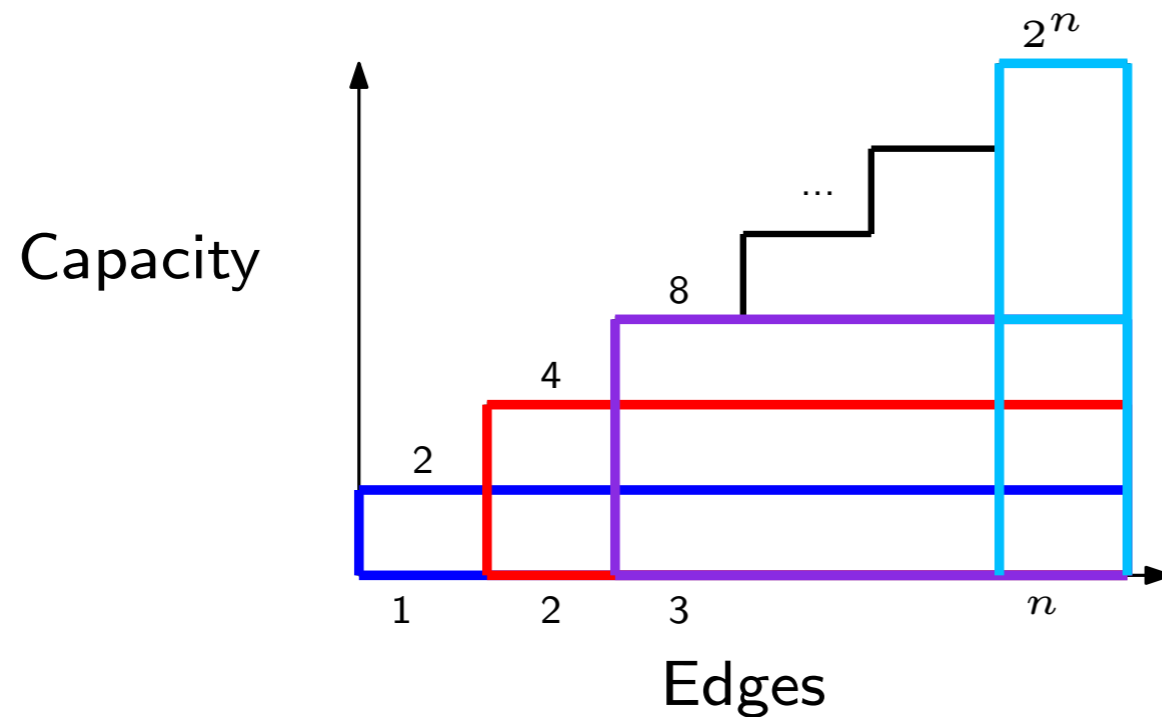
$$d_1 = 2 \quad w_1 = 1$$

$$d_2 = 4 \quad w_2 = 1$$

$$d_3 = 8 \quad w_3 = 1$$

# LP-based approaches?

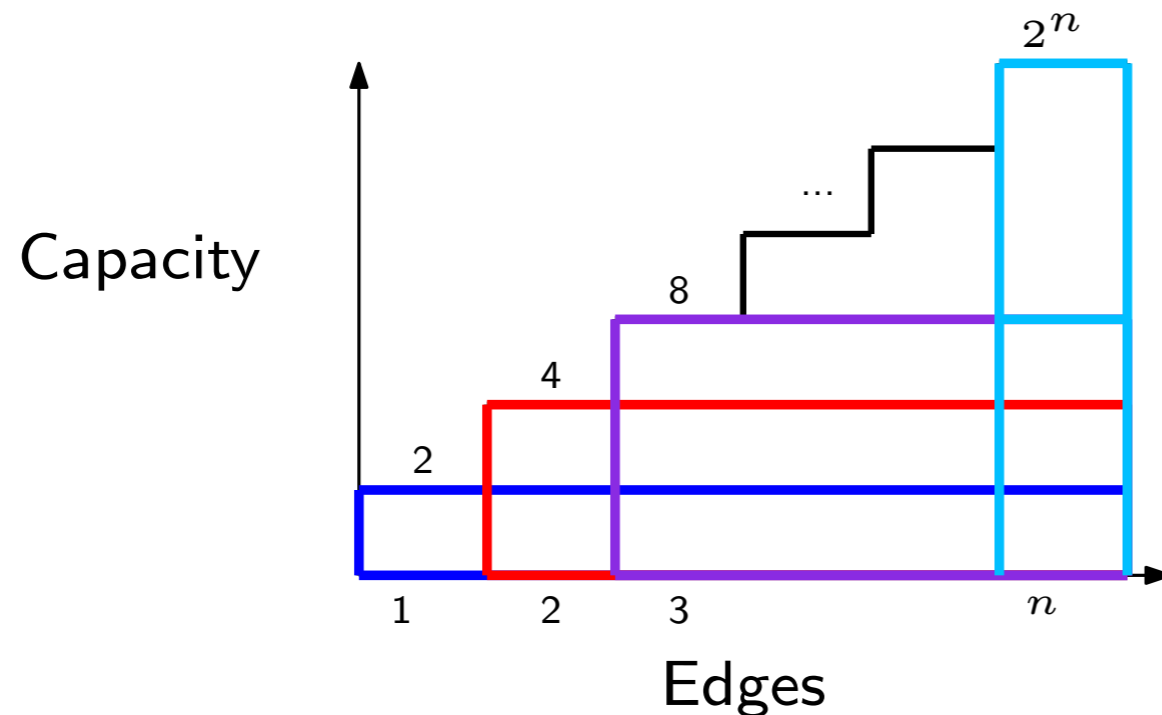
The standard LP has  $\Omega(n)$  integrality gap.



$$\begin{aligned} d_1 &= 2 & w_1 &= 1 \\ d_2 &= 4 & w_2 &= 1 \\ d_3 &= 8 & w_3 &= 1 \\ & \dots & & \\ d_n &= 2^n & w_n &= 1 \end{aligned}$$

# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.

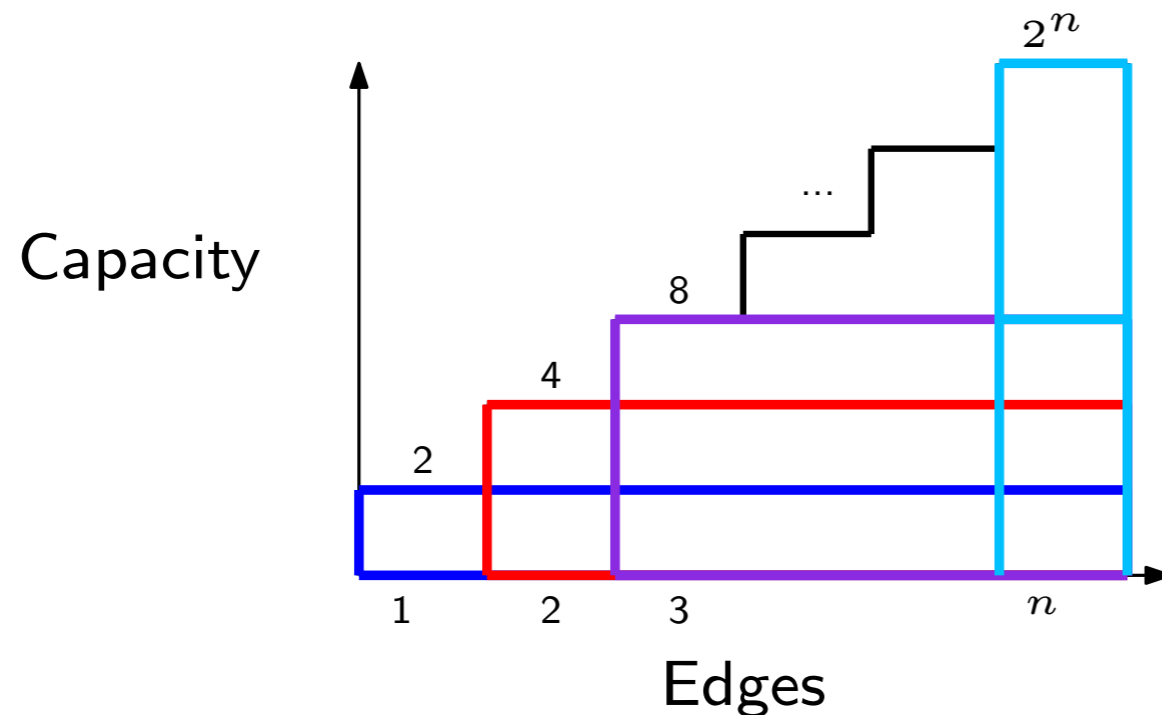


$$\begin{aligned} d_1 &= 2 & w_1 &= 1 \\ d_2 &= 4 & w_2 &= 1 \\ d_3 &= 8 & w_3 &= 1 \\ & \dots & & \\ d_n &= 2^n & w_n &= 1 \end{aligned}$$

LP solution has profit at least  $n/2$ :  $x_i = 1/2$  is feasible

# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.



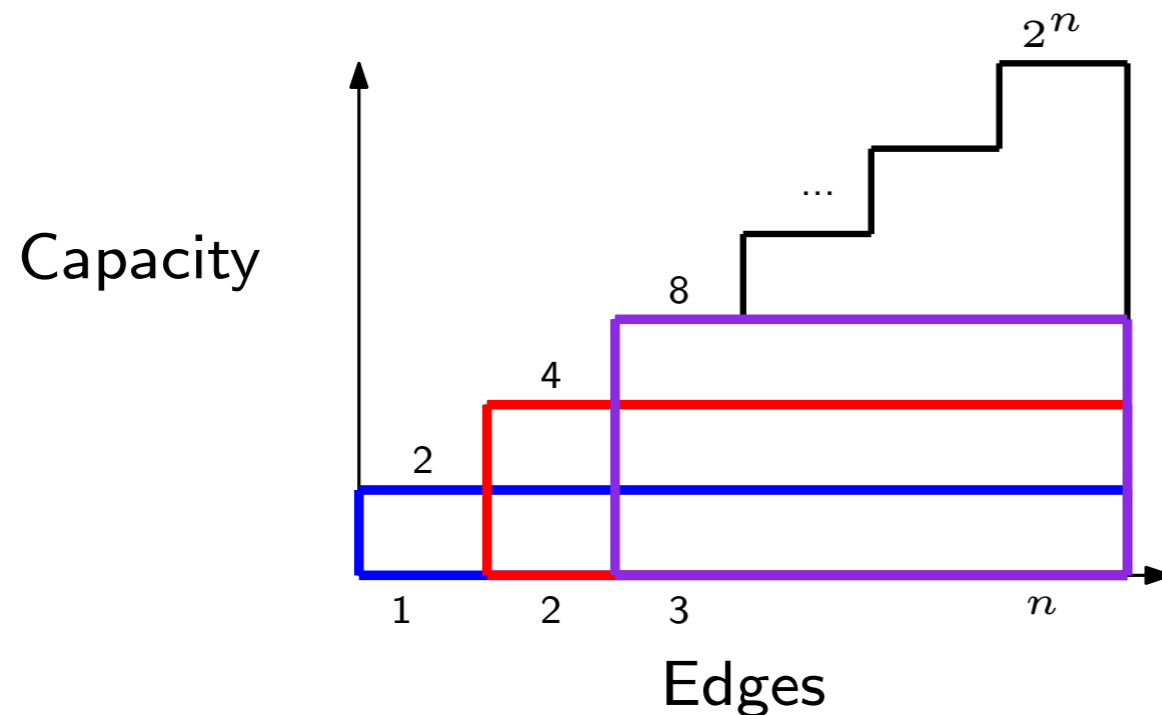
$$\begin{array}{ll} d_1 = 2 & w_1 = 1 \\ d_2 = 4 & w_2 = 1 \\ d_3 = 8 & w_3 = 1 \\ & \dots \\ d_n = 2^n & w_n = 1 \end{array}$$

LP solution has profit at least  $n/2$ :  $x_i = 1/2$  is feasible

Integral solution has profit 1: any integral solution routes at most one request

# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.



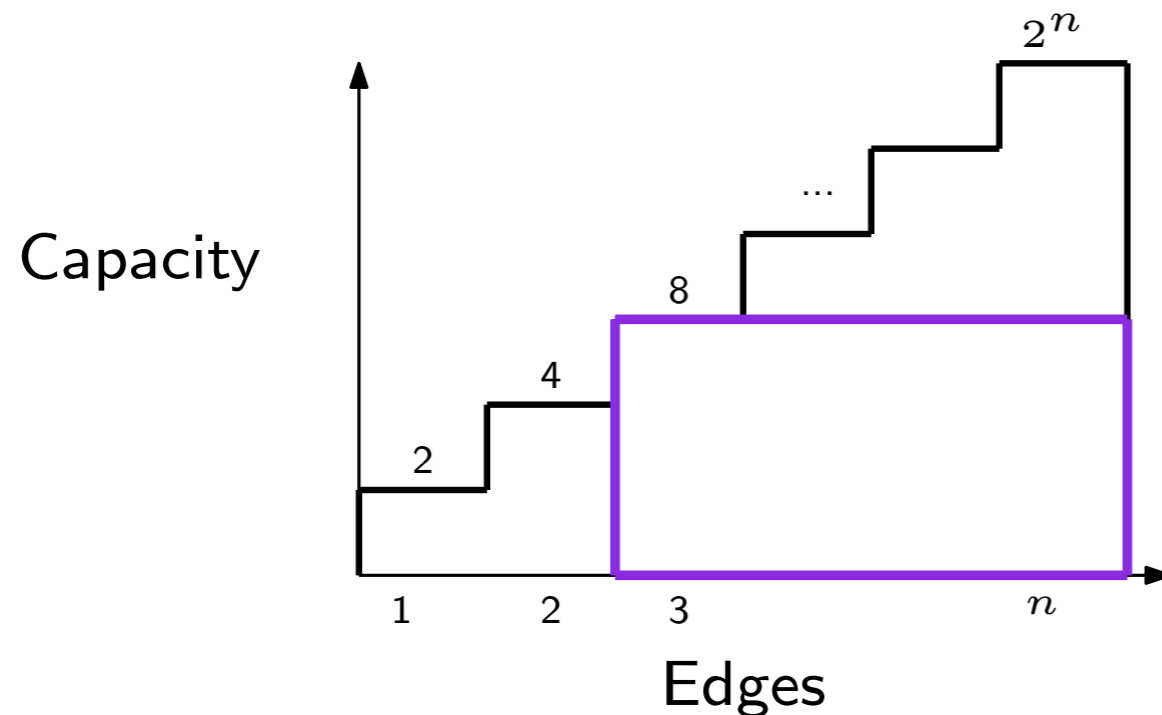
$d_1 = 2$	$w_1 = 1$
$d_2 = 4$	$w_2 = 1$
$d_3 = 8$	$w_3 = 1$

LP solution has profit at least  $n/2$ :  $x_i = 1/2$  is feasible

Integral solution has profit 1: any integral solution routes at most one request

# LP-based approaches?

The standard LP has  $\Omega(n)$  integrality gap.



$$d_3 = 8 \quad w_3 = 1$$

LP solution has profit at least  $n/2$ :  $x_i = 1/2$  is feasible

Integral solution has profit 1: any integral solution routes at most one request

LP-based approaches?

