

A General Approximation Technique For Constrained Forest Problems

M. X. Goemans and D. P. Williamson
Presentation by Boris Kapchits

May 11, 2006

The Algorithm

Algorithm Properties

The (IP) problem

function f
characteristics

The way to the
solution: Dual Problem

The dual problem

The algorithm: informal
description

The Algorithm: formal
description

feasibility proof for the
dual solution

Example: shortest
path.

(IP) solution feasibility

The approximation
factor

Implementation

Additions

The Algorithm

- Fits for many graph problem whose solution is a forest (cover like problems). For example: shortest path(exact), MST(exact), Steiner tree(approx.)...

- Fits for many graph problem whose solution is a forest (cover like problems). For example: shortest path(exact), MST(exact), Steiner tree(approx.)...
- Gives a $(2 - \Delta)$ approximation for any of these problems (The value of Δ will be detailed further).

- Fits for many graph problem whose solution is a forest (cover like problems). For example: shortest path(exact), MST(exact), Steiner tree(approx.)...
- Gives a $(2 - \Delta)$ approximation for any of these problems (The value of Δ will be detailed further).
- The algorithm time complexity is $n^2 \log n$.

$$\begin{aligned} & \text{minimize} && \sum_{e \in E} c_e x_e \\ & \text{s.t.} && x(\delta(S)) \geq f(S) \\ & && x_e \in \{0, 1\} \end{aligned}$$

Where:

- $f(S)$ defined as $f : 2^{|V|} \rightarrow \{0, 1\}$.
- $\delta(S)$ denotes the set of edges, having exactly one endpoint in S , namely the edges that “cover” S .
- $x(F) = \sum_{e \in F} x_e$

We demand that each active set of vertices will be “covered” by an edge.

function f characteristics

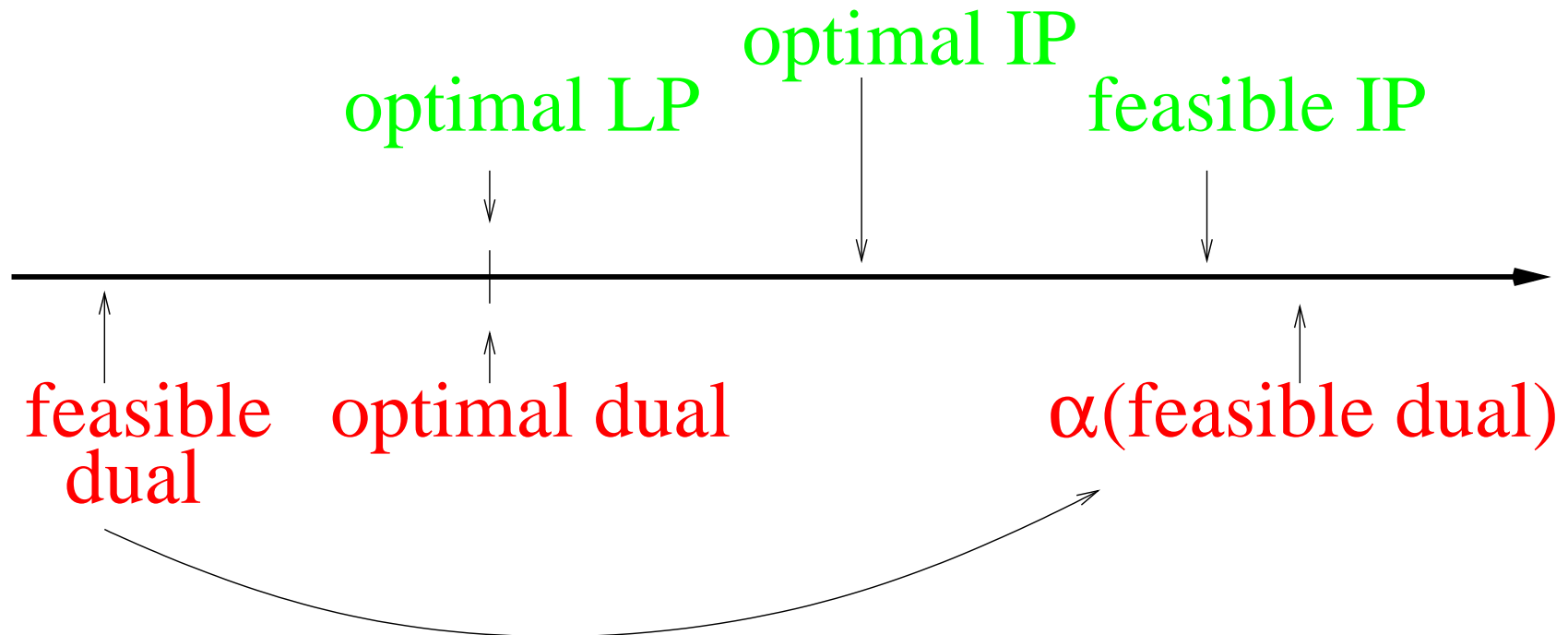
This function actually defines the problem, telling which sets of vertexes should be covered by an edge. Possible functions are, for example:

Input	$f(S)$	Minimal forest
	$f(S) = 1 \forall S$	Spanning trees
$s, t \in V$	$f(S) = \begin{cases} 1 & S \cap \{s, t\} = 1 \\ 0 & \text{otherwise} \end{cases}$	$s - t$ paths
$T \subseteq V$	$f(S) = \begin{cases} 1 & \emptyset \neq S \cap T \neq T \\ 0 & \text{otherwise} \end{cases}$	Steiner trees with terminals T

We deal only with *proper* functions, namely, the following properties should hold:

- *Symmetry*: $f(S) = f(V - S)$
- *Disjointness*: If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$
- $f(V) = 0$

The way to the solution: Dual Problem



We will construct in parallel feasible solutions for the dual and the IP problems, and show that the IP solution is at most $(2 - \frac{2}{|A|})$ times greater than the dual solution, and, hence, (IP) solution is a $(2 - \frac{2}{|A|})$ approximation.

