# Multicut and Integer Multicommodity Flow in Trees

Piotr Wojciechowski[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

April 17, 2014

# Outline

1. Multicut and Integer Multicommodity Flow in Trees
   - Definitions
   - IP formulation
   - Algorithm

# Outline

1. Multicut and Integer Multicommodity Flow in Trees
   - Definitions
   - IP formulation
   - Algorithm

**Definition (Minimum Multicut)**

We are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a non-negative capacity $c_e$ and a set of set of vertex pairs $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$.

**Definition (Minimum Multicut)**

We are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a non-negative capacity $c_e$ and a set of set of vertex pairs $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. The *m*inimum multicut is the set of edges with minimum total capacity that separates each $s_i$ from its corresponding $t_i$.

**Definition (Minimum Multicut)**

We are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a non-negative capacity $c_e$ and a set of set of vertex pairs $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. The *m*inimum multicut is the set of edges with minimum total capacity that separates each $s_i$ from its corresponding $t_i$.

**Definition (Maximum Integer Multicommodity Flow)**

We are given a graph $G$ and set of vertex pairs $S$ as in the multicut problem.

**Definition (Minimum Multicut)**

We are given an undirected graph $G = (V, E)$ where each edge $e \in E$ has a non-negative capacity $c_e$ and a set of set of vertex pairs $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. The *m*inimum multicut is the set of edges with minimum total capacity that separates each $s_i$ from its corresponding $t_i$.

**Definition (Maximum Integer Multicommodity Flow)**

We are given a graph $G$ and set of vertex pairs $S$ as in the multicut problem. If each $(s_i, t_i)$ pair is assigned its own commodity then the goal is to maximize the total flow of the system when an integral amount each commodity is routed.

# Outline

1. Multicut and Integer Multicommodity Flow in Trees
   - Definitions
   - IP formulation
   - Algorithm

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees.

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees. Since $G$ is a tree there is a unique path, $p_i$, from $s_i$ to $t_i$ and removing any edge in $p_i$ constitutes a $s_i, t_i$ cut.

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees. Since $G$ is a tree there is a unique path, $p_i$, from $s_i$ to $t_i$ and removing any edge in $p_i$ constitutes a $s_i, t_i$ cut.

## Minimum Multicut

The minimum multicut problem for trees can be formulated as the following IP:

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees. Since $G$ is a tree there is a unique path, $p_i$, from $s_i$ to $t_i$ and removing any edge in $p_i$ constitutes a $s_i, t_i$ cut.

## Minimum Multicut

The minimum multicut problem for trees can be formulated as the following IP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees. Since $G$ is a tree there is a unique path, $p_i$, from $s_i$ to $t_i$ and removing any edge in $p_i$ constitutes a $s_i, t_i$ cut.

## Minimum Multicut

The minimum multicut problem for trees can be formulated as the following IP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

## Restriction

We are only looking at finding the minimum multicut and maximum integer multicommodity flow for trees. Since $G$ is a tree there is a unique path, $p_i$, from $s_i$ to $t_i$ and removing any edge in $p_i$ constitutes a $s_i, t_i$ cut.

## Minimum Multicut

The minimum multicut problem for trees can be formulated as the following IP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

$$x_e \in \{0, 1\} \qquad e \in E$$

## LP relaxation

This can be relaxed to the following LP:

## LP relaxation

This can be relaxed to the following LP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

## LP relaxation

This can be relaxed to the following LP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

## LP relaxation

This can be relaxed to the following LP:

$$\begin{aligned} \min \sum_{e \in E} c_e \cdot x_e & \\ \sum_{e \in p_i} x_e \geq 1 & \quad i \in \{1, \ldots, k\} \\ x_e \geq 0 & \quad e \in E \end{aligned}$$

## LP relaxation

This can be relaxed to the following LP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

$$x_e \geq 0 \qquad e \in E$$

## Dual

We can construct the dual LP which is:

