LP-guided branching, part 1: The Multiway Cut problems

Marcin Pilipczuk, Michał Pilipczuk

Finse 1222,
March 20$^{\mathrm{th}}$, 2014

Multiway Cut

Edge Multiway Cut

Input: A graph G with some terminals $T \subseteq V(G)$, an integer k

Question: Is there a set of edges F with $|F| \leq k$, such that every path between two terminals is hit by F?

Multiway Cut

Edge Multiway Cut

Input: A graph G with some terminals $T \subseteq V(G)$, an integer k

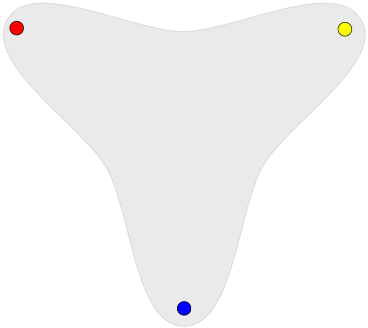
Question: Is there a set of edges F with $|F| \leq k$, such that every path between two terminals is hit by F?

Node Multiway Cut

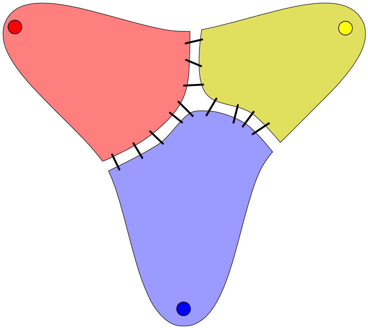
Input: A graph G with some terminals $T \subseteq V(G)$, an integer k

Question: Is there a set of vertices $X \subseteq V(G) \setminus T$ with $|X| \leq k$, s.t. every path between two terminals is hit by X?

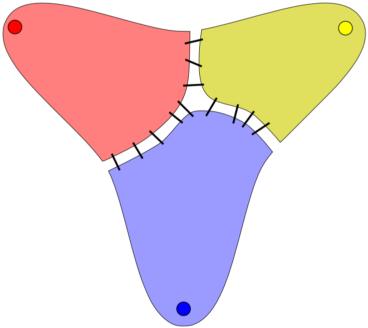
On the picture



On the picture



On the picture



If G is connected and F is optimum,
then every vertex is reachable from some terminal.

Edge Multiway Cut

- For $|T| = 2$ it is just an edge flow problem $\Rightarrow$ P-time solvable.

Edge Multiway Cut

- For $|T| = 2$ it is just an edge flow problem $\Rightarrow$ P-time solvable.
- NP-hard for $|T| \geq 3$.

Edge Multiway Cut

- For $|T| = 2$ it is just an edge flow problem $\Rightarrow$ P-time solvable.
- NP-hard for $|T| \geq 3$.
- **Goal**: $\mathcal{O}^\star(2^k)$ algorithm for Edge Multiway Cut.

Edge Multiway Cut

- For $|T| = 2$ it is just an edge flow problem $\Rightarrow$ P-time solvable.
- NP-hard for $|T| \geq 3$.
- **Goal**: $\mathcal{O}^\star(2^k)$ algorithm for Edge Multiway Cut.
  - This algorithm is due to Xiao.

When $|T| = 2$

- Let $T = \{s, t\}$.

When $|T| = 2$

- Let $T = \{s, t\}$.
- An (s, t) *cut* is a subset $A \subseteq V(G)$ such that $s \in A$, $t \notin A$.

When $|T| = 2$

- Let $T = \{s, t\}$.
- An (s, t) *cut* is a subset $A \subseteq V(G)$ such that $s \in A$, $t \notin A$.
- The *cutset* is $\Delta(A) := E(A, \overline{A})$. Denote $\delta(A) = |\Delta(A)|$.

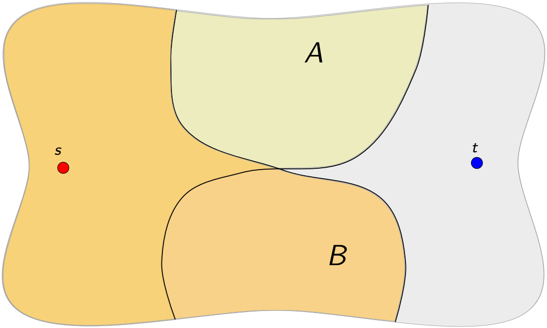
When $|T| = 2$

- Let $T = \{s, t\}$.
- An (s,t) *cut* is a subset $A \subseteq V(G)$ such that $s \in A$, $t \notin A$.
- The *cutset* is $\Delta(A) := E(A, \overline{A})$. Denote $\delta(A) = |\Delta(A)|$.
- An (s,t) cut A is *minimum* if $\delta(A)$ is minimum possible. The size of the minimum cut can be determined in polynomial time.

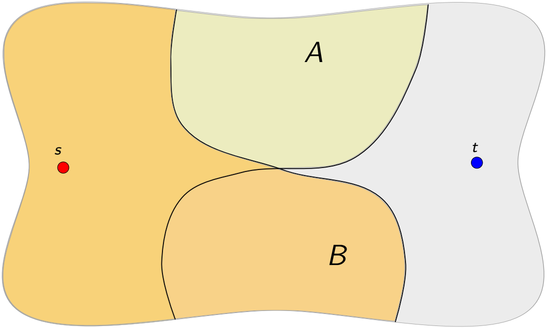
When $|T| = 2$

- Let $T = \{s, t\}$.
- An (s, t) *cut* is a subset $A \subseteq V(G)$ such that $s \in A$, $t \notin A$.
- The *cutset* is $\Delta(A) := E(A, \overline{A})$. Denote $\delta(A) = |\Delta(A)|$.
- An (s, t) cut A is *minimum* if $\delta(A)$ is minimum possible. The size of the minimum cut can be determined in polynomial time.
- **Note**: If (A, B) is a minimum cut, then A and B are connected. Hence $A = \texttt{reach}(s, G \setminus F)$, where F is the cutset.

Submodularity

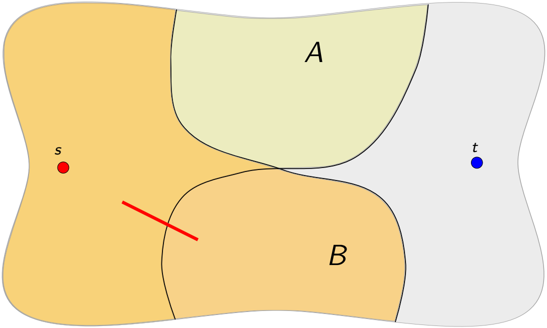


Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

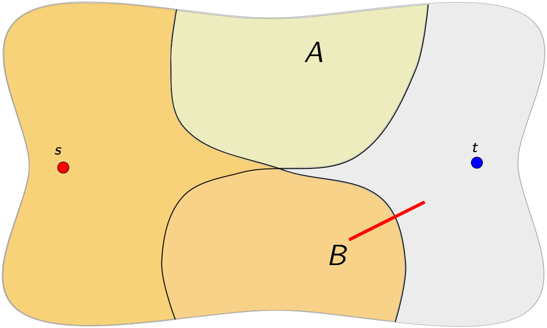
Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