■ $A = \{v \in V : f(\{v\}) = 1\}$

The dual problem

The relaxed primal problem

$$\begin{aligned} & \text{minimize} && \sum_{e \in E} c_e x_e \\ & \text{s.t.} && \\ & && x(\delta(S)) \geq f(S) \\ & && x_e \geq 0 \end{aligned}$$

The dual problem

$$\begin{aligned} & \text{maximize} && \sum_{S \subset V} f(S) \cdot y_S \\ & \text{s.t.} && \\ & && \sum_{S: e \in \delta(S)} y_S \leq c_e \\ & && y_S \geq 0 \end{aligned}$$

In the dual problem every set of vertices S is assigned a variable y_S .

The algorithm: informal description

$$\begin{aligned} & \text{maximize} && \sum_{S \subset V} f(S) \cdot y_S \\ & \text{s.t.} && \sum_{S: e \in \delta(S)} y_S \leq c_e \\ & && y_S \geq 0 \end{aligned}$$

Keeping in mind the dual problem we can think of the algorithm as of greedy construction of a feasible solution to the dual problem.

- At the initialization each vertex becomes a separate segment
- At each iteration we find the minimal value that can be added to y_S of each segment S . This increase makes a constraint tight for some edge e . This e is remembered as a candidate for the (IP) solution. The two components connected by e are merged.
- When no active components remain, we check which of the candidate edges are sufficient and output them as a solution.

The Algorithm: formal description

- 1: $F \leftarrow \emptyset$ /* the set of candidate edges. */
- 3: $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- 4: *for every* $v \in V$
- 5: $d(v) \leftarrow 0$
- 6: *while* $\exists C \in \mathcal{C} : f(C) = 1$
- 7: Find edge $e = (i, j)$ with $i \in C_p$ and $j \in C_q$, $C_p \neq C_q$ that minimizes
$$\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$$
- 8: $F \leftarrow F \cup \{e\}$
- 9: *for every* $C_r \in \mathcal{C}$, $f(C_r) = 1$
- 10: *for every* $v \in C_r$
- 11: $d(v) \leftarrow d(v) + \varepsilon$
- 13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
- 14: $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } (V, F - \{e\}), f(N) = 1\}$

The Algorithm: formal description

- 1: $F \leftarrow \emptyset$ /* the set of candidate edges. */
- 3: $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- 4: *for every* $v \in V$
- 5: $d(v) \leftarrow 0$
- 6: *while* $\exists C \in \mathcal{C} : f(C) = 1$
- 7: Find edge $e = (i, j)$ with $i \in C_p$ and $j \in C_q$, $C_p \neq C_q$ that minimizes
$$\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$$
- 8: $F \leftarrow F \cup \{e\}$
- 9: *for every* $C_r \in \mathcal{C}$, $f(C_r) = 1$
- 10: *for every* $v \in C_r$
- 11: $d(v) \leftarrow d(v) + \varepsilon$
- 13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
- 14: $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } (V, F - \{e\}), f(N) = 1\}$

The Algorithm: formal description

- 1: $F \leftarrow \emptyset$ /* the set of candidate edges. */
- 3: $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- 4: *for every* $v \in V$
- 5: $d(v) \leftarrow 0$
- 6: *while* $\exists C \in \mathcal{C} : f(C) = 1$
- 7: Find edge $e = (i, j)$ with $i \in C_p$ and $j \in C_q$, $C_p \neq C_q$ that minimizes
$$\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$$
- 8: $F \leftarrow F \cup \{e\}$
- 9: *for every* $C_r \in \mathcal{C}$, $f(C_r) = 1$
- 10: *for every* $v \in C_r$
- 11: $d(v) \leftarrow d(v) + \varepsilon$
- 13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
- 14: $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } (V, F - \{e\}), f(N) = 1\}$

The Algorithm: formal description

- 1: $F \leftarrow \emptyset$ /* the set of candidate edges. */
- 3: $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- 4: *for every* $v \in V$
- 5: $d(v) \leftarrow 0$
- 6: *while* $\exists C \in \mathcal{C} : f(C) = 1$
- 7: Find edge $e = (i, j)$ with $i \in C_p$ and $j \in C_q$, $C_p \neq C_q$ that minimizes
$$\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$$
- 8: $F \leftarrow F \cup \{e\}$
- 9: *for every* $C_r \in \mathcal{C}$, $f(C_r) = 1$
- 10: *for every* $v \in C_r$
- 11: $d(v) \leftarrow d(v) + \varepsilon$
- 13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
- 14: $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } (V, F - \{e\}), f(N) = 1\}$

The Algorithm: formal description

- 1: $F \leftarrow \emptyset$ /* the set of candidate edges. */
- 2: $LB \leftarrow 0$ /* The value of dual solution */
- 3: $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- 4: *for every* $v \in V$
- 5: $d(v) \leftarrow 0$
- 6: *while* $\exists C \in \mathcal{C} : f(C) = 1$
- 7: Find edge $e = (i, j)$ with $i \in C_p$ and $j \in C_q$, $C_p \neq C_q$ that minimizes
$$\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$$
- 8: $F \leftarrow F \cup \{e\}$
- 9: *for every* $C_r \in \mathcal{C}$, $f(C_r) = 1$
- 10: *for every* $v \in C_r$
- 11: $d(v) \leftarrow d(v) + \varepsilon$
- 12: $LB \leftarrow LB + \varepsilon \sum_{C \in \mathcal{C}} f(C)$
- 13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
- 14: $F' \leftarrow \{e \in F : \text{For some connected component } N \text{ of } (V, F - \{e\}), f(N) = 1\}$

feasibility proof for the dual solution

Claim: $d(v) = \sum_{v \in S} y_S$

Proof is by induction.