# LP-based approaches?

No-bottleneck assumption (NBA)

# LP-based approaches?

No-bottleneck assumption (NBA)

- $\max_i d_i \leq \min_e c_e$

# LP-based approaches?

No-bottleneck assumption (NBA)

- $\max_i d_i \leq \min_e c_e$

Standard LP has  $O(1)$  gap for UFP-NBA on paths [Chakrabarti et al, 2002]

# LP-based approaches?

No-bottleneck assumption (NBA)

- $\max_i d_i \leq \min_e c_e$

Standard LP has  $O(1)$  gap for UFP-NBA on paths [Chakrabarti et al, 2002]

Standard LP has  $O(1)$  gap for UFP-NBA on trees [Chekuri et al., 2003]

# LP-based approaches?

No-bottleneck assumption (NBA)

- $\max_i d_i \leq \min_e c_e$

Standard LP has  $O(1)$  gap for UFP-NBA on paths [Chakrabarti et al, 2002]

Standard LP has  $O(1)$  gap for UFP-NBA on trees [Chekuri et al., 2003]

UFP-NBA still hard

- Edge Disjoint Paths: best approx is  $\sqrt{n}$

# LP-based approaches?

No-bottleneck assumption (NBA)

- $\max_i d_i \leq \min_e c_e$

Standard LP has  $O(1)$  gap for UFP-NBA on paths [Chakrabarti et al, 2002]

Standard LP has  $O(1)$  gap for UFP-NBA on trees [Chekuri et al., 2003]

UFP-NBA still hard

- Edge Disjoint Paths: best approx is  $\sqrt{n}$

This talk: UFP *without* NBA

UFP without NBA

# UFP without NBA

Previous work (UFP on paths)

# UFP without NBA

Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]

# UFP without NBA

Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]
- Recently,  $O(\log n)$  [Bansal et al, 2009]

# UFP without NBA

Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]
- Recently,  $O(\log n)$  [Bansal et al, 2009]

Our results

# UFP without NBA

Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]
- Recently,  $O(\log n)$  [Bansal et al, 2009]

Our results

- A simple combinatorial algorithm that achieves an  $O(\log^2 n)$ -approx for UFP on trees

# UFP without NBA

Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]
- Recently,  $O(\log n)$  [Bansal et al, 2009]

Our results

- A simple combinatorial algorithm that achieves an  $O(\log^2 n)$ -approx for UFP on trees
- An LP relaxation with an  $O(\log^2 n)$  integrality gap for UFP on paths

# UFP without NBA

## Previous work (UFP on paths)

- $(1 + \epsilon)$  quasi-polynomial time approx: capacities and demands at most  $2^{\text{polylog}(n)}$  [Bansal et al, 2006]
- Recently,  $O(\log n)$  [Bansal et al, 2009]

## Our results

- A simple combinatorial algorithm that achieves an  $O(\log^2 n)$ -approx for UFP on trees
- An LP relaxation with an  $O(\log^2 n)$  integrality gap for UFP on paths
- A few other results (not covered in this talk)

UFP on trees

# UFP on trees

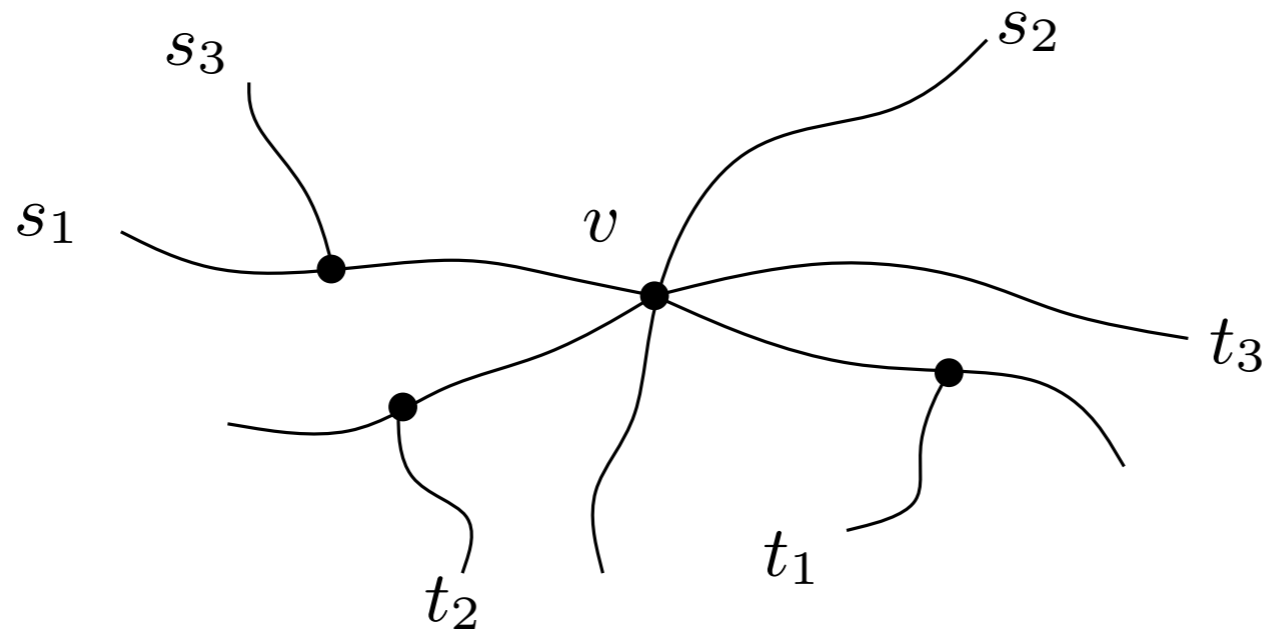
- Unit profits

# UFP on trees

- Unit profits
- Demand paths pass through a common vertex  $v$

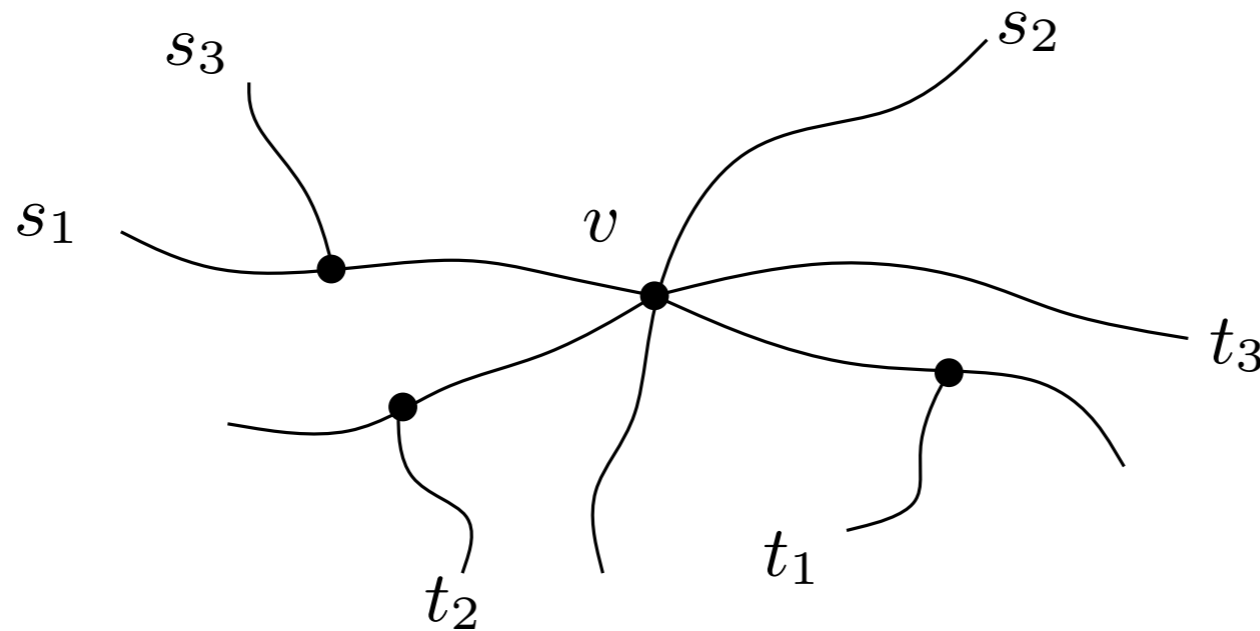
# UFP on trees

- Unit profits
- Demand paths pass through a common vertex  $v$



# UFP on trees

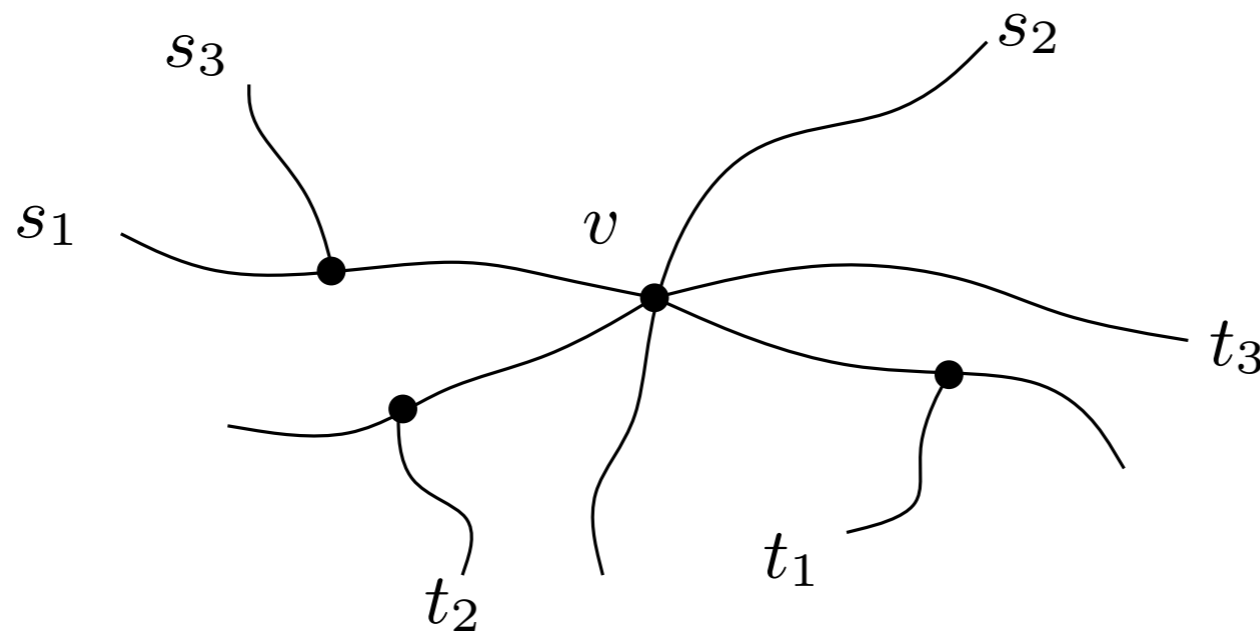
- Unit profits
- Demand paths pass through a common vertex  $v$



- Sort demands:  $d_1 \leq d_2 \leq \dots \leq d_k$

# UFP on trees

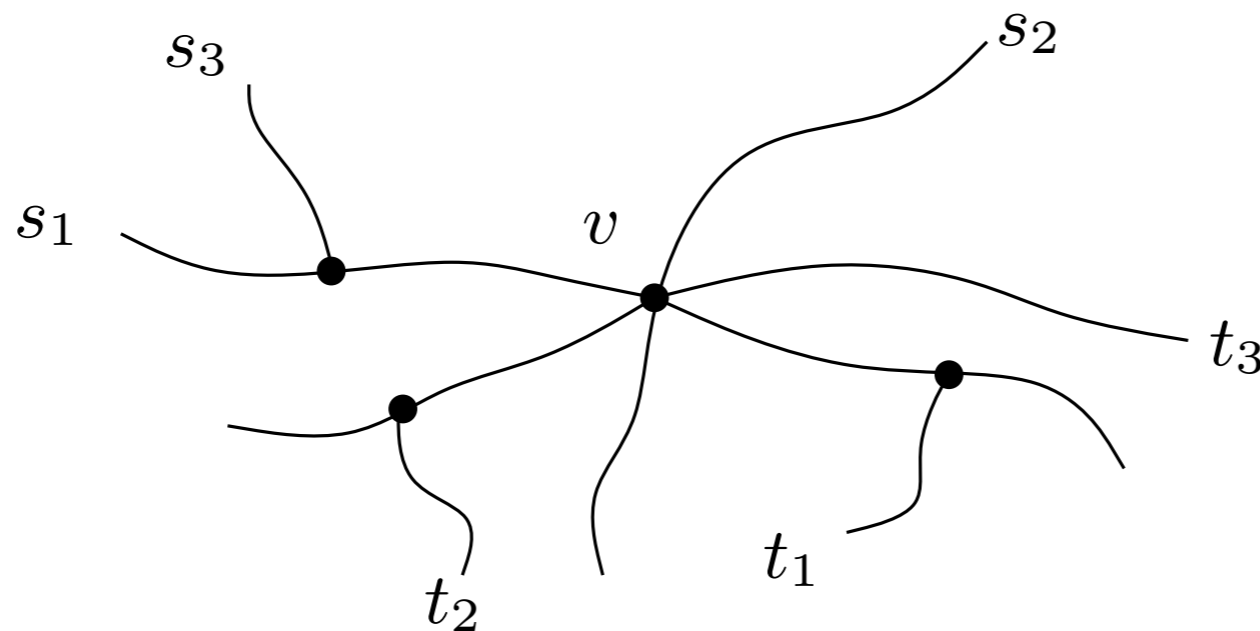
- Unit profits
- Demand paths pass through a common vertex  $v$