## LP relaxation

This can be relaxed to the following LP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

$$x_e \geq 0 \qquad e \in E$$

## Dual

We can construct the dual LP which is:

$$\max \sum_{i=1}^{k} f_i$$

## LP relaxation

This can be relaxed to the following LP:

$$\min \sum_{e \in E} c_e \cdot x_e$$

$$\sum_{e \in p_i} x_e \geq 1 \qquad i \in \{1, \ldots, k\}$$

$$x_e \geq 0 \qquad e \in E$$

## Dual

We can construct the dual LP which is:

$$\max \sum_{i=1}^{k} f_i$$

$$\sum_{i : e \in p_i} f_i \leq c_e \qquad e \in E$$

## LP relaxation

This can be relaxed to the following LP:

$$\begin{aligned} \min \sum_{e \in E} c_e \cdot x_e & \\ \sum_{e \in p_i} x_e \geq 1 & \quad i \in \{1, \ldots, k\} \\ x_e \geq 0 & \quad e \in E \end{aligned}$$

## Dual

We can construct the dual LP which is:

$$\begin{aligned} \max \sum_{i=1}^{k} f_i & \\ \sum_{i : e \in p_i} f_i \leq c_e & \quad e \in E \\ f_i \geq 0 & \quad i \in \{1, \ldots, k\} \end{aligned}$$

## LP relaxation

This can be relaxed to the following LP:

$$\begin{aligned} \min \sum_{e \in E} c_e \cdot x_e & \\ \sum_{e \in p_i} x_e \geq 1 & \quad i \in \{1, \ldots, k\} \\ x_e \geq 0 & \quad e \in E \end{aligned}$$

## Dual

We can construct the dual LP which is:

$$\begin{aligned} \max \sum_{i=1}^{k} f_i & \\ \sum_{i : e \in p_i} f_i \leq c_e & \quad e \in E \\ f_i \geq 0 & \quad i \in \{1, \ldots, k\} \end{aligned}$$

This is precisely the fractional version of the multicommodity flow problem.

## Primal-Dual Schema

We will construct a primal-dual schema based algorithm that will find a multicut and integer multicommodity flow that are within a factor 2 of each other.

## Primal-Dual Schema

We will construct a primal-dual schema based algorithm that will find a multicut and integer multicommodity flow that are within a factor 2 of each other. Thus this algorithm will serve as both a factor 2 approximation for the minimum multicut and a factor $\frac{1}{2}$ approximation for the maximum integer multicommodity flow.

## Primal-Dual Schema

We will construct a primal-dual schema based algorithm that will find a multicut and integer multicommodity flow that are within a factor 2 of each other. Thus this algorithm will serve as both a factor 2 approximation for the minimum multicut and a factor $\frac{1}{2}$ approximation for the maximum integer multicommodity flow.
We will treat multicut as the primal problem and ensure primal complementary slackness, $\alpha = 1$, while relaxing dual complementary slackness, $\beta = 2$.

## Primal-Dual Schema

We will construct a primal-dual schema based algorithm that will find a multicut and integer multicommodity flow that are within a factor 2 of each other. Thus this algorithm will serve as both a factor 2 approximation for the minimum multicut and a factor $\frac{1}{2}$ approximation for the maximum integer multicommodity flow.
We will treat multicut as the primal problem and ensure primal complementary slackness, $\alpha = 1$, while relaxing dual complementary slackness, $\beta = 2$.

## Primal Conditions

For each edge $e \in E$, we must have that $x_e \neq 0 \rightarrow \sum_{i:e \in p_i} f_i = c_e$.

## Primal-Dual Schema

We will construct a primal-dual schema based algorithm that will find a multicut and integer multicommodity flow that are within a factor 2 of each other. Thus this algorithm will serve as both a factor 2 approximation for the minimum multicut and a factor $\frac{1}{2}$ approximation for the maximum integer multicommodity flow.
We will treat multicut as the primal problem and ensure primal complementary slackness, $\alpha = 1$, while relaxing dual complementary slackness, $\beta = 2$.

## Primal Conditions

For each edge $e \in E$, we must have that $x_e \neq 0 \rightarrow \sum_{i:e \in p_i} f_i = c_e$.

## Relaxed Dual Conditions

For each $i \in \{1, \ldots, k\}$, we must have that $f_i \neq 0 \rightarrow \sum_{e \in p_i} x_e \leq 2$.