Claim: $\sum_{S:e \in \delta(S)} y_S \leq c_e$, in other words, the dual solution is feasible.

Proof is again, by induction.

Base: initially $y_S = 0$ for every S , therefore the claim holds.

Induction step: From the previous claim we know that for an edge e connecting pair of vertexes i and j from different segments $C_p, C_q \in \mathcal{C}$ in the current algorithm iteration, $\sum_{S:e \in \delta(S)} y_S = d(i) + d(j)$. Therefore, in a given iteration we can increase y_{C_p} and y_{C_q} by ε without violating the packing constrain for e if the following holds:

$$d(i) + d(j) + \varepsilon \cdot f(C_p) + \varepsilon \cdot f(C_q) \leq c_e$$

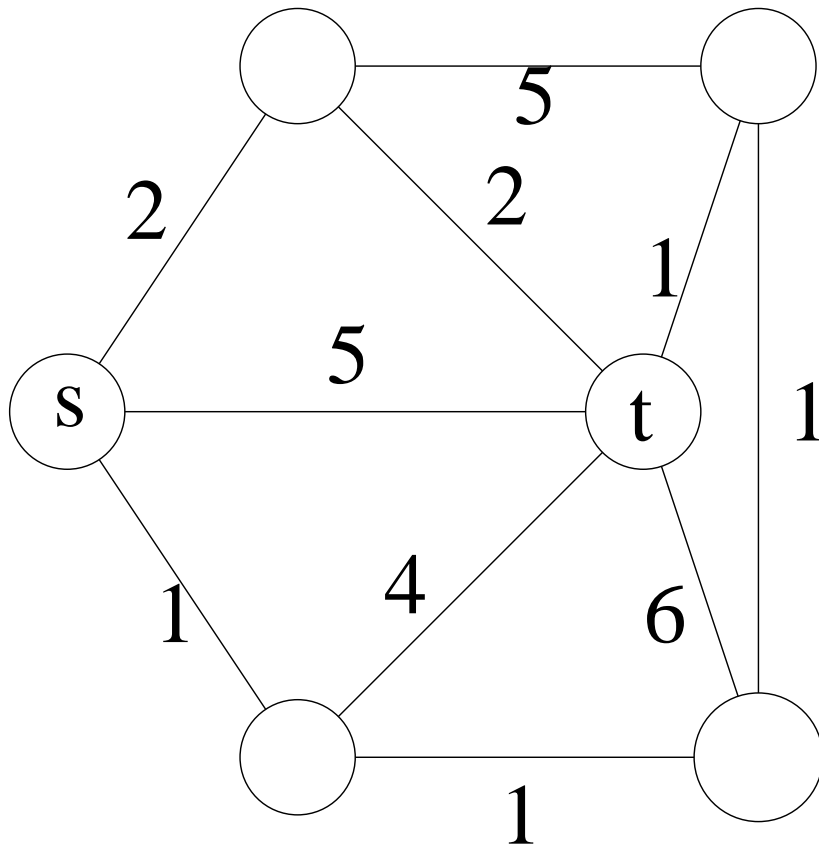
Therefore, if we pick the ε such that the packing constrain is not violated for any edge, the y_C of every active segment C can be increased by this value.

- Note, that if i and j are in the same segment of \mathcal{C} the y_C of this segment does not appear in the summation, and, therefore, the constrain can not be violated.

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

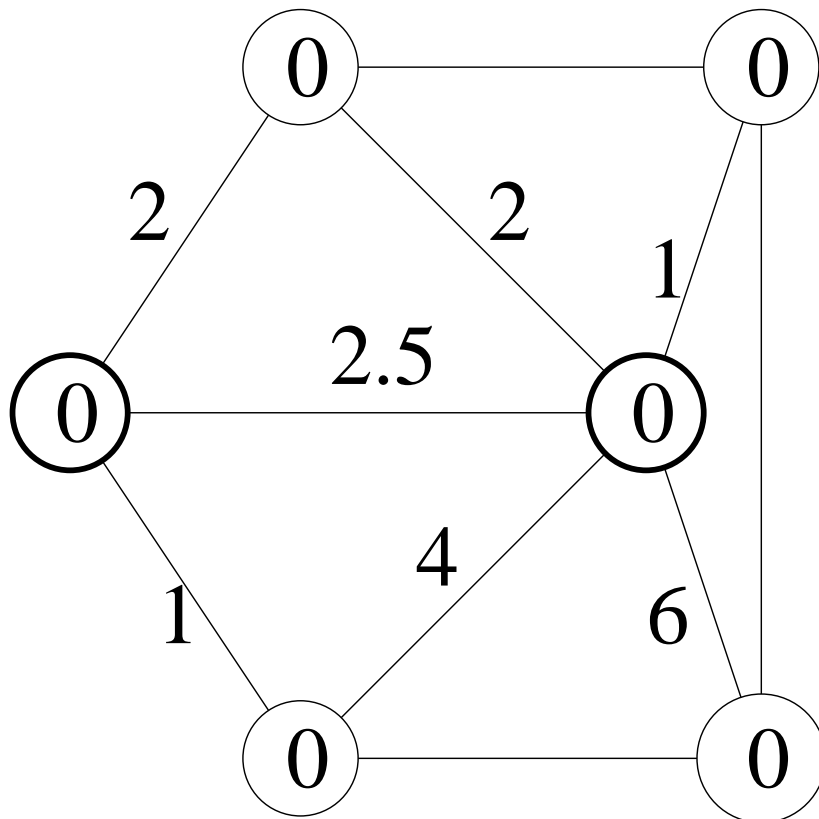


- The input graph. Numbers indicate the edge cost.