- Sort demands:  $d_1 \leq d_2 \leq \dots \leq d_k$
- Greedily select requests while maintaining feasibility

# UFP on trees

- Unit profits
- Demand paths pass through a common vertex  $v$



2-approximation

- Sort demands:  $d_1 \leq d_2 \leq \dots \leq d_k$
- Greedily select requests while maintaining feasibility

UFP on trees

# UFP on trees

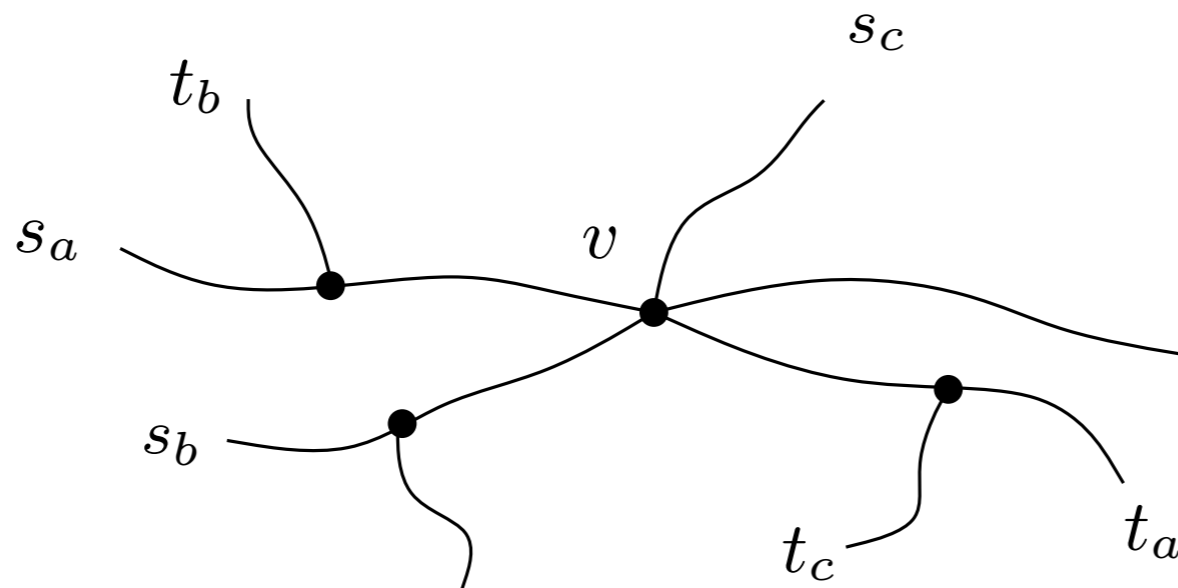
**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

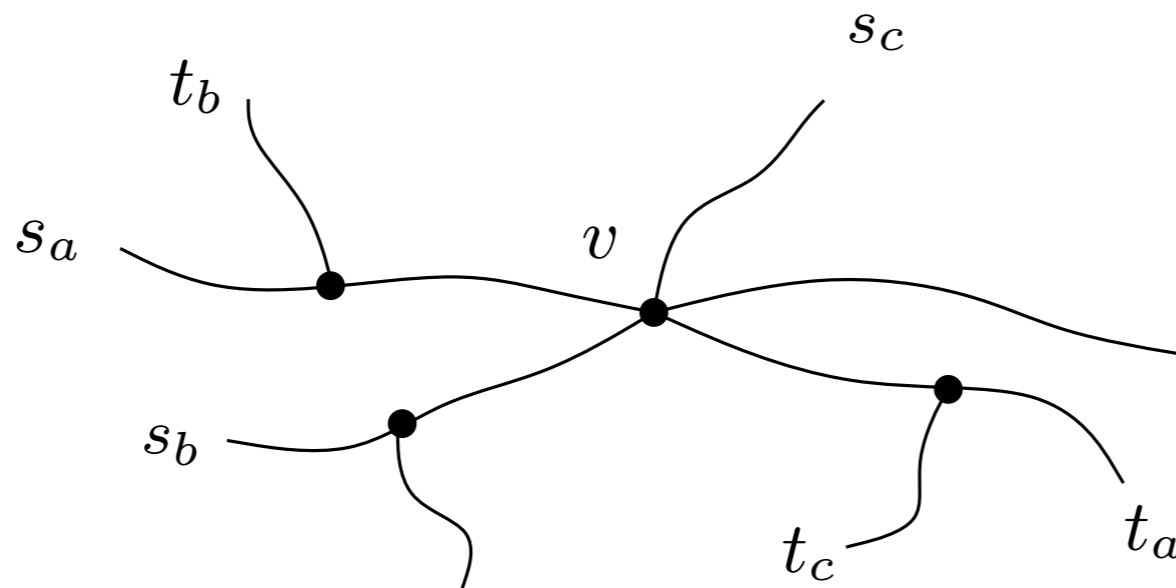
Swap two requests in optimal solution with a request selected by Greedy



# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy

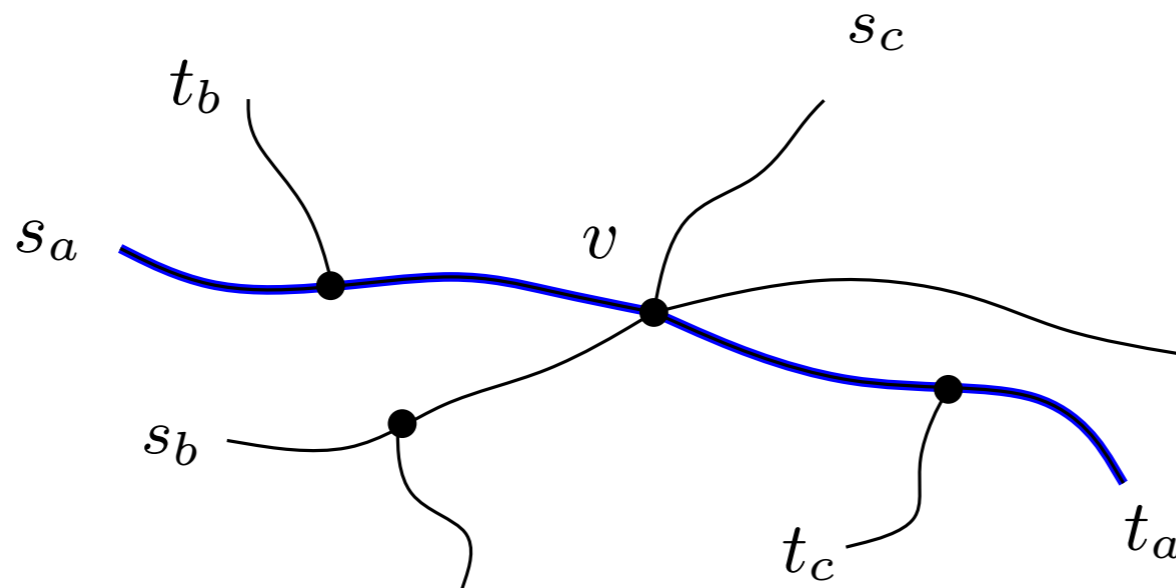


Greedy routes the request  $A$  with the smallest demand

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy

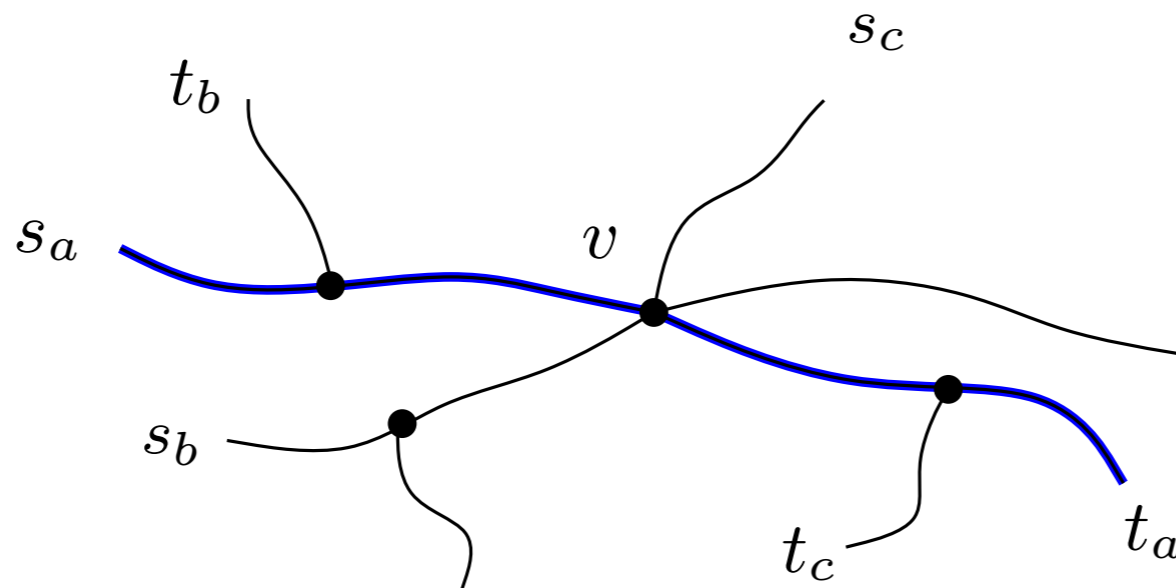


Greedy routes the request  $A$  with the smallest demand

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



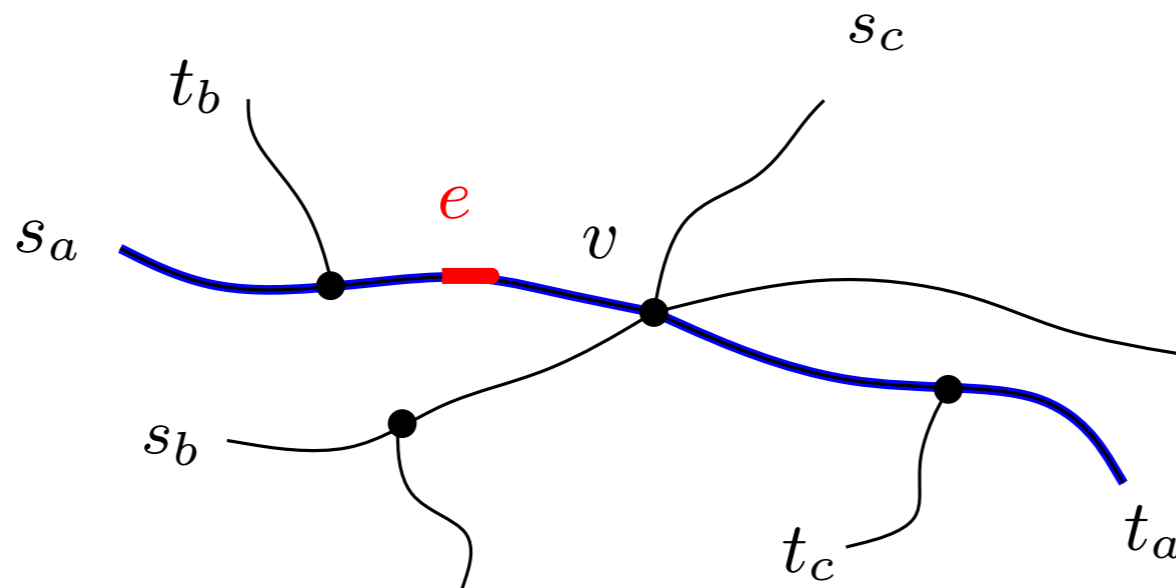
Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



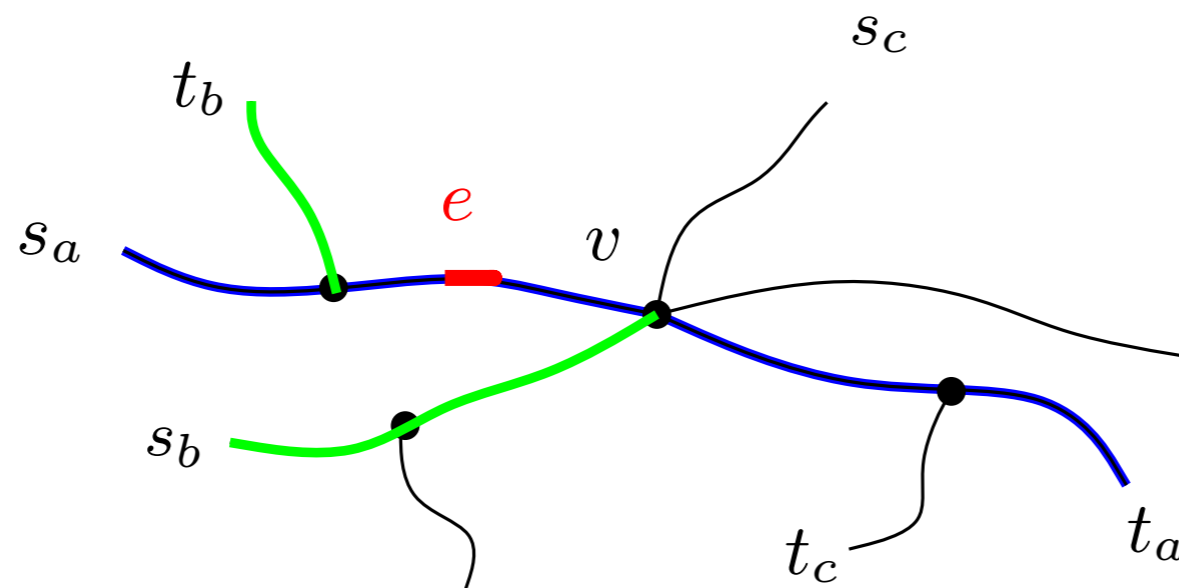
Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