1 0 0 1

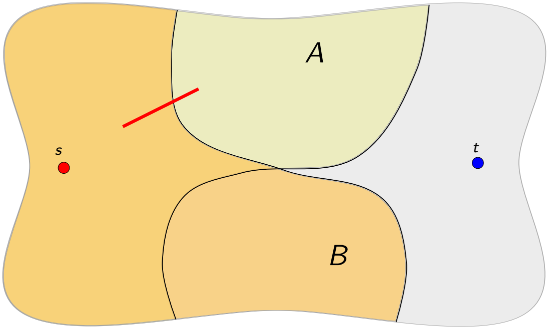
Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

0 1 1 0

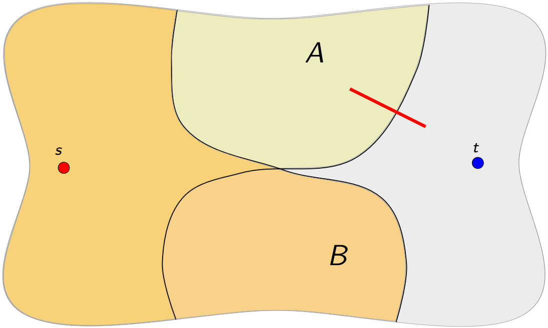
Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

0 1 0 1

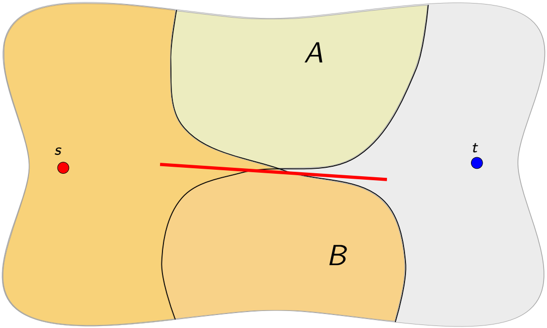
Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

1 0 1 0

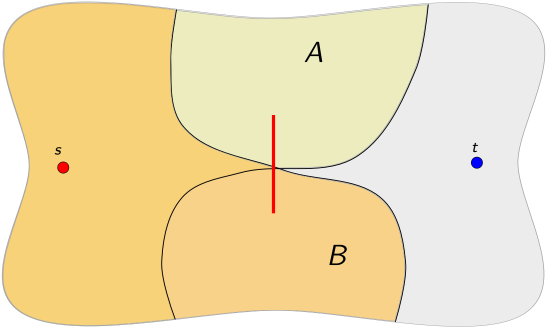
Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

1 1 1 1

Submodularity



$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$

1 1 0 0

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s, t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s, t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s, t) cuts A and B.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s,t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s,t) cuts A and B.
- Submodularity $\Rightarrow \delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s, t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s, t) cuts A and B.
- Submodularity $\Rightarrow \delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$.
- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s, t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s, t) cuts A and B.
- Submodularity $\Rightarrow \delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$.
- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.
- Hence $\delta(B) \geq \delta(A \cup B)$.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s,t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s,t) cuts A and B.
- Submodularity $\Rightarrow \delta(A)+\delta(B) \geq \delta(A \cup B)+\delta(A \cap B)$.
- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.
- Hence $\delta(B) \geq \delta(A \cup B)$.
- B is minimum $\Rightarrow A \cup B$ is a minimum (s,t) cut.

Furthest and closest cuts

Extreme minimum cuts

Among all minimum (s,t) cuts, there exists a unique cut A that is inclusion-wise maximal, and a unique one that is inclusion-wise minimal.

- **Proof**: Take any two minimum (s,t) cuts A and B.
- Submodularity $\Rightarrow \delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$.
- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.
- Hence $\delta(B) \geq \delta(A \cup B)$.
- B is minimum $\Rightarrow A \cup B$ is a minimum (s,t) cut.
- Symmetrical reasoning for $A \cap B$. □

Furthest and closest cuts

- These inclusion-wise maximal/minimal min-cuts are called *furthest from s* and *closest to s*, respectively.

Furthest and closest cuts

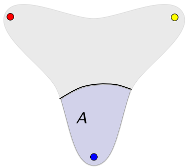
- These inclusion-wise maximal/minimal min-cuts are called *furthest from* s and *closest to* s, respectively.
- Any sensible max-flow algorithm can provide these cuts within the same running time.

Furthest and closest cuts

- These inclusion-wise maximal/minimal min-cuts are called *furthest from* s and *closest to* s, respectively.
- Any sensible max-flow algorithm can provide these cuts within the same running time.
- Naturally generalizes to S and T being sets of sources and sinks.

Min-cut reduction for EDGE MULTIWAY CUT

- Pick a terminal t, and let A be the $(\{t\}, T \setminus \{t\})$ min-cut that is furthest from t.

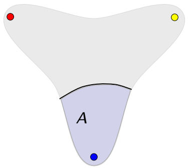


Min-cut reduction for EDGE MULTIWAY CUT

- Pick a terminal t, and let A be the $(\{t\}, T \setminus \{t\})$ min-cut that is furthest from t.

Lemma

There exists an optimal solution to the instance that does not include any edge with both endpoints in A.



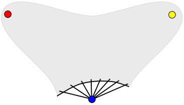
Min-cut reduction for EDGE MULTIWAY CUT

- Pick a terminal t, and let A be the $(\{t\}, T \setminus \{t\})$ min-cut that is furthest from t.

Lemma

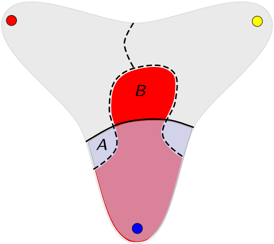
There exists an optimal solution to the instance that does not include any edge with both endpoints in A.

- **Note**: If the lemma is true, then it is safe to contract the whole set A onto t.



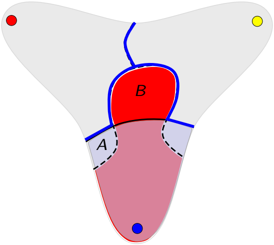
Min-cut reduction, proof

- Let F be any opt. solution, and let $B = \texttt{reach}(t, G \setminus F)$.



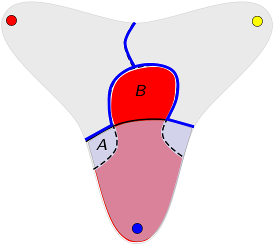
Min-cut reduction, proof

- Let F be any opt. solution, and let $B = \texttt{reach}(t, G \setminus F)$.
- Construct F' from F by selling every edge with both endpoints in $A \cup B$, and buying the remainder of $\Delta(A \cup B)$.



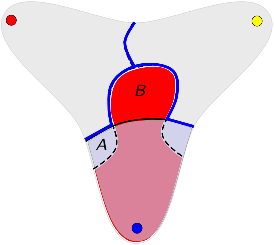
Min-cut reduction, proof

- Let F be any opt. solution, and let $B = \texttt{reach}(t, G \setminus F)$.
- Construct F' from F by selling every edge with both endpoints in $A \cup B$, and buying the remainder of $\Delta(A \cup B)$.
  - **Check (1).** $|F'| \leq |F|$.