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

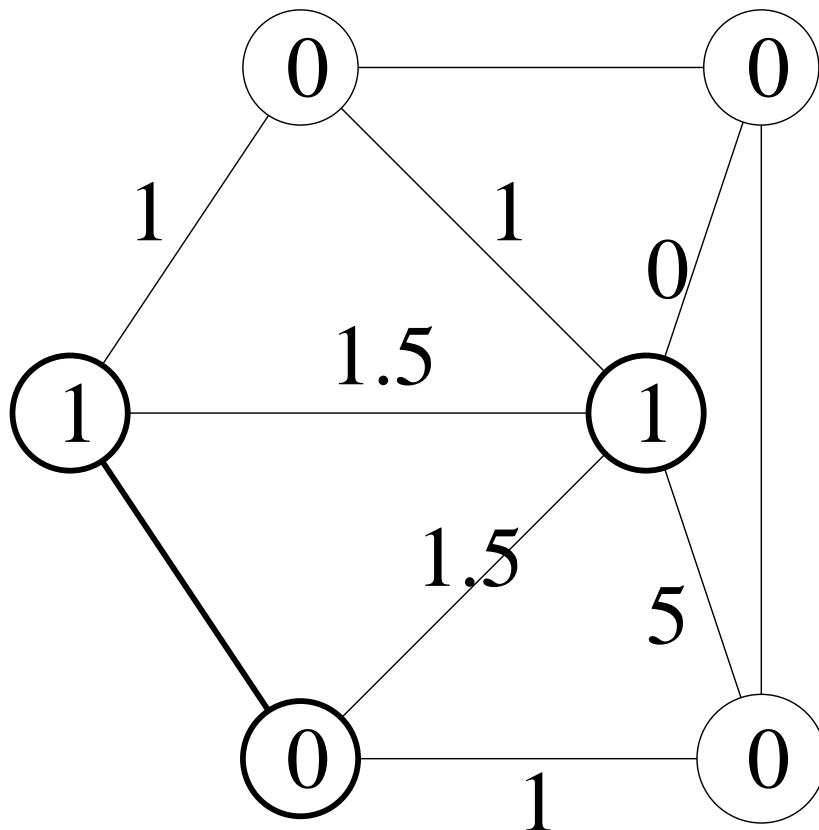


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost $\left(\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)} \right)$

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

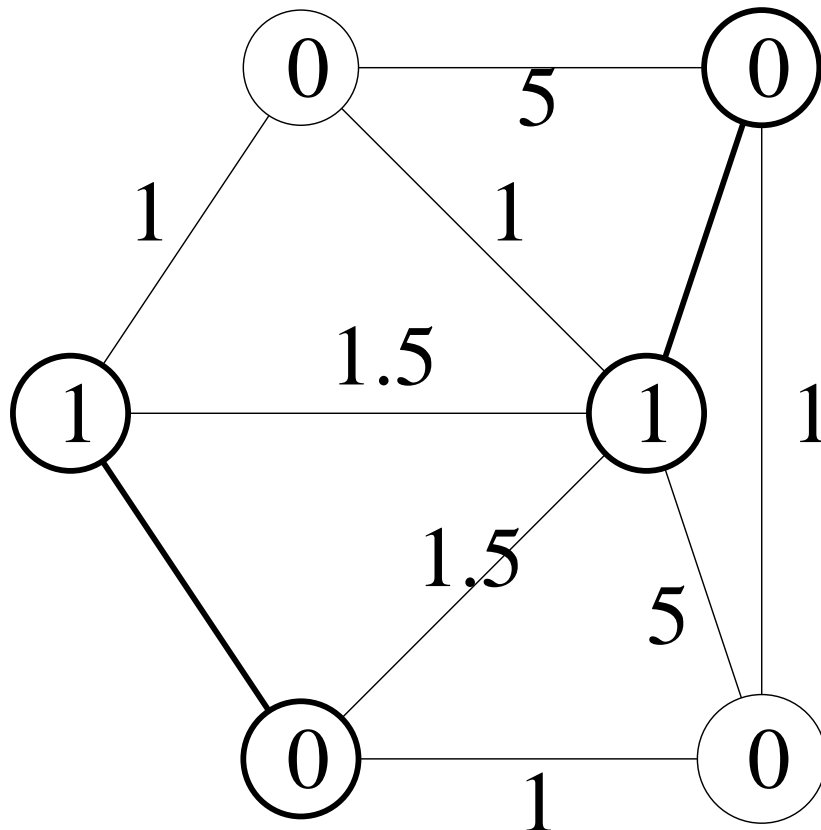


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost $\left(\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)} \right)$