# Outline

1. Multicut and Integer Multicommodity Flow in Trees
   - Definitions
   - IP formulation
   - Algorithm

## Root

Choose an arbitrary vertex, $r$, in $G$ to be the root.

**Root**

Choose an arbitrary vertex, $r$, in $G$ to be the root.

**Definition (Depth (vertex))**

The *depth* of a vertex $v \in V$ is the length of the unique path from $v$ to $r$.

**Root**

Choose an arbitrary vertex, $r$, in $G$ to be the root.

**Definition (Depth (vertex))**

The *depth* of a vertex $v \in V$ is the length of the unique path from $v$ to $r$.

**Definition (Lowest Common Ancestor)**

The *lowest common ancestor* of a pair of verticies $u, v \in V$, $lca(u, v)$, is the vertex along the unique path from $u$ to $v$ of minimum depth.

**Root**

Choose an arbitrary vertex, $r$, in $G$ to be the root.

**Definition (Depth (vertex))**

The *depth* of a vertex $v \in V$ is the length of the unique path from $v$ to $r$.

**Definition (Lowest Common Ancestor)**

The *lowest common ancestor* of a pair of verticies $u, v \in V$, $lca(u, v)$, is the vertex along the unique path from $u$ to $v$ of minimum depth.

**Definition (Depth (edge))**

Given two edges $e_1$ and $e_2$ along the path from a vertex $v$ to the root, $r$, the edge $e_1$ is *deeper* than $e_2$ if $e_1$ occurs before $e_2$ along the path.

**Note**

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

## Note

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

## Algorithm

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.

**Note**

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

**Algorithm**

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.
2. For each vertex $v$ in non-increasing order of depth.

**Note**

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

**Algorithm**

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.
2. For each vertex $v$ in non-increasing order of depth.
   1. For each pair $(s_i, t_i)$ such that $lca(s_i, t_i) = v$, greedily route integral flow from $s_i$ to $t_i$.

## Note

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

## Algorithm

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.
2. For each vertex $v$ in non-increasing order of depth.
   1. For each pair $(s_i, t_i)$ such that $lca(s_i, t_i) = v$, greedily route integral flow from $s_i$ to $t_i$.
   2. Add all saturated edges to $D$.

## Note

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

## Algorithm

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.
2. For each vertex $v$ in non-increasing order of depth.
   1. For each pair $(s_i, t_i)$ such that $lca(s_i, t_i) = v$, greedily route integral flow from $s_i$ to $t_i$.
   2. Add all saturated edges to $D$.
3. Let $e_1, \ldots e_l$ be the order in which the edges were added to $D$.

## Note

In the algorithm **f** represents the multicommodity flow and $D$ represents the edges in the multicut.

## Algorithm

1. $\mathbf{f} = \mathbf{0}$ and $D = \emptyset$.
2. For each vertex $v$ in non-increasing order of depth.
   1. For each pair $(s_i, t_i)$ such that $lca(s_i, t_i) = v$, greedily route integral flow from $s_i$ to $t_i$.
   2. Add all saturated edges to $D$.
3. Let $e_1, \ldots e_l$ be the order in which the edges were added to $D$.
4. For each $j = l, \ldots, 1$ if removing $e_j$ from $D$ preserves the cut remove it.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.
- We must have that $u = lca(s_j, t_j)$ is deeper in the tree than $v$.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.
- We must have that $u = lca(s_j, t_j)$ is deeper in the tree than $v$.
- When the algorithm processed the vertex $u$ an edge $e_3$ along the path from $s_j$ to $t_j$ must have been added to $D$.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.
- We must have that $u = lca(s_j, t_j)$ is deeper in the tree than $v$.
- When the algorithm processed the vertex $u$ an edge $e_3$ along the path from $s_j$ to $t_j$ must have been added to $D$.
- Later non-zero flow was routed through $e_1$ so $e_3 \neq e_1$.