Min-cut reduction, proof

- Let F be any opt. solution, and let $B = \texttt{reach}(t, G \setminus F)$.
- Construct F' from F by selling every edge with both endpoints in $A \cup B$, and buying the remainder of $\Delta(A \cup B)$.
  - **Check (1)**. $|F'| \leq |F|$.
  - **Check (2)**. F' is still a solution.



Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.

Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.
- It suffices to show that

$$|E(B, A \setminus B)| \geq |E(A \setminus B, \overline{A \cup B})|.$$

Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.
- It suffices to show that

$$|E(B, A \setminus B)| \geq |E(A \setminus B, \overline{A \cup B})|.$$

- By adding $|E(B, \overline{A \cup B})|$ to both sides, equivalently:

$$\delta(B) \geq \delta(A \cup B).$$

Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.
- It suffices to show that

$$|E(B, A \setminus B)| \geq |E(A \setminus B, \overline{A \cup B})|.$$

- By adding $|E(B, \overline{A \cup B})|$ to both sides, equivalently:

$$\delta(B) \geq \delta(A \cup B).$$

- Submodularity:

$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B).$$

Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.
- It suffices to show that

$$|E(B, A \setminus B)| \geq |E(A \setminus B, \overline{A \cup B})|.$$

- By adding $|E(B, \overline{A \cup B})|$ to both sides, equivalently:

$$\delta(B) \geq \delta(A \cup B).$$

- Submodularity:

$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B).$$

- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.

Min-cut reduction, proof of (1)

- We have sold at least $E(B, A \setminus B)$, and bought $E(A \setminus B, \overline{A \cup B})$.
- It suffices to show that

$$|E(B, A \setminus B)| \geq |E(A \setminus B, \overline{A \cup B})|.$$

- By adding $|E(B, \overline{A \cup B})|$ to both sides, equivalently:

$$\delta(B) \geq \delta(A \cup B).$$

- Submodularity:

$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B).$$

- A is minimum $\Rightarrow \delta(A) \leq \delta(A \cap B)$.
- Hence $\delta(B) \geq \delta(A \cup B)$ and we are done.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.
- F is an optimum solution; let uv be any edge of P that is in F.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.
- F is an optimum solution; let uv be any edge of P that is in F.
- $uv \in F \setminus F'$, so $u, v \in A \cup B$.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.
- F is an optimum solution; let uv be any edge of P that is in F.
- $uv \in F \setminus F'$, so $u, v \in A \cup B$.
- $G[A \cup B]$ is connected, so t is reachable from u in $G[A \cup B]$.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.
- F is an optimum solution; let uv be any edge of P that is in F.
- $uv \in F \setminus F'$, so $u, v \in A \cup B$.
- $G[A \cup B]$ is connected, so t is reachable from u in $G[A \cup B]$.
- $P[t', u]$ avoids $F' \supseteq \Delta(A \cup B)$.

Min-cut reduction, proof of (2)

- Assume P is a t'-t'' path untouched by F' such that $t' \neq t$.
- F is an optimum solution; let uv be any edge of P that is in F.
- $uv \in F \setminus F'$, so $u, v \in A \cup B$.
- $G[A \cup B]$ is connected, so t is reachable from u in $G[A \cup B]$.
- $P[t', u]$ avoids $F' \supseteq \Delta(A \cup B)$.
- Ergo there is a t'-t path avoiding $\Delta(A \cup B)$, a contradiction.

Second reduction rule

- **Second reduction rule.** If $tt' \in E(G)$ for some $t, t' \in T$, then remove tt' and decrease the budget by 1.

Second reduction rule

- **Second reduction rule.** If $tt' \in E(G)$ for some $t, t' \in T$, then remove tt' and decrease the budget by 1.
- Assume both the reduction rules are applied exhaustively.

Second reduction rule

- **Second reduction rule.** If $tt' \in E(G)$ for some $t, t' \in T$, then remove tt' and decrease the budget by 1.
- Assume both the reduction rules are applied exhaustively.
- Then the sets $\Delta(\{t\})$ for $t \in T$ are:

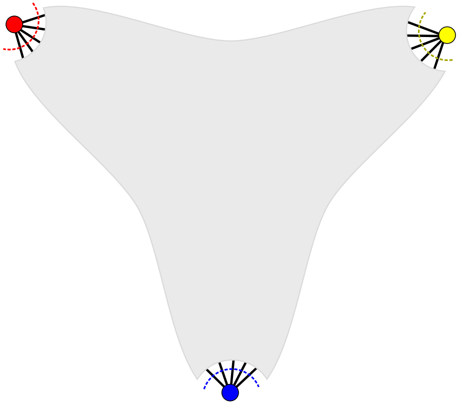
Second reduction rule

- **Second reduction rule.** If $tt' \in E(G)$ for some $t, t' \in T$, then remove tt' and decrease the budget by 1.
- Assume both the reduction rules are applied exhaustively.
- Then the sets $\Delta(\{t\})$ for $t \in T$ are:
  - pairwise disjoint;

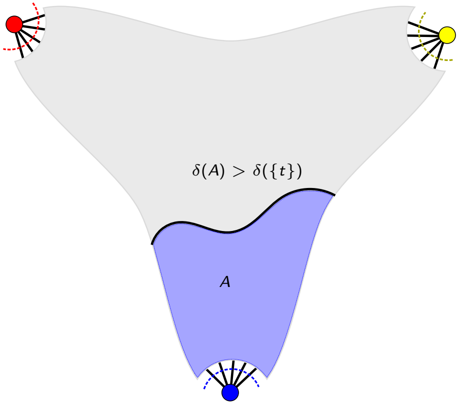
Second reduction rule

- **Second reduction rule.** If $tt' \in E(G)$ for some $t, t' \in T$, then remove tt' and decrease the budget by 1.
- Assume both the reduction rules are applied exhaustively.
- Then the sets $\Delta(\{t\})$ for $t \in T$ are:
  - pairwise disjoint;
  - the only minimum cuts from t to $T \setminus \{t\}$.

Structure of the instance

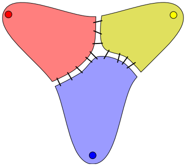


Structure of the instance



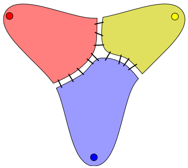
Digression: approximation

- Let F be a solution and $d_i = \delta(\mathtt{reach}(t_i, G \setminus F))$. Then $|F| = \frac{1}{2}\sum_i d_i$.



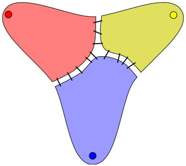
Digression: approximation

- Let F be a solution and $d_i = \delta(\mathtt{reach}(t_i, G \setminus F))$. Then $|F| = \frac{1}{2}\sum_i d_i$.
- Let C_i be the cutset of any min-cut between t_i and $T \setminus \{t_i\}$, and let $c_i = |C_i|$. Obviously $c_i \leq d_i$.