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

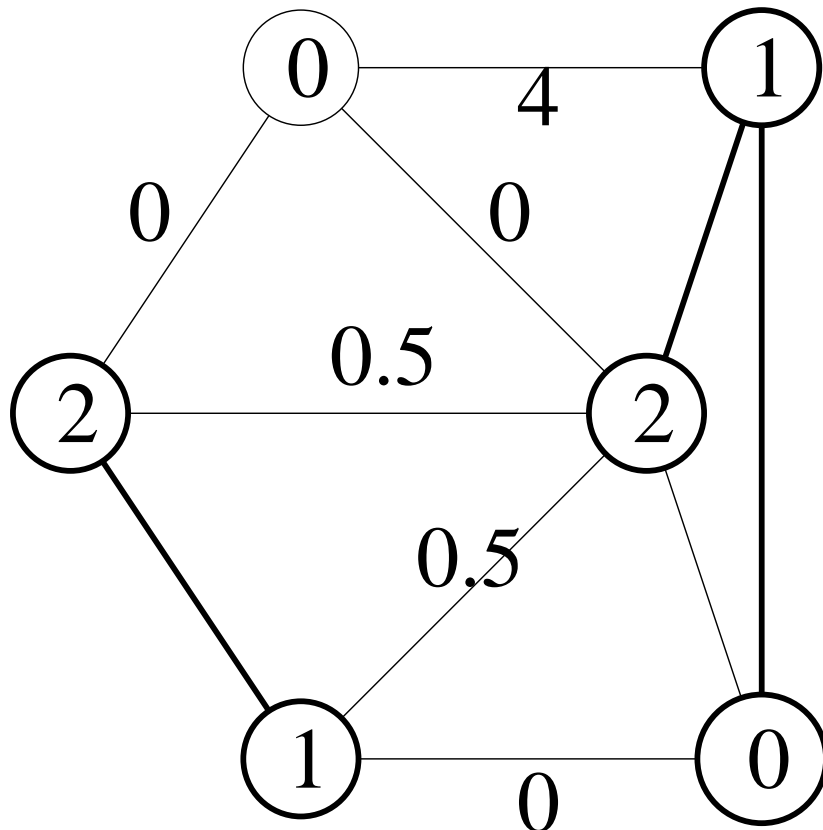


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost ($\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$)

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

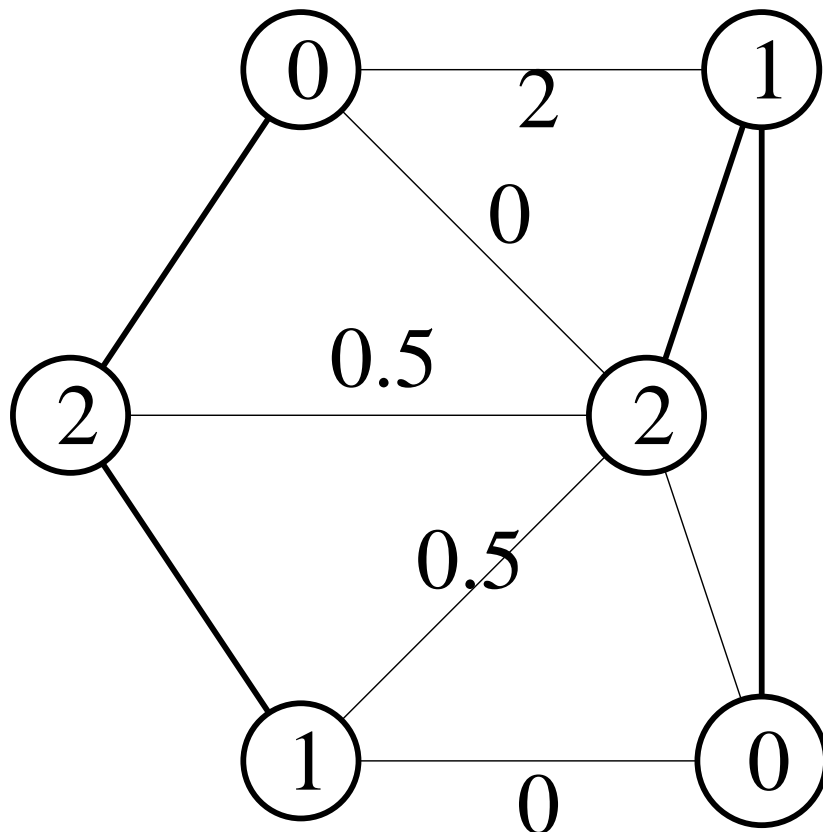


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost $\left(\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)} \right)$

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

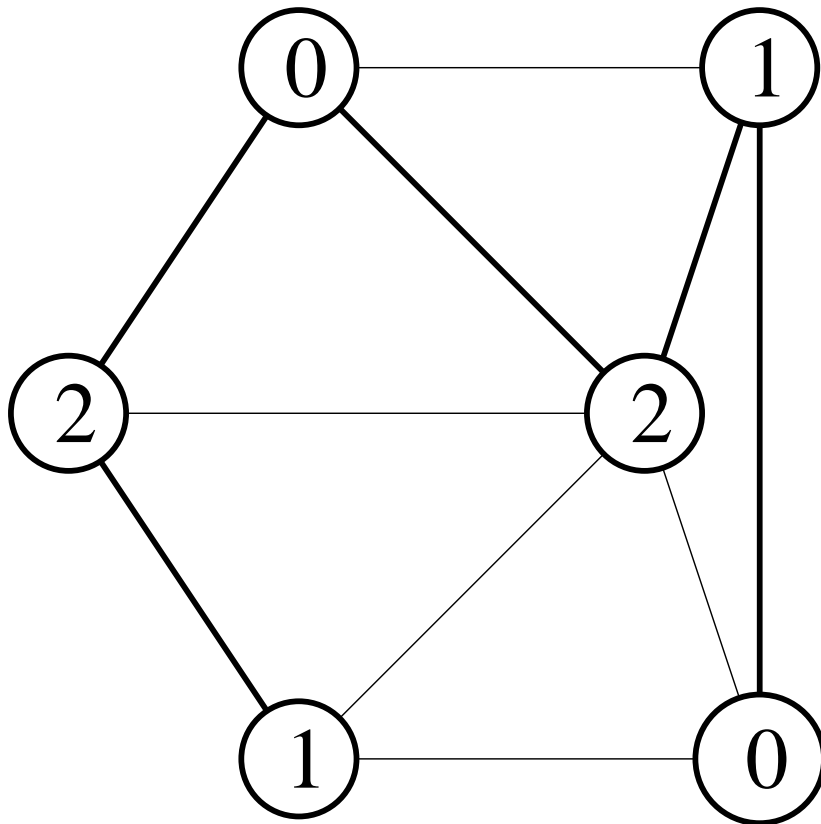


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost ($\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$)