$B$  uses  $e$

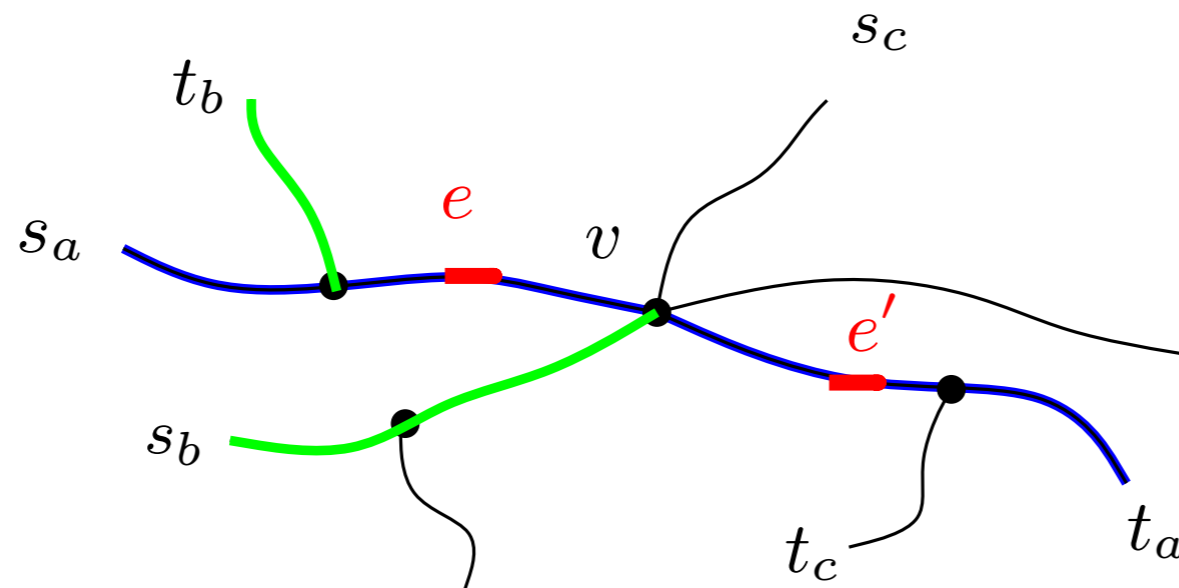
Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



$B$  uses  $e$

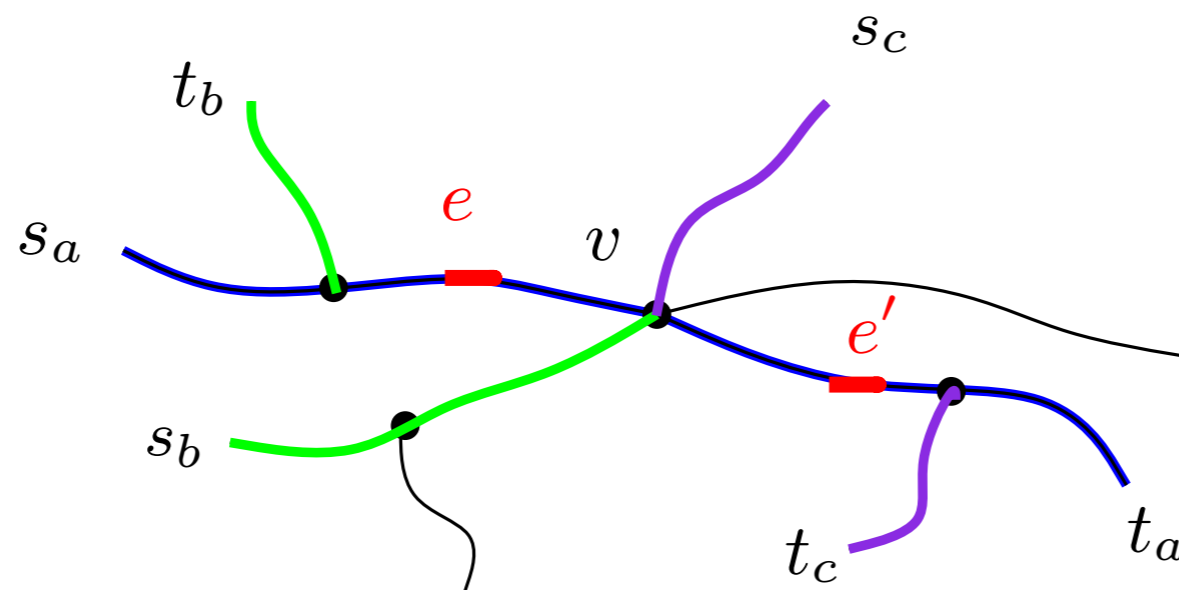
Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



$B$  uses  $e$   
 $C$  uses  $e'$

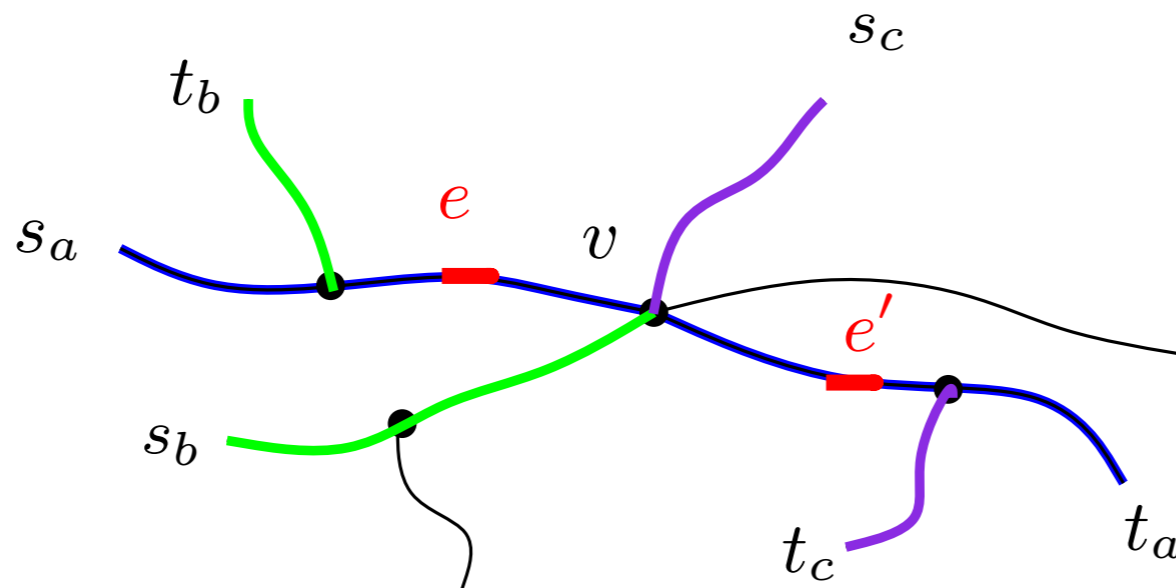
Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

# UFP on trees

**Lemma:** Greedy achieves a 2-approximation.

Swap two requests in optimal solution with a request selected by Greedy



$B$  uses  $e$

$C$  uses  $e'$

$$d_a \leq d_b$$

$$d_a \leq d_c$$

Greedy routes the request  $A$  with the smallest demand

Optimal solution routes (at most) two requests  $B, C$  that are "blocking"  $A$

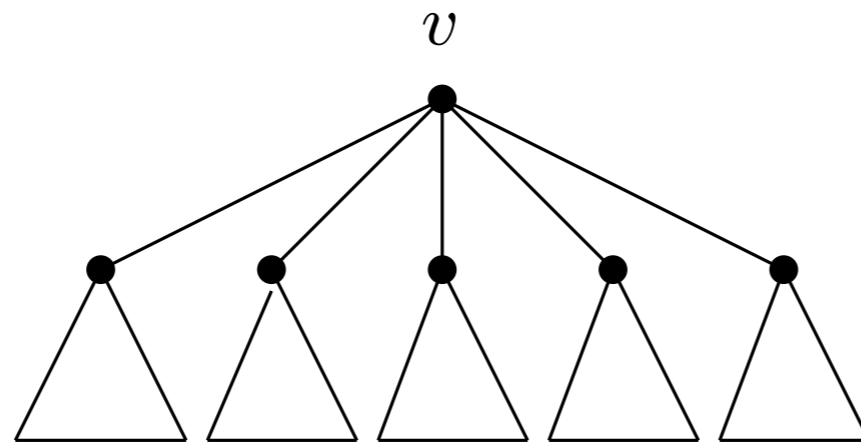
UFP on trees

# UFP on trees

**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices

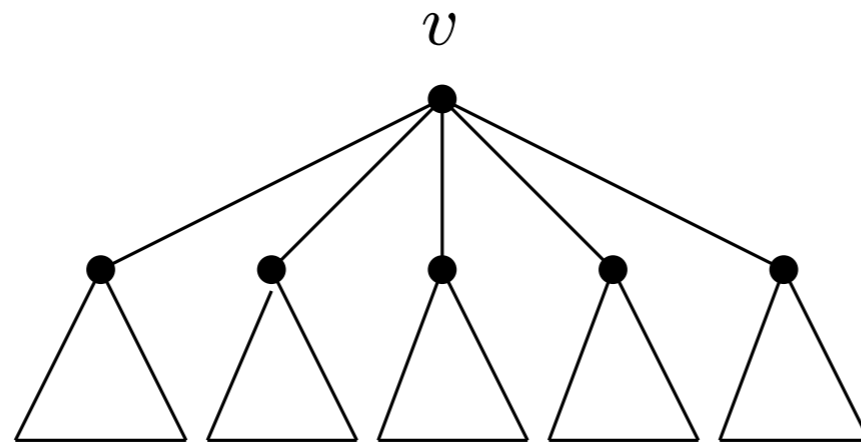
# UFP on trees

**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices



# UFP on trees

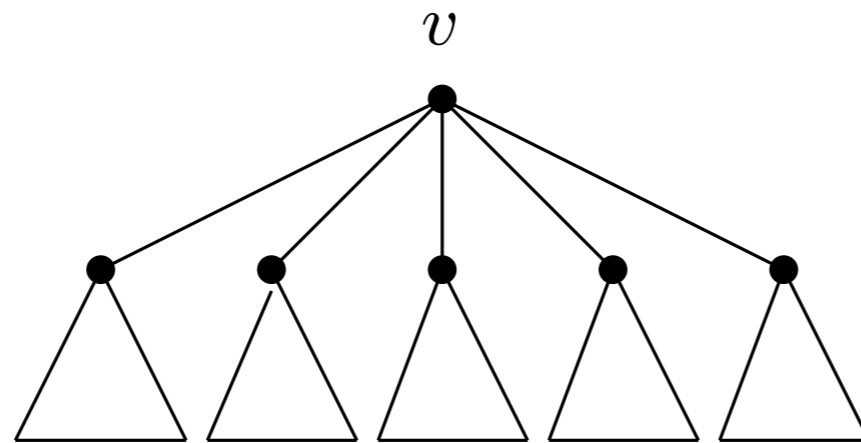
**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices



Many requests pass through  $v$ : use the previous algorithm

# UFP on trees

**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices

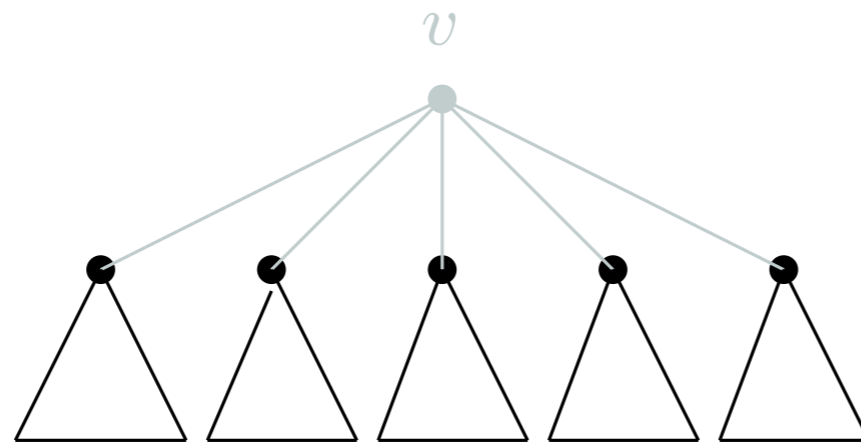


Many requests pass through  $v$ : use the previous algorithm

Many requests are contained in the components of  $T \setminus v$ : recurse

# UFP on trees

**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices

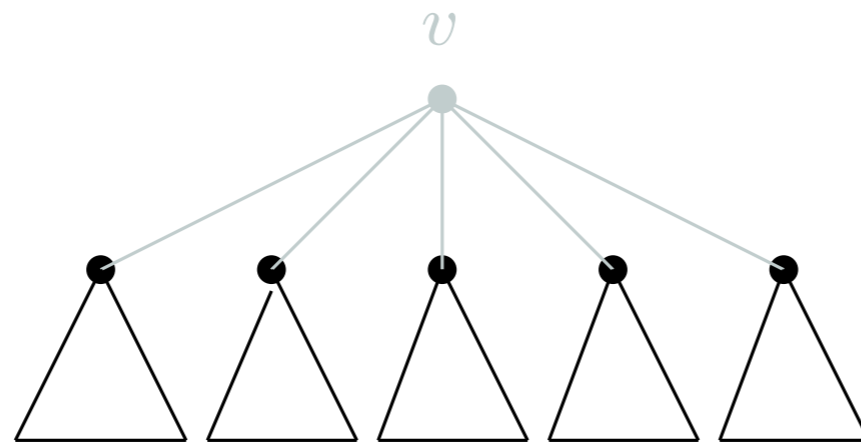


Many requests pass through  $v$ : use the previous algorithm

Many requests are contained in the components of  $T \setminus v$ : recurse

# UFP on trees

**Separator:** vertex  $v$  such that each connected component of  $T \setminus v$  has at most  $n/2$  vertices



$O(\log n)$ -approximation

Many requests pass through  $v$ : use the previous algorithm

Many requests are contained in the components of  $T \setminus v$ : recurse

UFP on trees

# UFP on trees

Using grouping and scaling, reduce arbitrary instances to **unit-profit** instances (we lose an  $O(\log n)$  factor)