**Theorem**

*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.
- We must have that $u = lca(s_j, t_j)$ is deeper in the tree than $v$.
- When the algorithm processed the vertex $u$ an edge $e_3$ along the path from $s_j$ to $t_j$ must have been added to $D$.
- Later non-zero flow was routed through $e_1$ so $e_3 \neq e_1$.
- $e_1$ was added to $D$ after $e_3$ so would have been considered for removal earlier.

**Theorem**

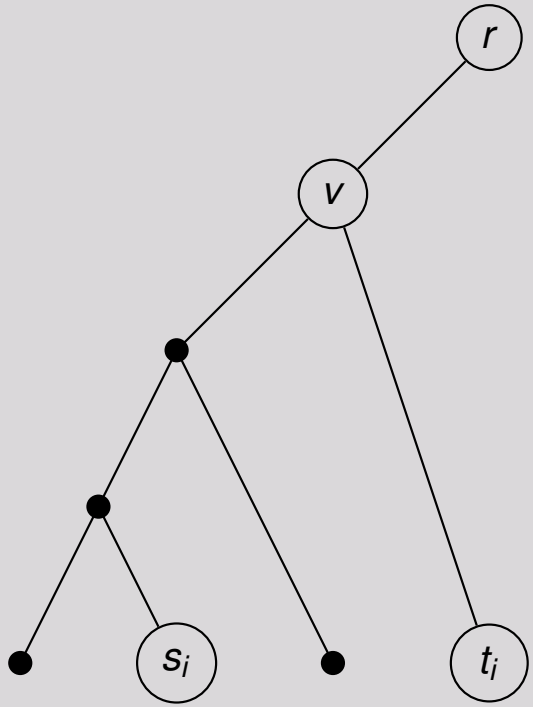
*Let $(s_i, t_i)$ be a pair of verticies such that $f_i \neq 0$ and let $v = lca(s_i, t_i)$. We have that at most one edge on the path from $s_i$ to $v$ is in $D$ and at most one edge from $t_i$ to $v$ is in $D$.*

**Proof.**

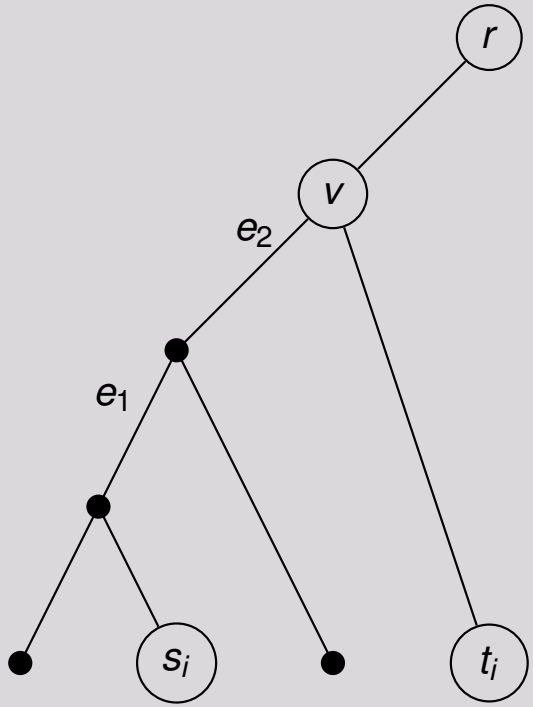
- Suppose otherwise, thus without loss of generality we can assume that the path from $s_i$ to $v$ contains at least 2 edges, $e_1$ and $e_2$.
- Let $e_1$ be the deeper of the two edges.
- Consider the moment in the algorithm when $e_1$ is being considered for removal
- It was not removed there must exist a pair of verticies $(s_j, t_j)$ such that $e_1$ is the only edge in $D$ along the path from $s_i$ to $t_i$.
- $e_2$ is not along this path.
- We must have that $u = lca(s_j, t_j)$ is deeper in the tree than $v$.
- When the algorithm processed the vertex $u$ an edge $e_3$ along the path from $s_j$ to $t_j$ must have been added to $D$.
- Later non-zero flow was routed through $e_1$ so $e_3 \neq e_1$.
- $e_1$ was added to $D$ after $e_3$ so would have been considered for removal earlier.
- This contradicts the fact that it was the only edge along the path from $s_j$ to $t_j$.