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$

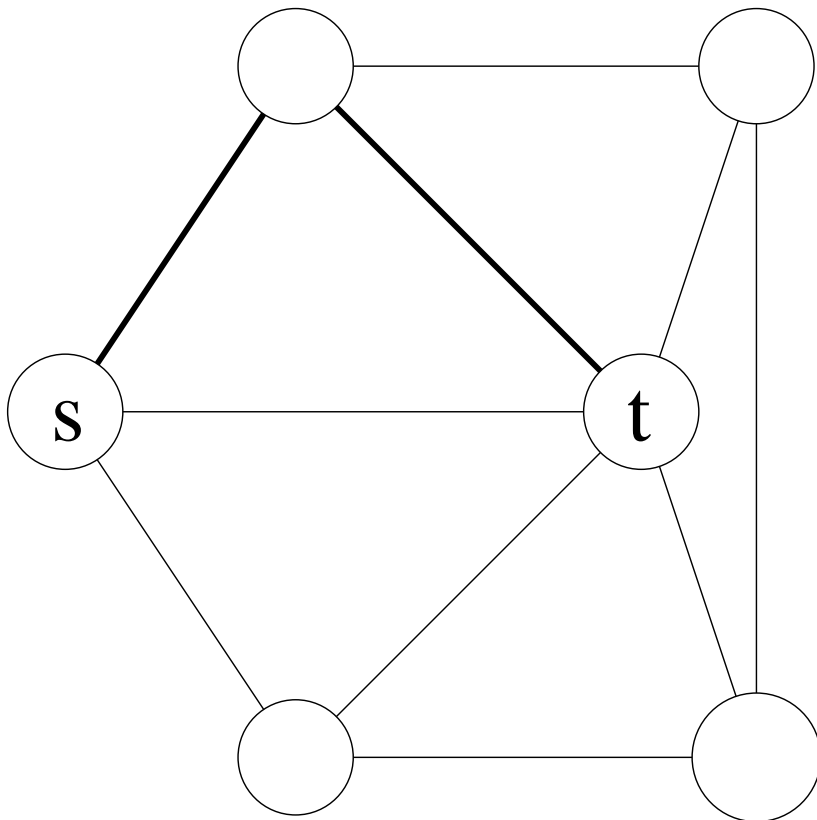


- Thick lines represent active components ($f(C) = 1$). Numbers on the nodes represent $d(v)$. Numbers on the edges represent the reduced cost $\left(\varepsilon_{(i,j)} = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)} \right)$

Example: shortest path.

Proper function is defined as follows:

$$f(S) = \begin{cases} 1 & |S \cap \{s, t\}| = 1 \\ 0 & \text{otherwise} \end{cases}$$



- Insufficient edges are removed, only the shortest path left.

The Algorithm

(IP) solution feasibility

IP solution feasibility:

Observation 2.1

IP solution feasibility:

Lemma 2.2

IP solution feasibility:

Lemma 2.2, cont.

IP solution feasibility:

Theorem 2.3

The approximation
factor

Implementation

Additions

(IP) solution feasibility

IP solution feasibility: Observation 2.1

Observation 2.1: If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.

It follows from the properties of f .

- By the symmetry property, $f(V - S) = f(S) = 0$.

IP solution feasibility: Observation 2.1

Observation 2.1: If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.

If follows from the properties of f .

- By the symmetry property, $f(V - S) = f(S) = 0$.
- Since $B \subseteq S$, B and $(V - S)$ are disjoint. By the disjointness property, $f((V - S) \cup B) = 0$.

IP solution feasibility: Observation 2.1

Observation 2.1: If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.

If follows from the properties of f .