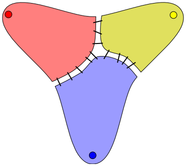
Digression: approximation

- Let F be a solution and $d_i = \delta(\texttt{reach}(t_i, G \setminus F))$. Then $|F| = \frac{1}{2}\sum_i d_i$.
- Let C_i be the cutset of any min-cut between t_i and $T \setminus \{t_i\}$, and let $c_i = |C_i|$. Obviously $c_i \leq d_i$.
- $F^\star := \bigcup_i C_i$ is always a solution (even if we omit one of them)!



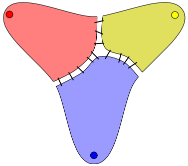
Digression: approximation

- Let F be a solution and $d_i = \delta(\texttt{reach}(t_i, G \setminus F))$. Then $|F| = \frac{1}{2}\sum_i d_i$.
- Let C_i be the cutset of any min-cut between t_i and $T \setminus \{t_i\}$, and let $c_i = |C_i|$. Obviously $c_i \leq d_i$.
- $F^\star := \bigcup_i C_i$ is always a solution (even if we omit one of them)!
- $|F^\star| \leq \sum_i c_i \leq \sum_i d_i \leq 2|F|$, hence $F^\star$ is a 2-approximation.



Digression: approximation

- Let F be a solution and $d_i = \delta(\texttt{reach}(t_i, G \setminus F))$. Then $|F| = \frac{1}{2}\sum_i d_i$.
- Let C_i be the cutset of any min-cut between t_i and $T \setminus \{t_i\}$, and let $c_i = |C_i|$. Obviously $c_i \leq d_i$.
- $F^\star := \bigcup_i C_i$ is always a solution (even if we omit one of them)!
- $|F^\star| \leq \sum_i c_i \leq \sum_i d_i \leq 2|F|$, hence $F^\star$ is a 2-approximation.
- If we omit the largest C_i, we get $(2 - \frac{2}{|T|})$-approximation.



The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2} \sum_i c_i$.

The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2}\sum_i c_i$.
- If we are dealing with a YES-instance, then

$$\lambda(G, T) \leq OPT \leq k.$$

The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2}\sum_i c_i$.
- If we are dealing with a YES-instance, then

$$\lambda(G, T) \leq OPT \leq k.$$

- If the instance is moreover non-trivial, then $2\lambda(G, T) > k$, so in particular $\frac{k}{2} < \lambda(G, T) \leq OPT \leq k$.

The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2} \sum_i c_i$.
- If we are dealing with a YES-instance, then

$$\lambda(G, T) \leq OPT \leq k.$$

- If the instance is moreover non-trivial, then $2\lambda(G, T) > k$, so in particular $\frac{k}{2} < \lambda(G, T) \leq OPT \leq k$.
- **Main idea**: A branching algorithm makes progress not only if the budget decreases, but **also** if $\lambda(G, T)$ increases!

The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2}\sum_i c_i$.
- If we are dealing with a YES-instance, then

$$\lambda(G, T) \leq OPT \leq k.$$

- If the instance is moreover non-trivial, then $2\lambda(G, T) > k$, so in particular $\frac{k}{2} < \lambda(G, T) \leq OPT \leq k$.
- **Main idea**: A branching algorithm makes progress not only if the budget decreases, but **also** if $\lambda(G, T)$ increases!
  - The budget can decrease at most k times.

The min-cuts as a lower bound

- Let $\lambda(G, T) = \frac{1}{2} \sum_i c_i$.
- If we are dealing with a YES-instance, then

$$\lambda(G, T) \leq OPT \leq k.$$

- If the instance is moreover non-trivial, then $2\lambda(G, T) > k$, so in particular $\frac{k}{2} < \lambda(G, T) \leq OPT \leq k$.
- **Main idea**: A branching algorithm makes progress not only if the budget decreases, but **also** if $\lambda(G, T)$ increases!
  - The budget can decrease at most k times.
  - The lower bound cannot increase to more than the budget.

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \le \phi(G, T, k) < \frac{k}{2}$

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \leq \phi(G, T, k) < \frac{k}{2}$
- **Step 1**. Apply both reduction rules exhaustively.

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \leq \phi(G, T, k) < \frac{k}{2}$
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any edge ut incident to a terminal t, and branch into two subcases:

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \leq \phi(G, T, k) < \frac{k}{2}$
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any edge ut incident to a terminal t, and branch into two subcases:
  (a) ut will be in the solution, so delete ut and decrement k by 1;

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \leq \phi(G, T, k) < \frac{k}{2}$
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any edge ut incident to a terminal t, and branch into two subcases:
  (a) ut will be in the solution, so delete ut and decrement k by 1;
  (b) ut will not be in the solution, so contract ut.

The algorithm

- Let $\phi(G, T, k) := k - \lambda(G, T) = k - \frac{1}{2}\sum_i c_i$. In a nontrivial YES-instance we have $0 \leq \phi(G, T, k) < \frac{k}{2}$
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any edge ut incident to a terminal t, and branch into two subcases:
  (a) ut will be in the solution, so delete ut and decrement k by 1;
  (b) ut will not be in the solution, so contract ut.
- **Step 3**. Proceed with Steps 1 and 2 up to the point when every terminal becomes isolated (YES), or $\phi(G, T, k)$ becomes negative (NO).

Deleting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume *ut* is deleted.

Deleting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is deleted.
  - k is decremented by 1.

Deleting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is deleted.
  - k is decremented by 1.
  - The min-cut from t decreases by 1.

Deleting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is deleted.
  - k is decremented by 1.
  - The min-cut from t decreases by 1.
  - The min-cut from any other terminal t' does not change, since any $(\{t'\}, T \setminus \{t'\})$ cut that includes ut has larger cutset than the cut $\{t'\}$.

Deleting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is deleted.
  - k is decremented by 1.
  - The min-cut from t decreases by 1.
  - The min-cut from any other terminal t' does not change, since any $(\{t'\}, T \setminus \{t'\})$ cut that includes ut has larger cutset than the cut $\{t'\}$.
- Hence, the potential decreases by exactly $1 - \frac{1}{2} = \frac{1}{2}$.

Contracting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume *ut* is contracted.

Contracting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is contracted.
  - k stays the same.

Contracting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume *ut* is contracted.
  - k stays the same.
  - The min-cut from t increases by at least 1.

Contracting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume *ut* is contracted.
  - k stays the same.
  - The min-cut from t increases by at least 1.
  - The min-cut from any other terminal t' stays the same.

Contracting the edge

$$\phi(G, T, k) := k - \frac{1}{2}\sum_i c_i$$

- Assume ut is contracted.
  - k stays the same.
  - The min-cut from t increases by at least 1.
  - The min-cut from any other terminal t' stays the same.
- Hence, the potential decreases by at least $\frac{1}{2}$.

Wrapping up

- Potential is less than $\frac{k}{2}$ in the beginning, and decreases by at least $\frac{1}{2}$ at each branch.

Wrapping up

- Potential is less than $\frac{k}{2}$ in the beginning, and decreases by at least $\frac{1}{2}$ at each branch.
- Hence we get an $\mathcal{O}^\star(2^k)$ algorithm.