**Theorem:** There exists an  $O(\log^2 n)$ -approximation algorithm for trees.

# LP relaxations

# LP relaxations

Recall standard LP:

**Standard LP**     $\max \sum_i w_i x_i$

$$\sum_{i: e \in P_i} d_i x_i \leq c_e \quad (\forall e \in E(G))$$

$$x_i \in [0, 1] \quad (\forall i \in \{1, \dots, k\})$$

# LP relaxations

Recall standard LP:

$$\begin{array}{ll} \text{Standard LP} & \max \sum_i w_i x_i \\ & \sum_{i: e \in P_i} d_i x_i \leq c_e \quad (\forall e \in E(G)) \\ & x_i \in [0, 1] \quad (\forall i \in \{1, \dots, k\}) \end{array}$$

**Bottleneck** for request  $R_i$ : min capacity edge on its path

# LP relaxations

Recall standard LP:

$$\begin{array}{ll} \text{Standard LP} & \max \sum_i w_i x_i \\ & \sum_{i: e \in P_i} d_i x_i \leq c_e \quad (\forall e \in E(G)) \\ & x_i \in [0, 1] \quad (\forall i \in \{1, \dots, k\}) \end{array}$$

**Bottleneck** for request  $R_i$ : min capacity edge on its path

Recall the bad example: each request uses the entire capacity of its bottleneck edge

# LP relaxations

Recall standard LP:

<b>Standard LP</b>	$\max \sum_i w_i x_i$	
	$\sum_{i: e \in P_i} d_i x_i \leq c_e$	$(\forall e \in E(G))$
	$x_i \in [0, 1]$	$(\forall i \in \{1, \dots, k\})$

**Bottleneck** for request  $R_i$ : min capacity edge on its path

Recall the bad example: each request uses the entire capacity of its bottleneck edge

**Theorem:** For any  $\delta > 0$ , the standard LP has  $O(\log(1/\delta)/\delta^3)$  gap for UFP on trees if the demand of each request is at most  $(1 - \delta)$  times the capacity of its bottleneck edge.

A new relaxation

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

- standard techniques fail
- interesting open question: relaxation with a good gap

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

- standard techniques fail
- interesting open question: relaxation with a good gap

Introduce constraints to handle "big" requests

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

- standard techniques fail
- interesting open question: relaxation with a good gap

Introduce constraints to handle "big" requests

Let  $\mathcal{B}_e$  be the set of all "big" requests that use the edge  $e$

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

- standard techniques fail
- interesting open question: relaxation with a good gap

Introduce constraints to handle "big" requests

Let  $\mathcal{B}_e$  be the set of all "big" requests that use the edge  $e$

For each set  $B \subseteq \mathcal{B}_e$ , let  $f(B)$  be the maximum **number** of requests in  $B$  that can be simultaneously routed

# A new relaxation

Requests that take up (almost) the entire capacity of an edge?

- standard techniques fail
- interesting open question: relaxation with a good gap

Introduce constraints to handle "big" requests

Let  $\mathcal{B}_e$  be the set of all "big" requests that use the edge  $e$

For each set  $B \subseteq \mathcal{B}_e$ , let  $f(B)$  be the maximum **number** of requests in  $B$  that can be simultaneously routed

**New constraints:** the total extent to which requests in  $B$  are selected by the LP must be at most  $f(S)$

A new relaxation

# A new relaxation

**UFP-LP**     $\max \sum_i w_i x_i$

$$\begin{array}{llll} \sum_{i: e \in P_i} d_i x_i & \leq & c_e & (\forall e \in E(G)) & \text{[capacity constraints]} \\ \sum_{R_i \in B} x_i & \leq & f(B) & (\forall e \in E(G), B \subseteq \mathcal{B}_e) & \text{[cardinality constraints]} \\ x_i & \in & [0, 1] & (\forall i \in \{1, \dots, k\}) & \end{array}$$

# A new relaxation

**UFP-LP**     $\max \sum_i w_i x_i$

$$\begin{array}{llll} \sum_{i: e \in P_i} d_i x_i & \leq & c_e & (\forall e \in E(G)) & \text{[capacity constraints]} \\ \sum_{R_i \in B} x_i & \leq & f(B) & (\forall e \in E(G), B \subseteq \mathcal{B}_e) & \text{[cardinality constraints]} \\ x_i & \in & [0, 1] & (\forall i \in \{1, \dots, k\}) & \end{array}$$

Solving UFP-LP is highly nontrivial

# A new relaxation

**UFP-LP**     $\max \sum_i w_i x_i$

$$\begin{array}{llll} \sum_{i: e \in P_i} d_i x_i & \leq & c_e & (\forall e \in E(G)) & \text{[capacity constraints]} \\ \sum_{R_i \in B} x_i & \leq & f(B) & (\forall e \in E(G), B \subseteq \mathcal{B}_e) & \text{[cardinality constraints]} \\ x_i & \in & [0, 1] & (\forall i \in \{1, \dots, k\}) & \end{array}$$

Solving UFP-LP is highly nontrivial

- Approximate separation oracle

# A new relaxation

**UFP-LP**     $\max \sum_i w_i x_i$

$$\begin{array}{llll} \sum_{i: e \in P_i} d_i x_i & \leq & c_e & (\forall e \in E(G)) & \text{[capacity constraints]} \\ \sum_{R_i \in B} x_i & \leq & f(B) & (\forall e \in E(G), B \subseteq \mathcal{B}_e) & \text{[cardinality constraints]} \\ x_i & \in & [0, 1] & (\forall i \in \{1, \dots, k\}) & \end{array}$$

Solving UFP-LP is highly nontrivial

- Approximate separation oracle
- Ellipsoid method

# A new relaxation

$$\mathbf{UFP-LP} \quad \max \sum_i w_i x_i$$

$$\begin{aligned} \sum_{i: e \in P_i} d_i x_i &\leq c_e & (\forall e \in E(G)) & \quad \text{[capacity constraints]} \\ \sum_{R_i \in B} x_i &\leq f(B) & (\forall e \in E(G), B \subseteq \mathcal{B}_e) & \quad \text{[cardinality constraints]} \\ x_i &\in [0, 1] & (\forall i \in \{1, \dots, k\}) & \end{aligned}$$

Solving UFP-LP is highly nontrivial

- Approximate separation oracle
- Ellipsoid method

**Theorem:** UFP-LP has  $O(\log^2 n)$  integrality gap.

# Separation Oracle

# Separation Oracle

If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

# Separation Oracle

If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

**Proof idea:** Given a set  $B$  such that  $x(B) > 18f(S)$ , find a subset  $S \subseteq B$  with simpler structure

# Separation Oracle

If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

**Proof idea:** Given a set  $B$  such that  $x(B) > 18f(S)$ , find a subset  $S \subseteq B$  with simpler structure

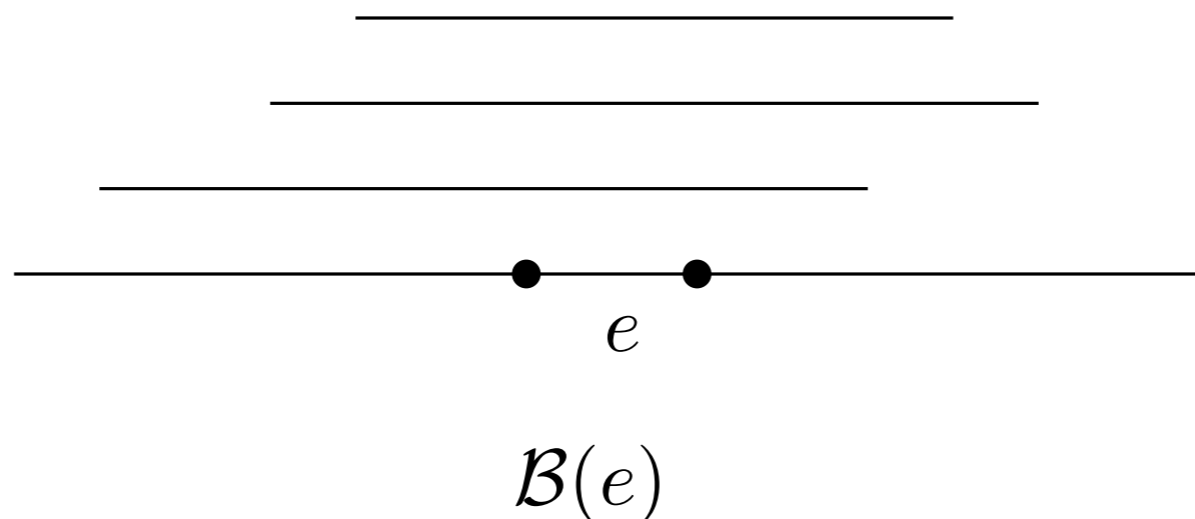
- $f(S) = 1$  and  $x(S) > 1$

# Separation Oracle

If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

**Proof idea:** Given a set  $B$  such that  $x(B) > 18f(S)$ , find a subset  $S \subseteq B$  with simpler structure

- $f(S) = 1$  and  $x(S) > 1$



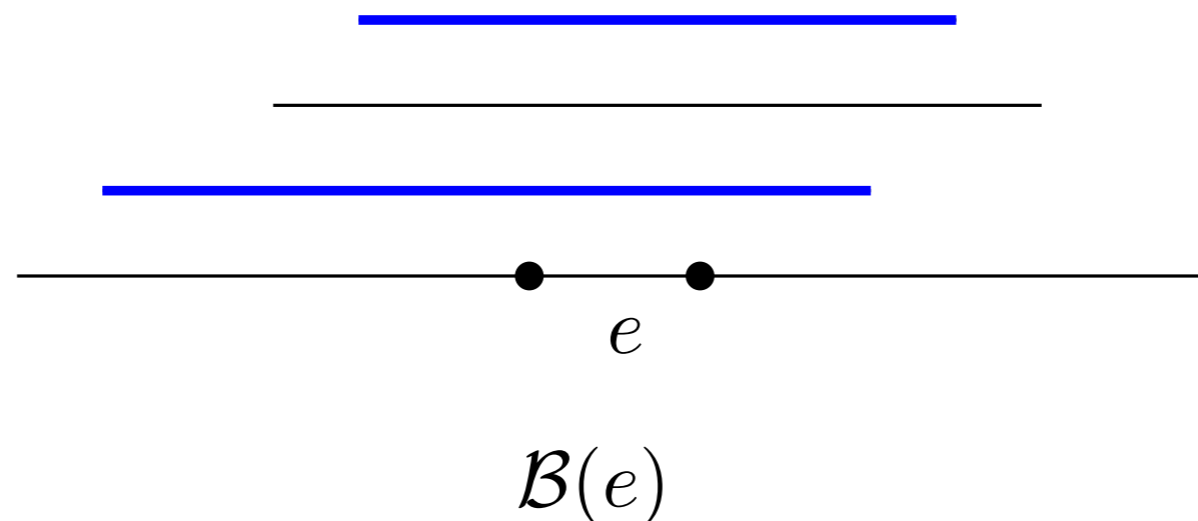
# Separation Oracle

If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

**Proof idea:** Given a set  $B$  such that  $x(B) > 18f(S)$ , find a subset  $S \subseteq B$  with simpler structure

- $f(S) = 1$  and  $x(S) > 1$

$\mathcal{B}_{\text{left}}(e)$



# Separation Oracle

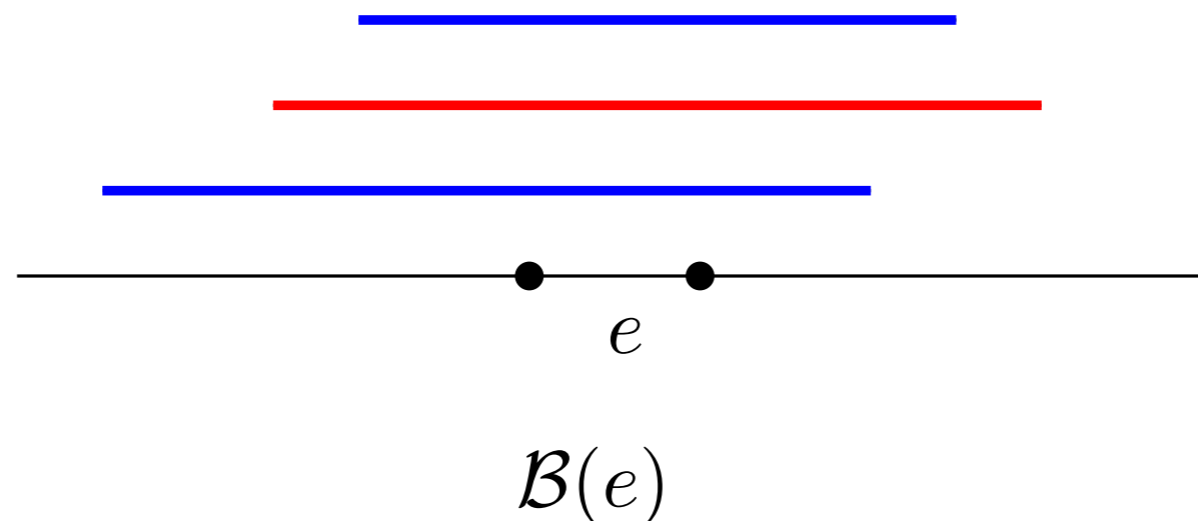
If there is a cardinality constraint that is violated by a (small) constant factor, we can find a violated constraint