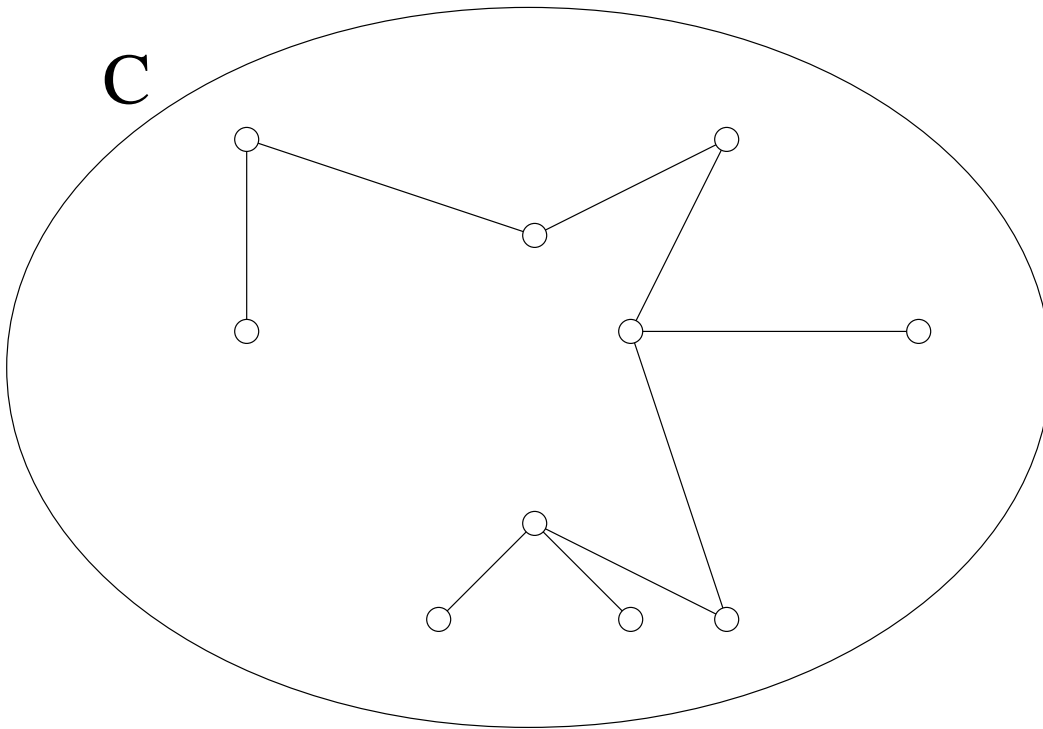
- By the symmetry property, $f(V - S) = f(S) = 0$.
- Since $B \subseteq S$, B and $(V - S)$ are disjoint. By the disjointness property, $f((V - S) \cup B) = 0$.
- $V - ((V - S) \cup B) = S - B$, so, by the symmetry property, $f(S - B) = f((V - S) \cup B) = 0$

This observation will be used further.

IP solution feasibility: Lemma 2.2

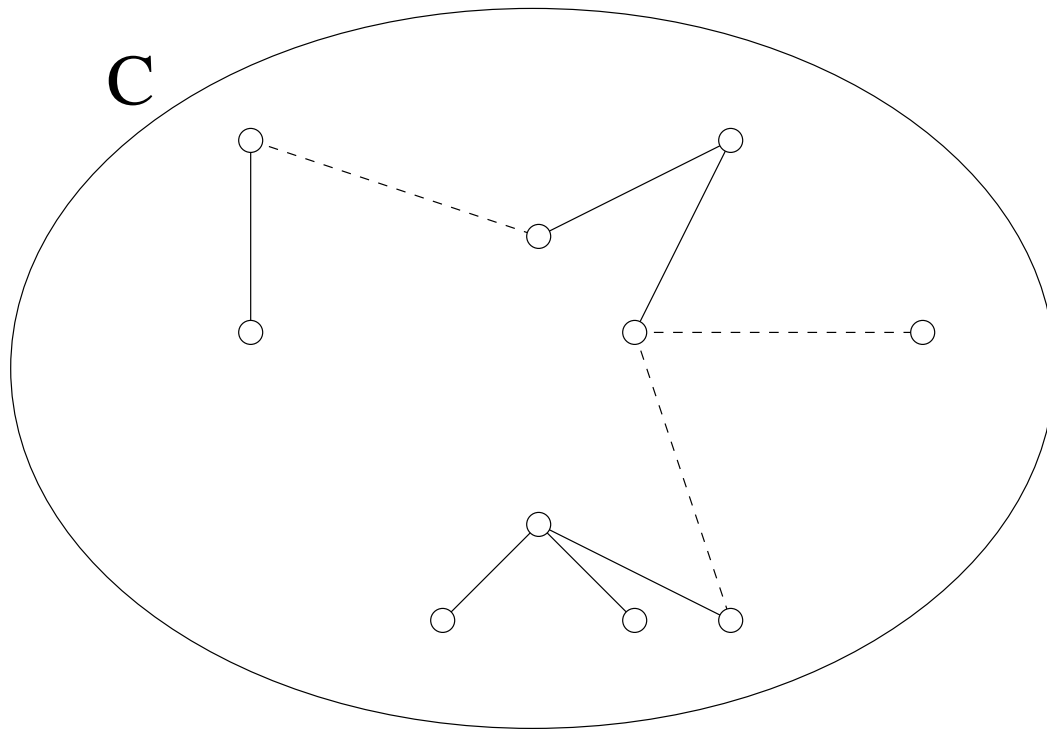
Lemma 2.2: For each connected component N of F' , $f(N) = 0$.

- By the construction of F' , $N \subseteq C$ for some segment C of F .