Wrapping up

- Potential is less than $\frac{k}{2}$ in the beginning, and decreases by at least $\frac{1}{2}$ at each branch.
- Hence we get an $\mathcal{O}^\star(2^k)$ algorithm.
- **Crucial point**: A branching rule can lead to some progress, even if this progress is not visible in the budget.

Technical details

- **Problem 1**. Deleting an edge may disconnect the graph, and we described the reductions for connected graphs.

Technical details

- **Problem 1**. Deleting an edge may disconnect the graph, and we described the reductions for connected graphs.
  - Apply the rules to each connected component separately. Everything goes smoothly.

Technical details

- **Problem 1**. Deleting an edge may disconnect the graph, and we described the reductions for connected graphs.
  - Apply the rules to each connected component separately. Everything goes smoothly.
- **Problem 2**. We need to make sure that the potential does not increase during reduction rules.

Technical details

- **Problem 1**. Deleting an edge may disconnect the graph, and we described the reductions for connected graphs.
  - Apply the rules to each connected component separately. Everything goes smoothly.
- **Problem 2**. We need to make sure that the potential does not increase during reduction rules.
  - **Min-cut rule**: Contractions can only increase min-cuts, so the potential can only decrease.

Technical details

- **Problem 1**. Deleting an edge may disconnect the graph, and we described the reductions for connected graphs.
  - Apply the rules to each connected component separately. Everything goes smoothly.
- **Problem 2**. We need to make sure that the potential does not increase during reduction rules.
  - **Min-cut rule**: Contractions can only increase min-cuts, so the potential can only decrease.
  - **Second rule**: k decreases by 1, $\sum_i c_i$ decreases by 2 $\Rightarrow$ the potential stays the same.

Plan for now

- An $\mathcal{O}^\star(2^k)$ algorithm for Node Multiway Cut.

Plan for now

- An $\mathcal{O}^\star(2^k)$ algorithm for NODE MULTIWAY CUT.
  - Based on a joint work with Cygan and Wojtaszczyk.

The cuts

- **First idea**: Take vertex cuts instead of edge cuts!

The cuts

- **First idea**: Take vertex cuts instead of edge cuts!
- **Problem**: Sum of min-cuts is only a $|T|$-approximation of the optimum solution.

The cuts

- **First idea**: Take vertex cuts instead of edge cuts!
- **Problem**: Sum of min-cuts is only a $|T|$-approximation of the optimum solution.
  - A star with terminals on the petals.

The cuts

- **First idea**: Take vertex cuts instead of edge cuts!
- **Problem**: Sum of min-cuts is only a $|T|$-approximation of the optimum solution.
  - A star with terminals on the petals.
- We need a smarter lower bound.

Linear programming, recap

- A linear program consists of:

Linear programming, recap

- A linear program consists of:
  - a vector of variables $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$;

Linear programming, recap

- A linear program consists of:
  - a vector of variables $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$;
  - a set of linear constraints ($a_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}$) of the form

$$\sum_{i=1}^{n} a_{ij} x_i \leq b_j.$$

Linear programming, recap

- A linear program consists of:
  - a vector of variables $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$;
  - a set of linear constraints ($a_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}$) of the form

$$\sum_{i=1}^{n} a_{ij} x_i \leq b_j.$$

  - a goal vector $c \in \mathbb{R}^n$.

Linear programming, recap

- A linear program consists of:
  - a vector of variables $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$;
  - a set of linear constraints ($a_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}$) of the form

$$\sum_{i=1}^{n} a_{ij} x_i \leq b_j.$$

  - a goal vector $c \in \mathbb{R}^n$.
- The goal is to find a vector x that minimizes/maximizes $\sum_{i=1}^{n} c_i x_i$ while satisfying all the constraints.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.
  - We can have exponentially many constraints.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.
  - We can have exponentially many constraints.
  - But we need to provide a polynomial-time oracle that, for a given vector, either

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.
  - We can have exponentially many constraints.
  - But we need to provide a polynomial-time oracle that, for a given vector, either
    (a) concludes that all the constraints are satisfied; or

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.
  - We can have exponentially many constraints.
  - But we need to provide a polynomial-time oracle that, for a given vector, either
    (a) concludes that all the constraints are satisfied; or
    (b) provides a constraint that is broken.

Linear programming, recap

- If we additionally require that the variables must be integral, then we get *integer programming* which is NP-hard.
- However, linear programming can be solved in polynomial time using the *ellipsoid method*.
- This method works in a more general *separation oracle model*.
  - We can have exponentially many constraints.
  - But we need to provide a polynomial-time oracle that, for a given vector, either
    - (a) concludes that all the constraints are satisfied; or
    - (b) provides a constraint that is broken.
- **Usage**: Model a problem as an integer program, and relax the integer constraints to linear ones. The solution to the relaxation is a lower bound for the solution to the integer program.

Integer program for NMWC

- For every non-terminal u, we have a variable x_u.

Integer program for NMWC

- For every non-terminal u, we have a variable x_u.
- **Integer constraints**: $\forall_u \; x_u \in \{0, 1\}$, denoting whether the vertex is chosen or not.

Integer program for NMWC

- For every non-terminal u, we have a variable x_u.
- **Integer constraints**: $\forall_u \; x_u \in \{0, 1\}$, denoting whether the vertex is chosen or not.
- **Goal**: Minimize $\sum_u x_u$.

Integer program for NMWC

- For every non-terminal u, we have a variable x_u.
- **Integer constraints**: $\forall_u \ x_u \in \{0, 1\}$, denoting whether the vertex is chosen or not.
- **Goal**: Minimize $\sum_u x_u$.
- **Linear constraints**: for every path P between two different terminals, we have:

$$\sum_{u \in V(P)} x_u \geq 1.$$

Linear relaxation

- For every non-terminal u, we have a variable x_u.
- **Relaxed constraints**: $\forall_u \; 0 \leq x_u \leq 1$, denoting in what *fraction* the vertex is chosen to the solution.
- **Goal**: Minimize $\sum_u x_u$.
- **Linear constraints**: for every path P between two different terminals, we have:

$$\sum_{u \in V(P)} x_u \geq 1.$$

Linear relaxation

- For every non-terminal u, we have a variable x_u.
- **Relaxed constraints**: $\forall_u \; 0 \leq x_u \leq 1$, denoting in what *fraction* the vertex is chosen to the solution.
- **Goal**: Minimize $\sum_u x_u$.
- **Linear constraints**: for every path P between two different terminals, we have:

$$\sum_{u \in V(P)} x_u \geq 1.$$

- **Separation oracle**: Dijkstra with vertex weights.

Half-integrality

- **Garg et al.**: This LP-relaxation of NMWC is *half-integral*, i.e., there exists an optimum solution that assigns only values $0, \frac{1}{2}, 1$.

Half-integrality

- **Garg et al.**: This LP-relaxation of NMWC is *half-integral*, i.e., there exists an optimum solution that assigns only values $0, \frac{1}{2}, 1$.
  - The proof is not difficult, but uses primal-dual complementary slackness condition.

Half-integrality

- **Garg et al.**: This LP-relaxation of NMWC is *half-integral*, i.e., there exists an optimum solution that assigns only values $0, \frac{1}{2}, 1$.
  - The proof is not difficult, but uses primal-dual complementary slackness condition.
  - See the proof in Chapter 19 of Vazirani.