□

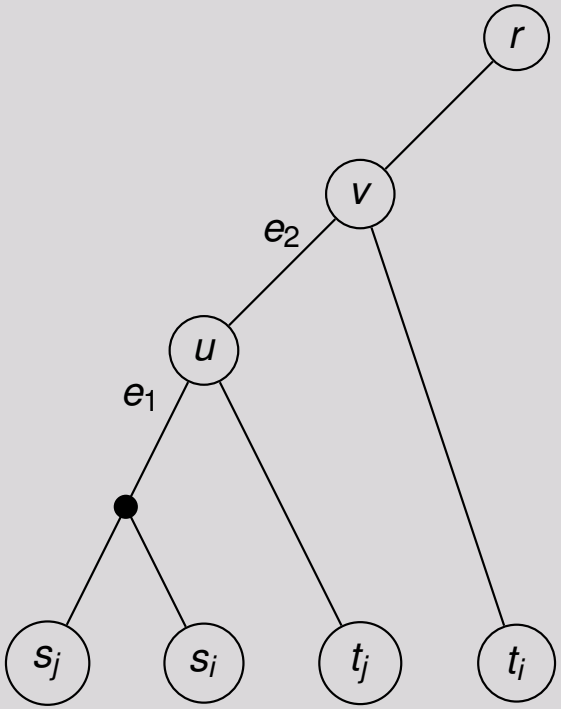
## Example



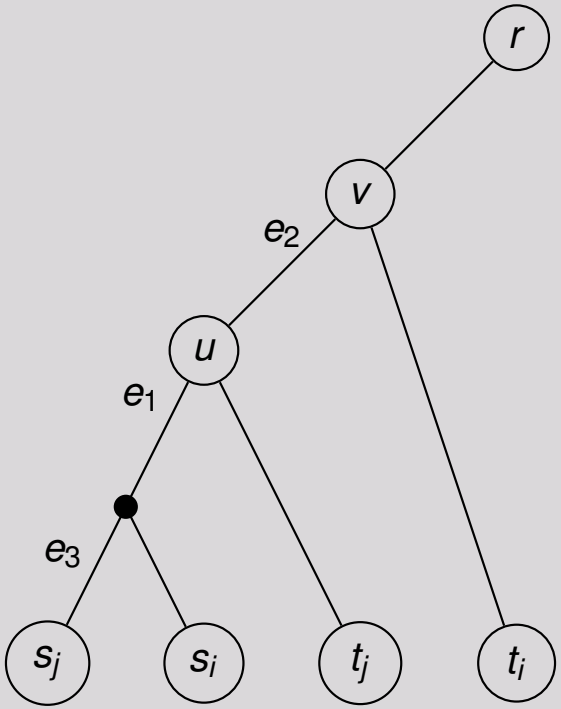
## Example



## Example



## Example



**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.
- $D$ contains all saturated edges and so is a valid multicut.

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.
- $D$ contains all saturated edges and so is a valid multicut.
- We remove edges from $D$ only if it still remains a valid multicut.

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.
- $D$ contains all saturated edges and so is a valid multicut.
- We remove edges from $D$ only if it still remains a valid multicut.
- Each edge in $D$ is saturated so the primal slackness condition is satisfied.

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.
- $D$ contains all saturated edges and so is a valid multicut.
- We remove edges from $D$ only if it still remains a valid multicut.
- Each edge in $D$ is saturated so the primal slackness condition is satisfied.
- Each path with non-zero flow has at most two edges in $D$ so the relaxed dual slackness condition is satisfied.

**Theorem**

*The algorithm obtains an approximation factor of 2 for multicut and $\frac{1}{2}$ for integer multicommodity flows.*

**Proof.**

- The flow generated after all vericies have been processed is maximal.
- $D$ contains all saturated edges and so is a valid multicut.
- We remove edges from $D$ only if it still remains a valid multicut.
- Each edge in $D$ is saturated so the primal slackness condition is satisfied.
- Each path with non-zero flow has at most two edges in $D$ so the relaxed dual slackness condition is satisfied.
- The capacity of the multicut is at most twice the flow.

□