**Proof idea:** Given a set  $B$  such that  $x(B) > 18f(S)$ , find a subset  $S \subseteq B$  with simpler structure

- $f(S) = 1$  and  $x(S) > 1$

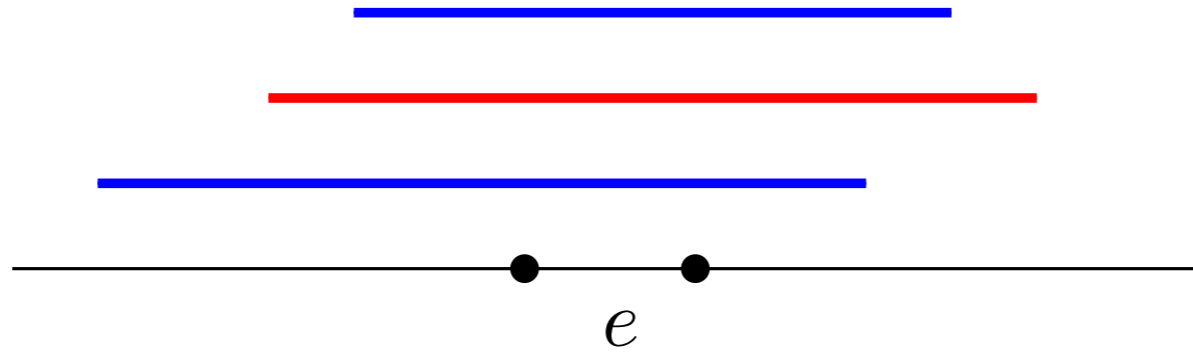
$\mathcal{B}_{\text{left}}(e)$

$\mathcal{B}_{\text{right}}(e)$



# Separation Oracle

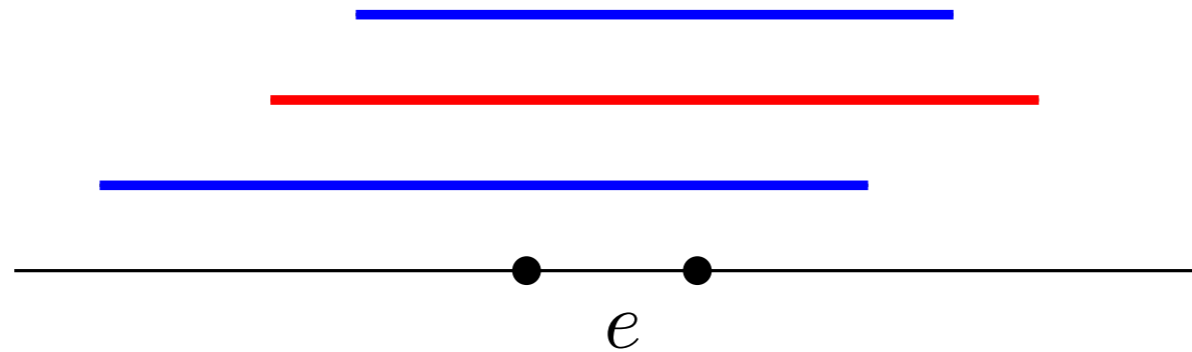
$\mathcal{B}_{\text{left}}(e)$



$\mathcal{B}_{\text{right}}(e)$

# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$



$\mathcal{B}_{\text{right}}(e)$

## Separation Oracle

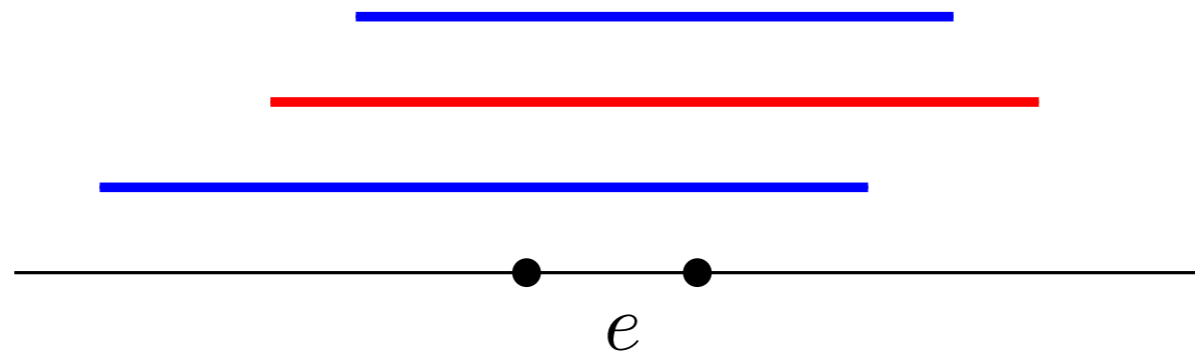
for each request  $R_i$  in  $\mathcal{B}_{\text{left}}(e)$

let  $S = \{R_i\} \cup \{R_j \mid R_j \in \mathcal{B}_{\text{left}}(e), d_j > d_i, R_i \text{ blocks } R_j\}$

if  $x(S) > 1$  return  $S$

# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$



$\mathcal{B}_{\text{right}}(e)$

## Separation Oracle

for each request  $R_i$  in  $\mathcal{B}_{\text{left}}(e)$

let  $S = \{R_i\} \cup \{R_j \mid R_j \in \mathcal{B}_{\text{left}}(e), d_j > d_i, R_i \text{ blocks } R_j\}$

if  $x(S) > 1$  return  $S$

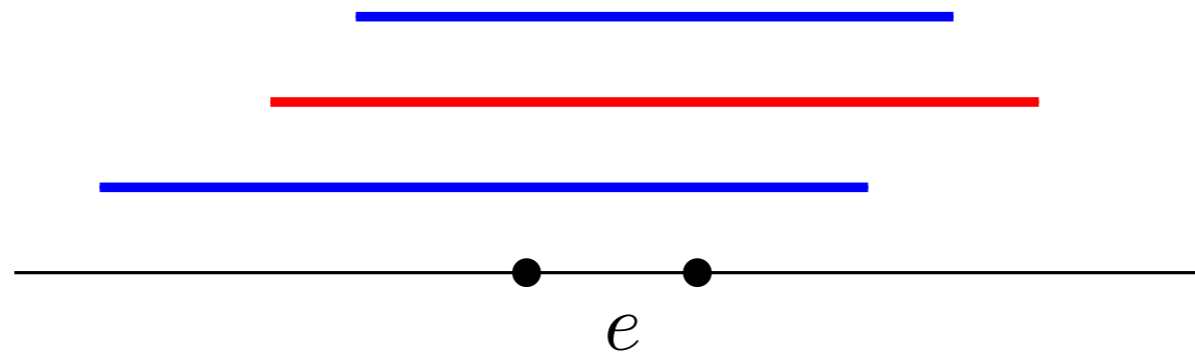
for each request  $R_i$  in  $\mathcal{B}_{\text{right}}(e)$

let  $S = \{R_i\} \cup \{R_j \mid R_j \in \mathcal{B}_{\text{right}}(e), d_j > d_i, R_i \text{ blocks } R_j\}$

if  $x(S) > 1$  return  $S$

# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$



$\mathcal{B}_{\text{right}}(e)$

## Separation Oracle

for each request  $R_i$  in  $\mathcal{B}_{\text{left}}(e)$

let  $S = \{R_i\} \cup \{R_j | R_j \in \mathcal{B}_{\text{left}}(e), d_j > d_i, R_i \text{ blocks } R_j\}$

if  $x(S) > 1$  return  $S$   $f(S) = 1$

for each request  $R_i$  in  $\mathcal{B}_{\text{right}}(e)$

let  $S = \{R_i\} \cup \{R_j | R_j \in \mathcal{B}_{\text{right}}(e), d_j > d_i, R_i \text{ blocks } R_j\}$

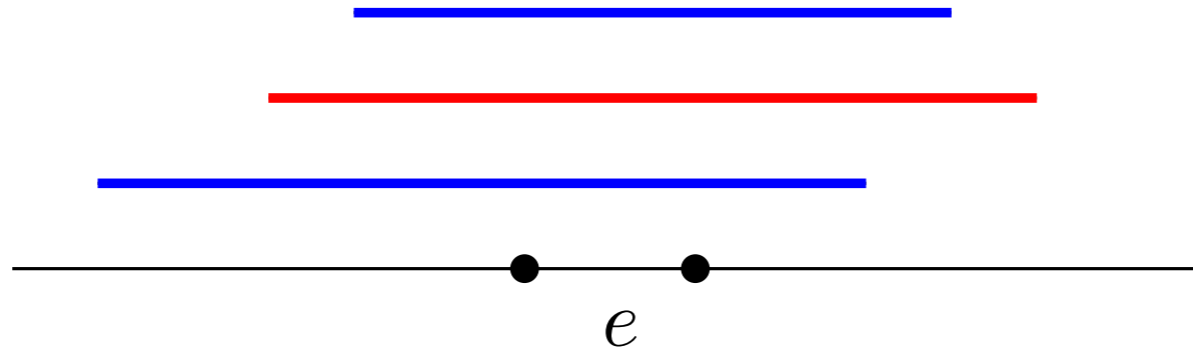
if  $x(S) > 1$  return  $S$   $f(S) = 1$