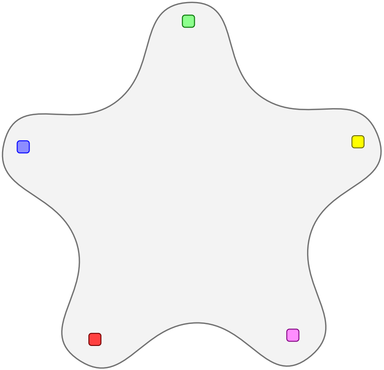
Half-integrality

- **Garg et al.**: This LP-relaxation of NMWC is *half-integral*, i.e., there exists an optimum solution that assigns only values $0, \frac{1}{2}, 1$.
  - The proof is not difficult, but uses primal-dual complementary slackness condition.
  - See the proof in Chapter 19 of Vazirani.
- **Note**: Self-reducibility $\Rightarrow$ We can find half-integral solution in polynomial time.

Half-integrality

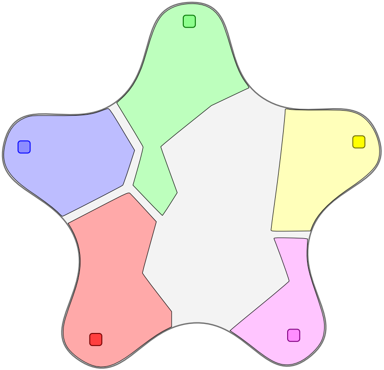
- **Garg et al.**: This LP-relaxation of NMWC is *half-integral*, i.e., there exists an optimum solution that assigns only values $0, \frac{1}{2}, 1$.
  - The proof is not difficult, but uses primal-dual complementary slackness condition.
  - See the proof in Chapter 19 of Vazirani.
- **Note**: Self-reducibility $\Rightarrow$ We can find half-integral solution in polynomial time.
- **Note**: Obviously $OPT_{\mathrm{LP}} \leq OPT$, but also $OPT \leq 2 \cdot OPT_{\mathrm{LP}}$, since we can round all the halves up to ones. So this rounding yields a 2-approximation.

Structure of the solution



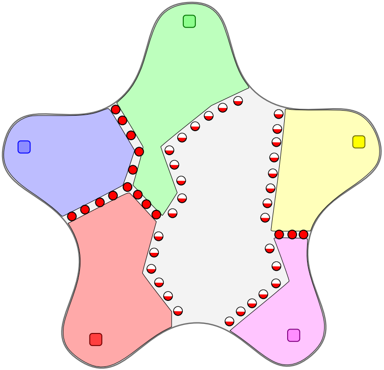
Take any half-integral optimum solution F.

Structure of the solution



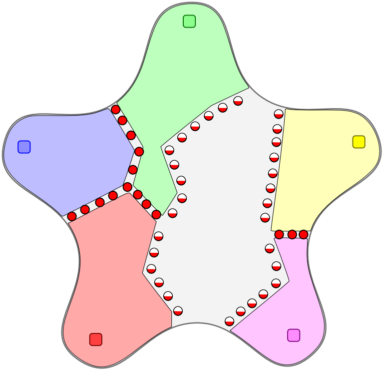
For $t \in T$, the *zero-region* U_t comprises vertices reachable from t using paths of weight 0.

Structure of the solution



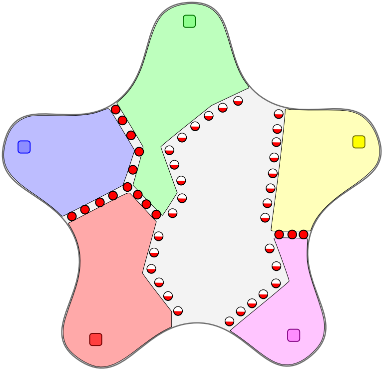
Define F' by putting 1-s on vertices that see ≥ 2 zero-regions, and $\frac{1}{2}$ on those seeing 1 region.

Structure of the solution



Observe: F' is still a solution,
and $F'(u) \leq F(u)$ for each u.

Structure of the solution



Conclusion: $F = F'$

Structure of the solution

Structure of the solution to the relaxation

Every optimum half-integral solution F has the following form:

- $F(u) = 1$ if u is in the neighbourhood of two or more zero-regions.
- $F(u) = \frac{1}{2}$ if u is in the neighbourhood of exactly one zero-regions.
- $F(u) = 0$ otherwise.

Structure of the solution

Structure of the solution to the relaxation

Every optimum half-integral solution F has the following form:

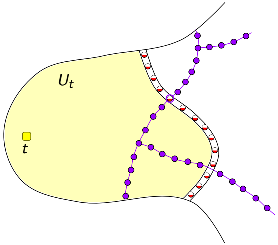
- $F(u) = 1$ if u is in the neighbourhood of two or more zero-regions.
- $F(u) = \frac{1}{2}$ if u is in the neighbourhood of exactly one zero-regions.
- $F(u) = 0$ otherwise.

Reduction of Guillemot

There is always an optimum solution of NMWC that does not touch any U_t. Hence, it is safe to contract every region U_t onto t.

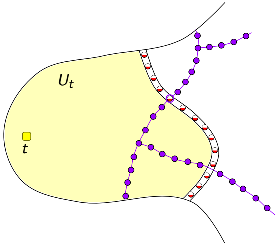
Proof of the reduction

- Assume for simplicity that the LP solution does not use any ones.



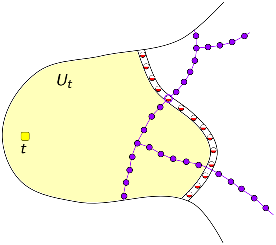
Proof of the reduction

- Assume for simplicity that the LP solution does not use any ones.
- Let X be an optimum solution to NMWC; let $B = X \cap \bigcup_{t \in T} U_t$.



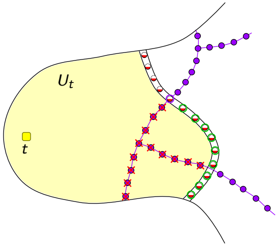
Proof of the reduction

- Assume for simplicity that the LP solution does not use any ones.
- Let X be an optimum solution to NMWC; let $B = X \cap \bigcup_{t \in T} U_t$.
- Let C be the set of those vertices of $N(\bigcup_{t \in T} U_t)$ that cannot be reached from respective t without passing through B.



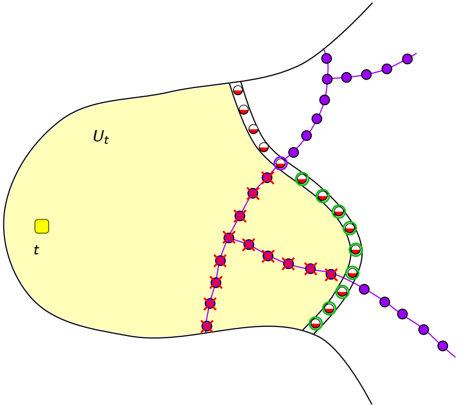
Proof of the reduction

- Assume for simplicity that the LP solution does not use any ones.
- Let X be an optimum solution to NMWC; let $B = X \cap \bigcup_{t \in T} U_t$.
- Let C be the set of those vertices of $N(\bigcup_{t \in T} U_t)$ that cannot be reached from respective t without passing through B.
- Replace B with C, i.e., consider $X' := (X \setminus B) \cup C$.



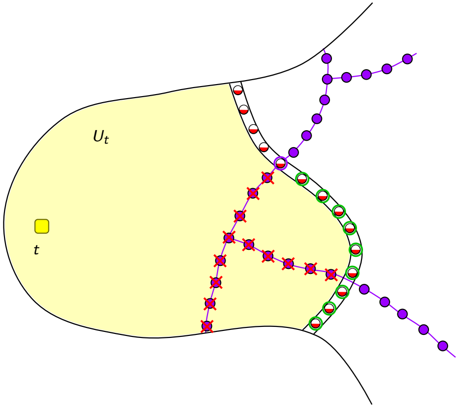
Verifying X'

- Again, we check two things.



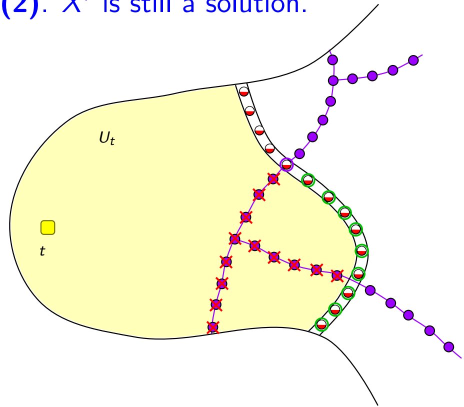
Verifying X'

- Again, we check two things.
  - **Check (1)**. $|X'| \leq |X|$, eq. $|C| \leq |B|$.



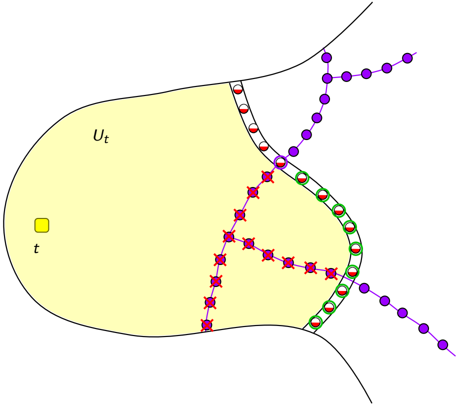
Verifying X'

- Again, we check two things.
  - **Check (1)**. $|X'| \leq |X|$, eq. $|C| \leq |B|$.
  - **Check (2)**. X' is still a solution.



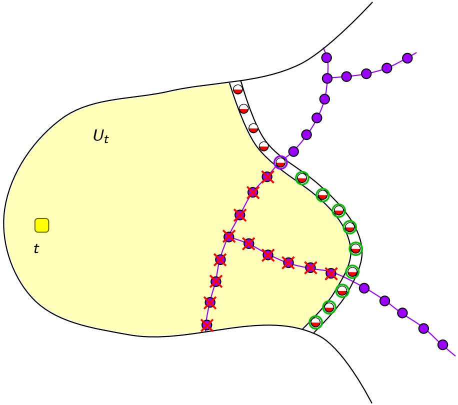
Check (2)

- The same argument as for the edge version.



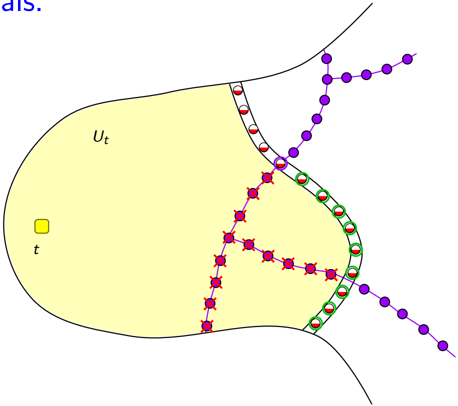
Check (2)

- The same argument as for the edge version.
- A t'-t'' path P untouched by X' must contain a vertex of B.



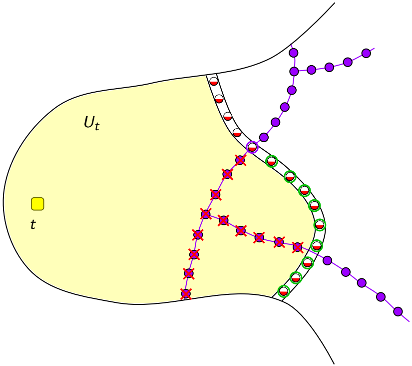
Check (2)

- The same argument as for the edge version.
- A t'-t'' path P untouched by X' must contain a vertex of B.
- Contradiction with $N(U_t \cup \texttt{reach}(t, G \setminus X))$ separating t from other terminals.



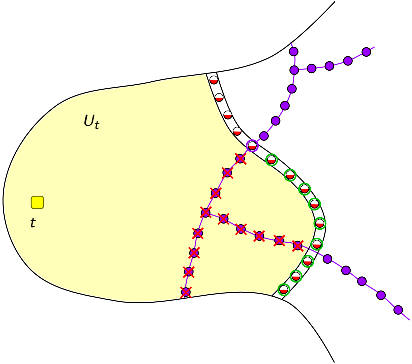
Check (1)

- Assume for contradiction that $|C| > |B|$.



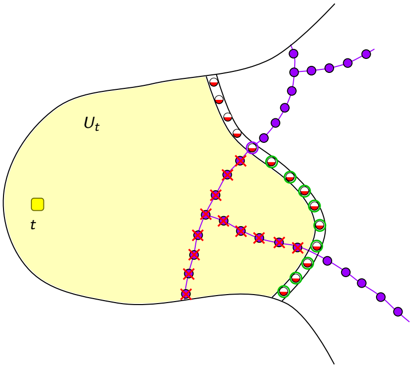
Check (1)

- Assume for contradiction that $|C| > |B|$.
- In F, do $-\varepsilon$ on C and $+\varepsilon$ on B.



Check (1)

- Assume for contradiction that $|C| > |B|$.
- In F, do $-\varepsilon$ on C and $+\varepsilon$ on B.
- The obtained F' is still a solution, and has strictly lower cost.



Second reduction

- Ones used by the LP can be seen as halves from two or more directions; everything goes through smoothly.

Second reduction

- Ones used by the LP can be seen as halves from two or more directions; everything goes through smoothly.
- After the application of regions U_t, vertices that saw two regions are vertices that see two terminals.

Second reduction

- Ones used by the LP can be seen as halves from two or more directions; everything goes through smoothly.
- After the application of regions U_t, vertices that saw two regions are vertices that see two terminals.
- Such non-terminals must be always chosen.

Second reduction

- Ones used by the LP can be seen as halves from two or more directions; everything goes through smoothly.
- After the application of regions U_t, vertices that saw two regions are vertices that see two terminals.
- Such non-terminals must be always chosen.
- **Corollary**: Safe to greedily choose vertices assigned 1 by the LP.