# Separation Oracle

# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$

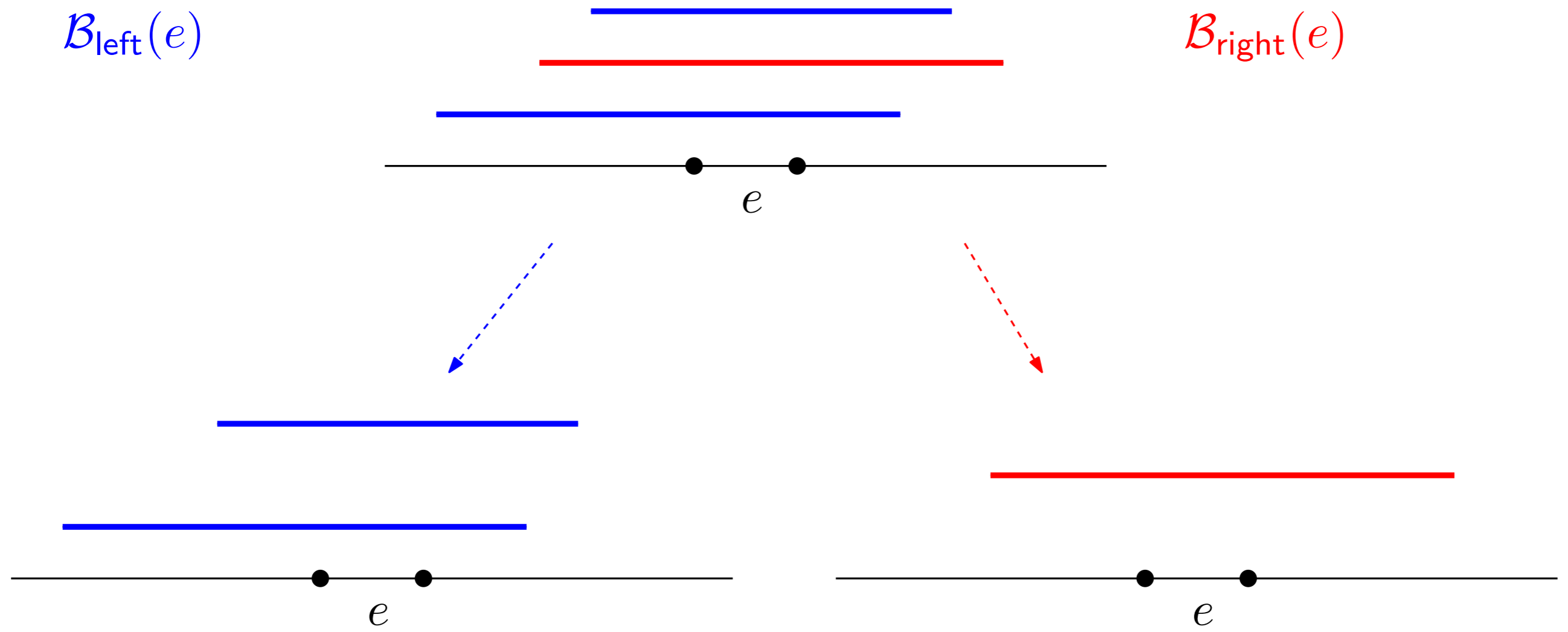
$\mathcal{B}_{\text{right}}(e)$



# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$

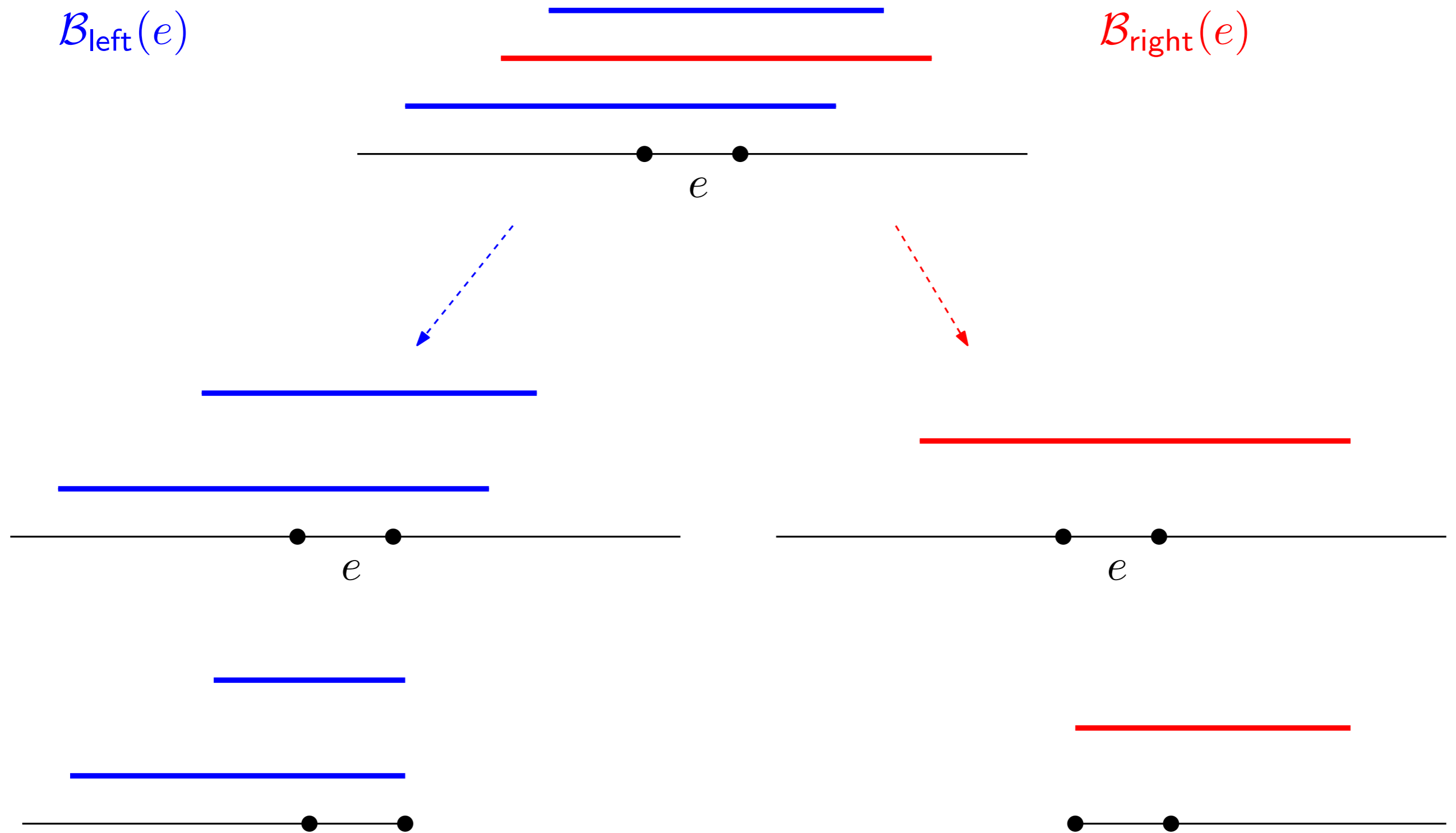
$\mathcal{B}_{\text{right}}(e)$



# Separation Oracle

$\mathcal{B}_{\text{left}}(e)$

$\mathcal{B}_{\text{right}}(e)$



# Separation Oracle

# Separation Oracle



"left"

$$x(B') > 9f(B')$$

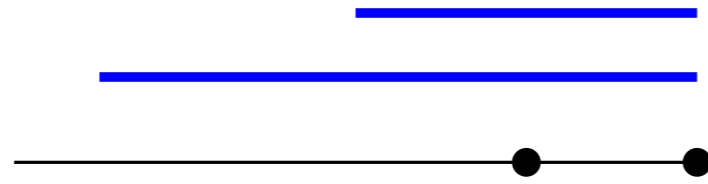
*or*



"right"

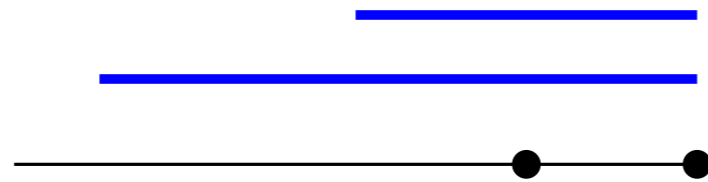
$$x(B'') > 9f(B'')$$

# Separation Oracle



$$x(B) > 9f(B)$$

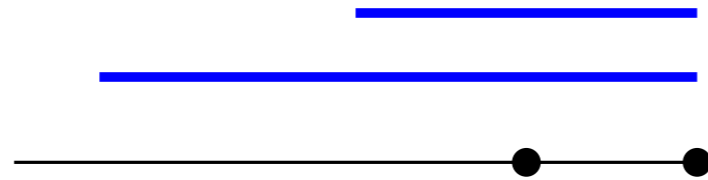
# Separation Oracle



$$x(B) > 9f(B)$$

Given  $B$  s.t.  $x(B) > 9f(S)$ , find a subset  $S \subseteq B$  with simpler structure

# Separation Oracle



$$x(B) > 9f(B)$$

Given  $B$  s.t.  $x(B) > 9f(S)$ , find a subset  $S \subseteq B$  with simpler structure

**Theorem:** There exists a set  $S$  with simpler structure ( $f(S) = 1, x(S) > 1$ )

# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

- Guess the smallest demand request  $R_i$  in  $S$

# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$

# Separation Oracle

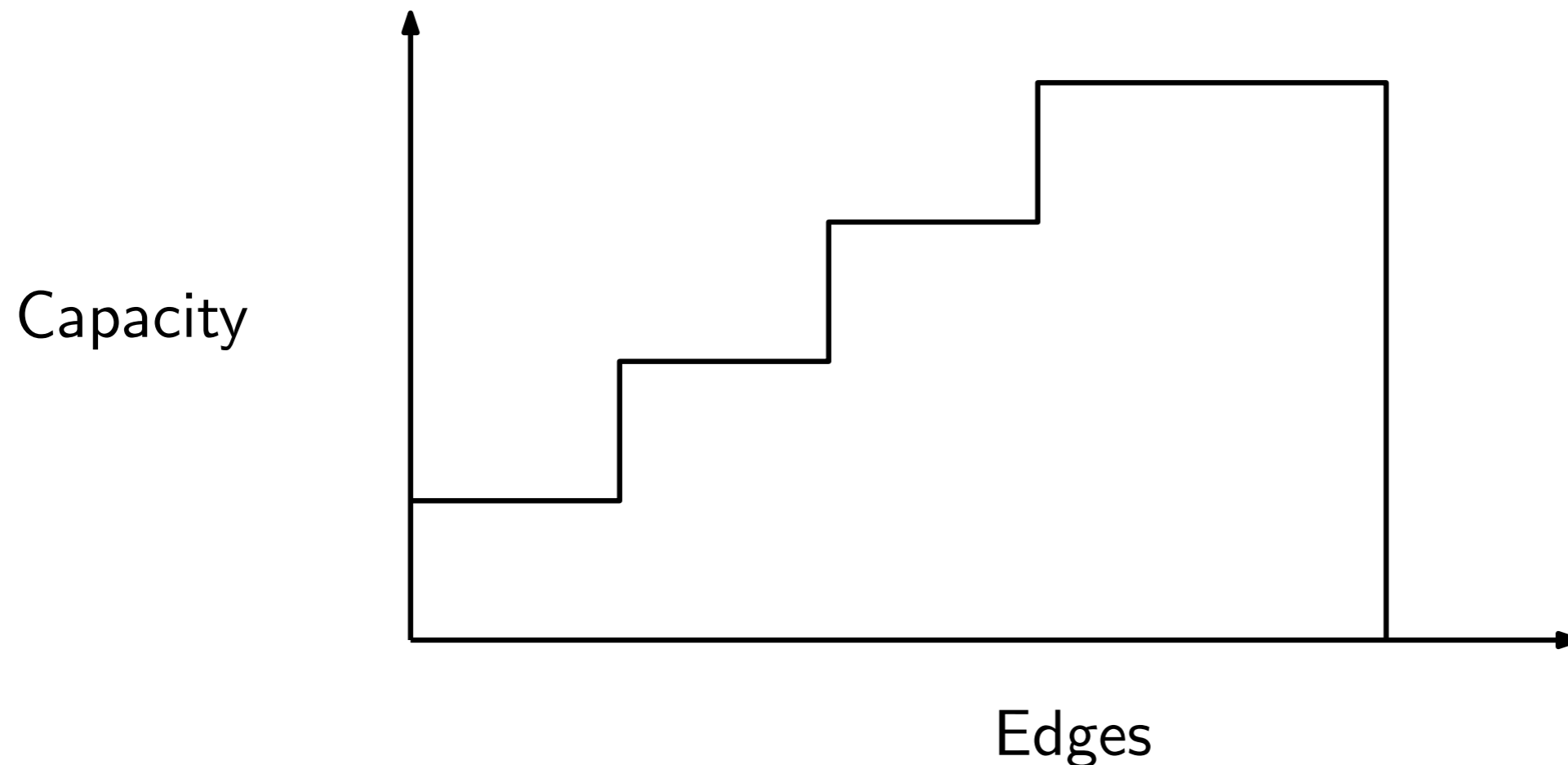
There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$   $S \subseteq S'$

# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

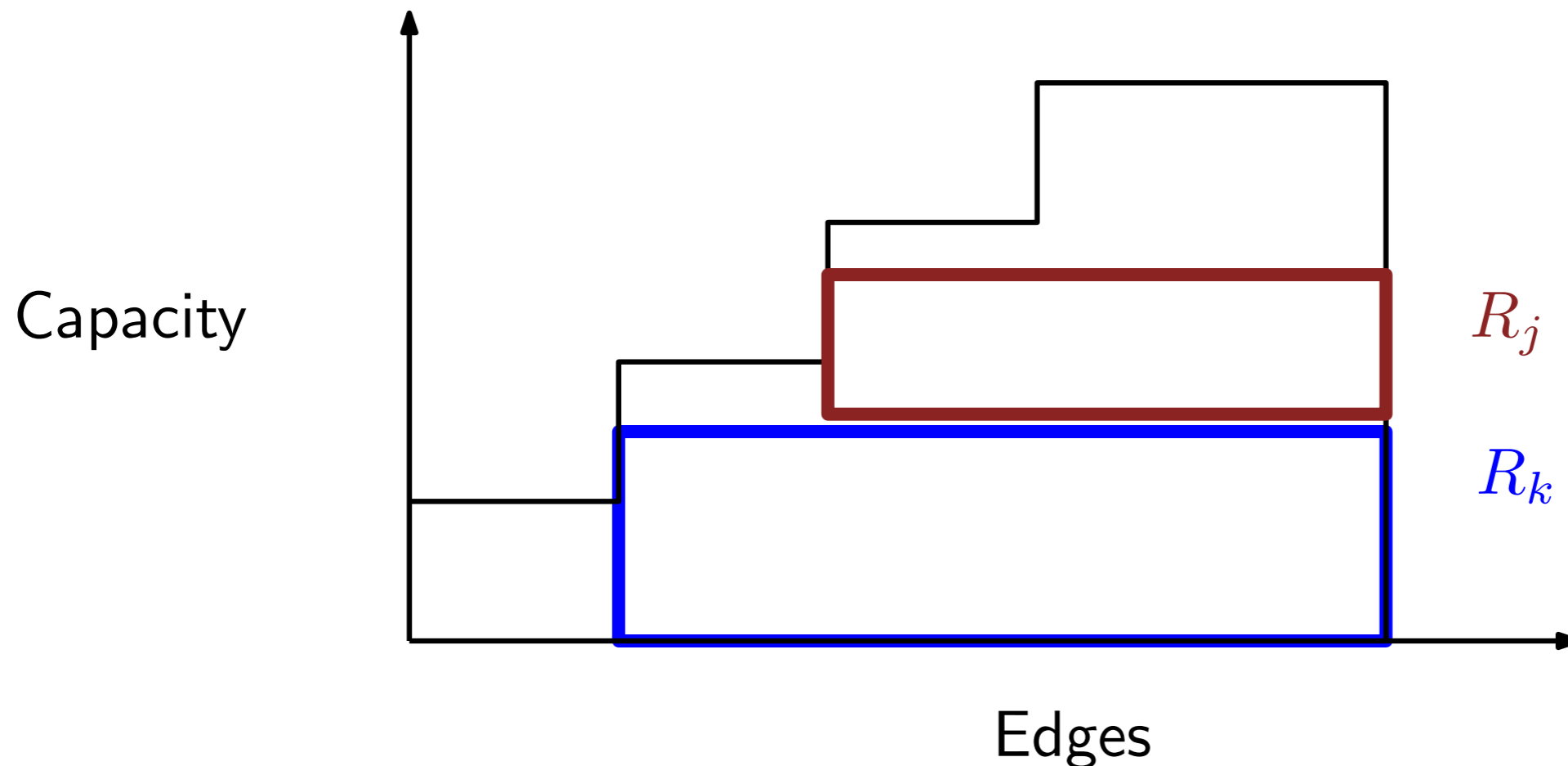
- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$   $S \subseteq S'$



# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

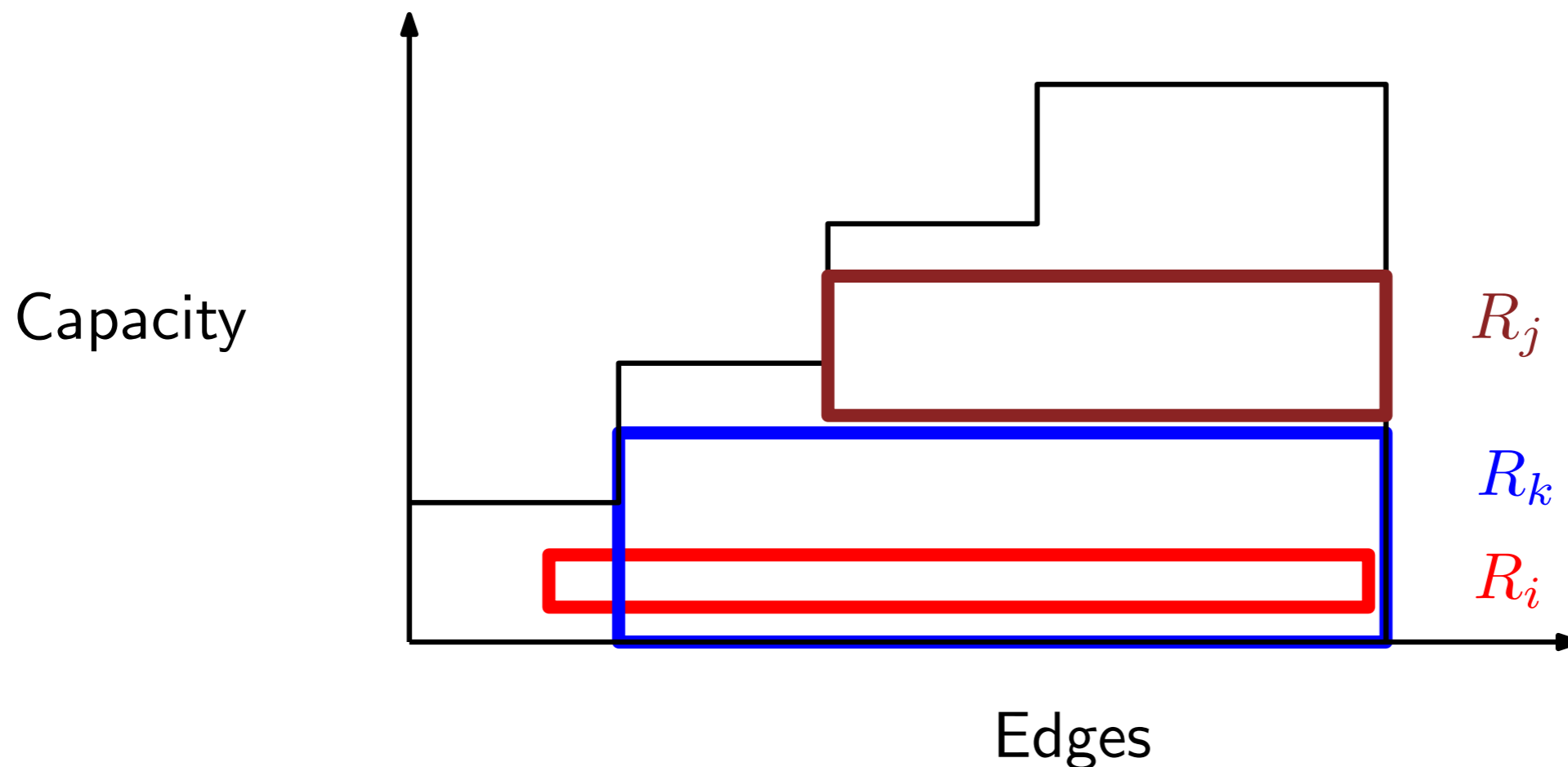
- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$   $S \subseteq S'$



# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

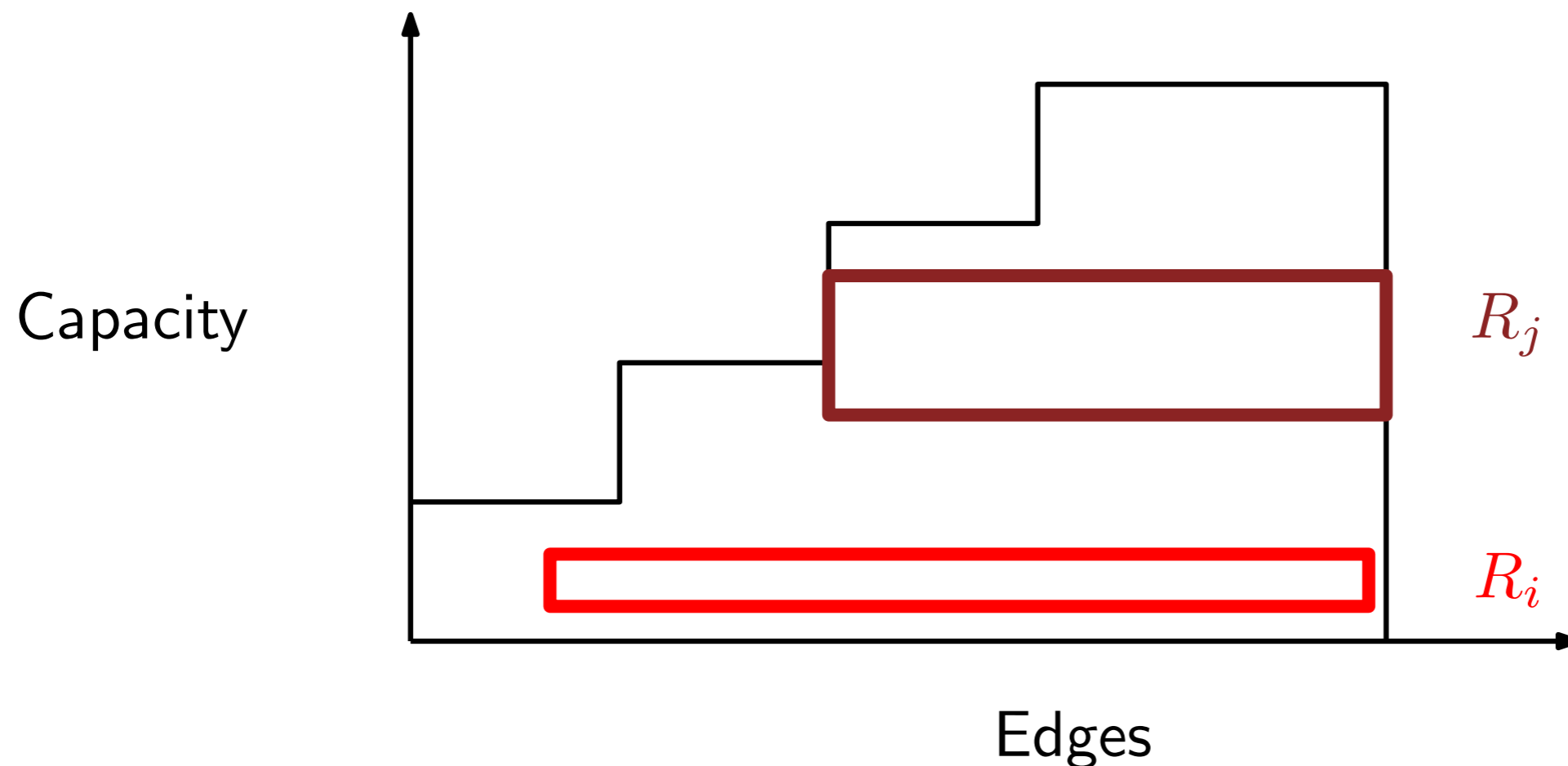
- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$   $S \subseteq S'$



# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

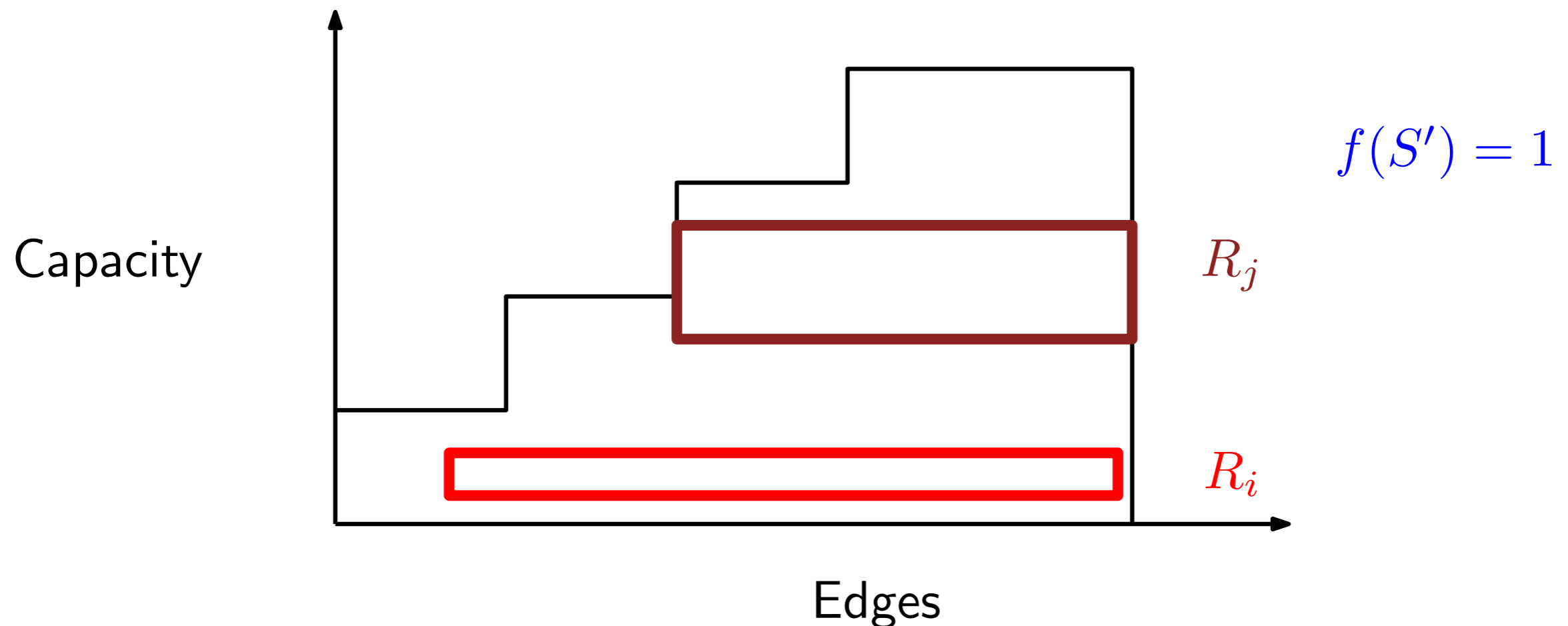
- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$   $S \subseteq S'$



# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

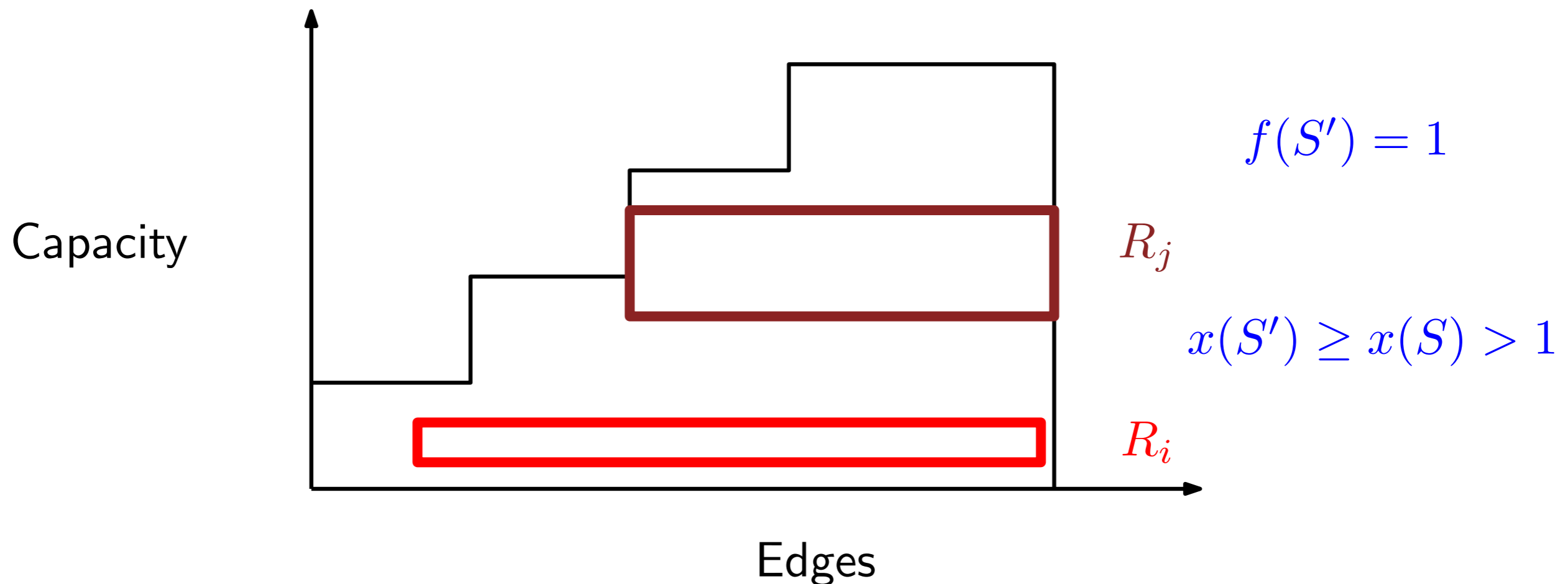
- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$        $S \subseteq S'$



# Separation Oracle

There exists  $S$  s.t.  $f(S) = 1$  and  $x(S) > 1$

- Guess the smallest demand request  $R_i$  in  $S$
- $S'$ :  $R_i$  and requests  $R_j$  s.t.  $d_j \geq d_i$  and  $R_i$  blocks  $R_j$        $S \subseteq S'$



# Open Problems

Bansal et al. gave an  $O(\log n)$ -approximation for paths.

- Is there an  $O(\log n)$ -approximation for trees as well?
- Is there an  $O(1)$ -approximation for paths?

What is the integrality gap of the new LP relaxation?

- We conjecture that it is much better than  $O(\log^2 n)$

The new relaxation can be extended to UFP on trees

- Can we separate over the LP?

THANK YOU