Second reduction

- Ones used by the LP can be seen as halves from two or more directions; everything goes through smoothly.
- After the application of regions U_t, vertices that saw two regions are vertices that see two terminals.
- Such non-terminals must be always chosen.
- **Corollary**: Safe to greedily choose vertices assigned 1 by the LP.
- After applying both rules exhaustively, we can assume that the **only** half-integral optimum solution to LP assigns $\frac{1}{2}$ on each $N(t)$, for $t \in T$.

The algorithm

- We do almost exactly the same.

The algorithm

- We do almost exactly the same.
- **Step 1**. Apply both reduction rules exhaustively.

The algorithm

- We do almost exactly the same.
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any vertex u adjacent to a terminal t, and branch into two subcases:

The algorithm

- We do almost exactly the same.
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any vertex u adjacent to a terminal t, and branch into two subcases:
  (a) u will be in the solution, so delete u and decrement k by 1;

The algorithm

- We do almost exactly the same.
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any vertex u adjacent to a terminal t, and branch into two subcases:
  (a) u will be in the solution, so delete u and decrement k by 1;
  (b) u will not be in the solution, so contract u onto t.

The algorithm

- We do almost exactly the same.
- **Step 1**. Apply both reduction rules exhaustively.
- **Step 2**. Pick any vertex u adjacent to a terminal t, and branch into two subcases:
  (a) u will be in the solution, so delete u and decrement k by 1;
  (b) u will not be in the solution, so contract u onto t.
- **Step 3**. Proceed with Steps 1 and 2 up to the point when every terminal becomes isolated (YES), or the LP solution exceeds the budget (NO).

Potential

- Potential: $\phi(G, T, k) = k - OPT_{\mathrm{LP}}$.

Potential

- Potential: $\phi(G, T, k) = k - OPT_{\mathrm{LP}}$.
- In a non-trivial YES instance we have

$$OPT_{\mathrm{LP}} \le k \le 2 \cdot OPT_{\mathrm{LP}},$$

so $0 \le \phi(G, T, k) \le k/2$.

Potential

- Potential: $\phi(G, T, k) = k - OPT_{\mathrm{LP}}$.
- In a non-trivial YES instance we have

$$OPT_{\mathrm{LP}} \leq k \leq 2 \cdot OPT_{\mathrm{LP}},$$

  so $0 \leq \phi(G, T, k) \leq k/2$.
- We analyse what happens with the potential in the branches.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into an LP-solution of I by putting 1 on u.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into an LP-solution of I by putting 1 on u.
  - Every half-integral LP-solution of I that puts 1 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into an LP-solution of I by putting 1 on u.
  - Every half-integral LP-solution of I that puts 1 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.
  - Hence every LP-solution of I' must have cost at least $OPT_{\mathrm{LP}}(I) - \frac{1}{2}$.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into an LP-solution of I by putting 1 on u.
  - Every half-integral LP-solution of I that puts 1 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.
  - Hence every LP-solution of I' must have cost at least $OPT_{\mathrm{LP}}(I) - \frac{1}{2}$.
  - But there is an LP-solution of I' of cost $OPT_{\mathrm{LP}}(I) - \frac{1}{2}$.

Deleting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k decreases by 1, and OPT_{LP} decreases by exactly $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into an LP-solution of I by putting 1 on u.
  - Every half-integral LP-solution of I that puts 1 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.
  - Hence every LP-solution of I' must have cost at least $OPT_{\mathrm{LP}}(I) - \frac{1}{2}$.
  - But there is an LP-solution of I' of cost $OPT_{\mathrm{LP}}(I) - \frac{1}{2}$.
- Hence, potential ϕ decreases by exactly $1 - \frac{1}{2} = \frac{1}{2}$.

Contracting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k stays the same, and OPT_{LP} increases by at least $\frac{1}{2}$.

Contracting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k stays the same, and OPT_{LP} increases by at least $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into a half-integral LP-solution of I by putting 0 on u.

Contracting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k stays the same, and OPT_{LP} increases by at least $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into a half-integral LP-solution of I by putting 0 on u.
  - But every half-integral solution of I that puts 0 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.

Contracting u

$$\phi(G, T, k) = k - OPT_{\mathrm{LP}}.$$

- Budget k stays the same, and OPT_{LP} increases by at least $\frac{1}{2}$.
  - Every half-integral LP-solution of I' can be transformed into a half-integral LP-solution of I by putting 0 on u.
  - But every half-integral solution of I that puts 0 on u has cost at least $OPT_{\mathrm{LP}}(I) + \frac{1}{2}$.
- Hence potential ϕ decreases by at least $\frac{1}{2}$.

Wrapping up

- The potential is $k/2$ at the beginning and decreases by $\frac{1}{2}$ at each step.

Wrapping up

- The potential is $k/2$ at the beginning and decreases by $\frac{1}{2}$ at each step.
- Again, reductions can only decrease the potential.

Wrapping up

- The potential is $k/2$ at the beginning and decreases by $\frac{1}{2}$ at each step.
- Again, reductions can only decrease the potential.
- Hence, we have an $\mathcal{O}^\star(2^k)$ algorithm.

Conclusions

- **Take-away message**: Use some polynomial-time computable lower bound to guide a branching algorithm.

Conclusions

- **Take-away message**: Use some polynomial-time computable lower bound to guide a branching algorithm.
- Progress is achieved not only by decreasing the budget, but also by increasing the lower bound.

Conclusions

- **Take-away message**: Use some polynomial-time computable lower bound to guide a branching algorithm.
- Progress is achieved not only by decreasing the budget, but also by increasing the lower bound.
- Solution to an LP relaxation may be a more robust lower bound than a combinatorial object.

Conclusions

- At the end of the day, we get a standard branching algorithm, just with an exotic progress measure.

Conclusions

- At the end of the day, we get a standard branching algorithm, just with an exotic progress measure.
  - Hence, we can open the toolbox of optimizing branching vectors via case analysis, and try to reduce the running time.

Conclusions

- At the end of the day, we get a standard branching algorithm, just with an exotic progress measure.
  - Hence, we can open the toolbox of optimizing branching vectors via case analysis, and try to reduce the running time.
  - **Cao, Chen, Fan**: $\mathcal{O}^\star(1.84^k)$ for Edge-MWC.

Conclusions

- At the end of the day, we get a standard branching algorithm, just with an exotic progress measure.
  - Hence, we can open the toolbox of optimizing branching vectors via case analysis, and try to reduce the running time.
  - **Cao, Chen, Fan**: $\mathcal{O}^\star(1.84^k)$ for EDGE-MWC.
  - Nothing better than $\mathcal{O}^\star(2^k)$ is known for NODE-MWC.

Conclusions

- At the end of the day, we get a standard branching algorithm, just with an exotic progress measure.
  - Hence, we can open the toolbox of optimizing branching vectors via case analysis, and try to reduce the running time.
  - **Cao, Chen, Fan**: $\mathcal{O}^\star(1.84^k)$ for Edge-MWC.
  - Nothing better than $\mathcal{O}^\star(2^k)$ is known for Node-MWC.
- **Tomorrow**: Applications of the same concept to Vertex Cover above Maximum Matching and other problems.