IP solution feasibility: Lemma 2.2

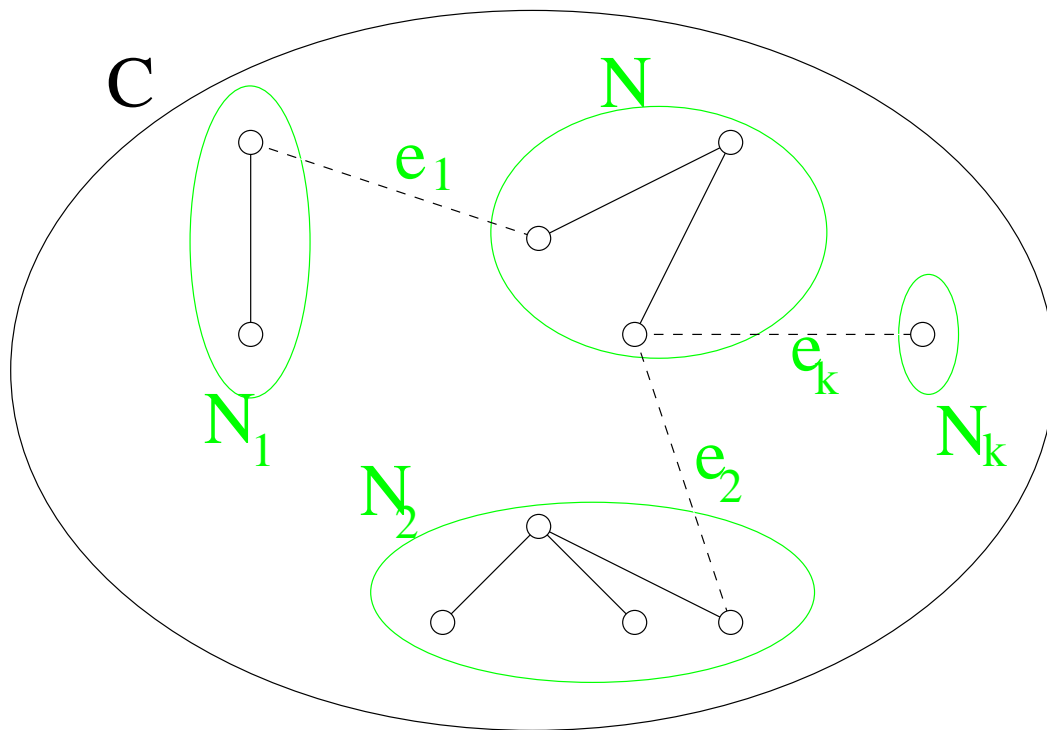
Lemma 2.2: For each connected component N of F' , $f(N) = 0$.



- By the construction of F' , $N \subseteq C$ for some segment C of F .
- Some edges of F were removed as insufficient, and the segment C was partitioned into a number of sub-segments

IP solution feasibility: Lemma 2.2

Lemma 2.2: For each connected component N of F' , $f(N) = 0$.

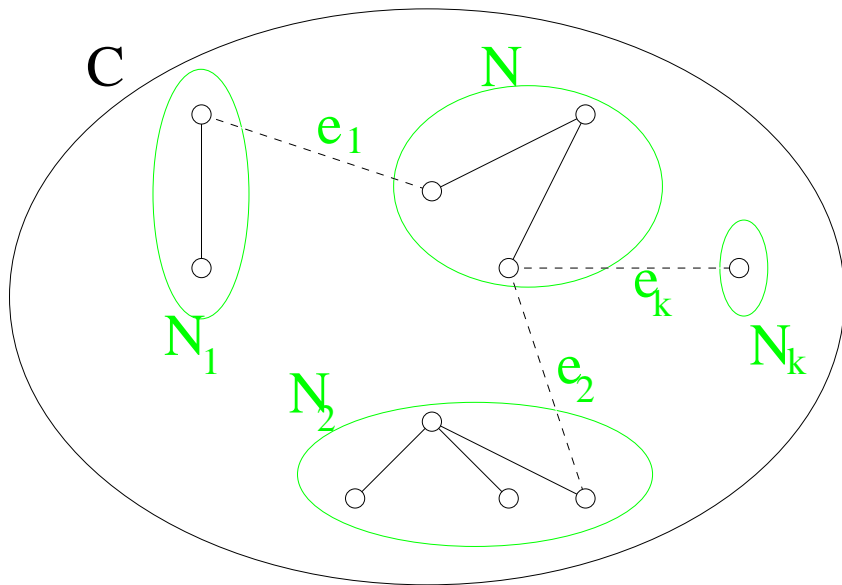


- By the construction of F' , $N \subseteq C$ for some segment C of F .
- Some edges of F were removed as insufficient, and the segment C was partitioned into a number of sub-segments.
- Let us denote the removed edges that are in $\delta(N)$ as e_1, \dots, e_k and the sub-segments different from N , created by removing these edges as N_1, \dots, N_k . Note that k can be also 0.

IP solution feasibility: Lemma 2.2, cont.

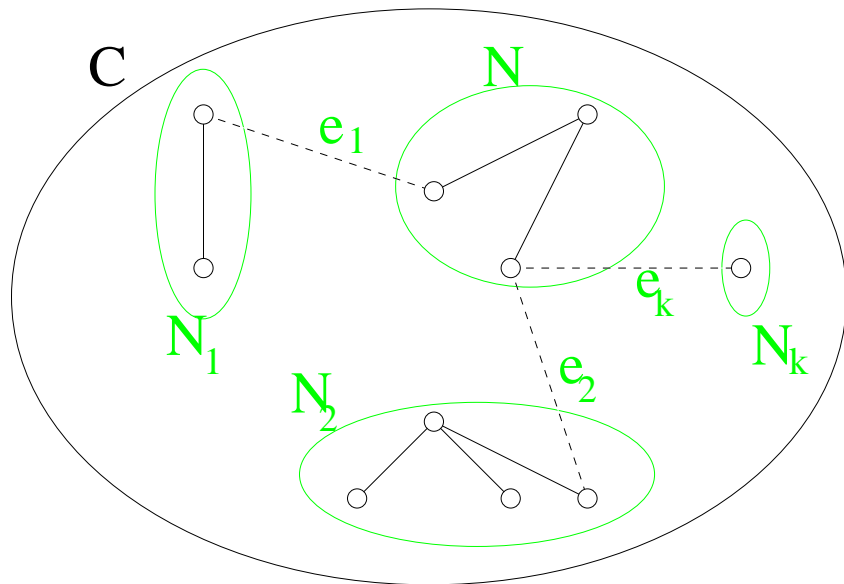
Lemma 2.2: For each connected component N of F' , $f(N) = 0$.

- Since $e_i \notin F'$, we know that $f(N_i) = 0$



IP solution feasibility: Lemma 2.2, cont.

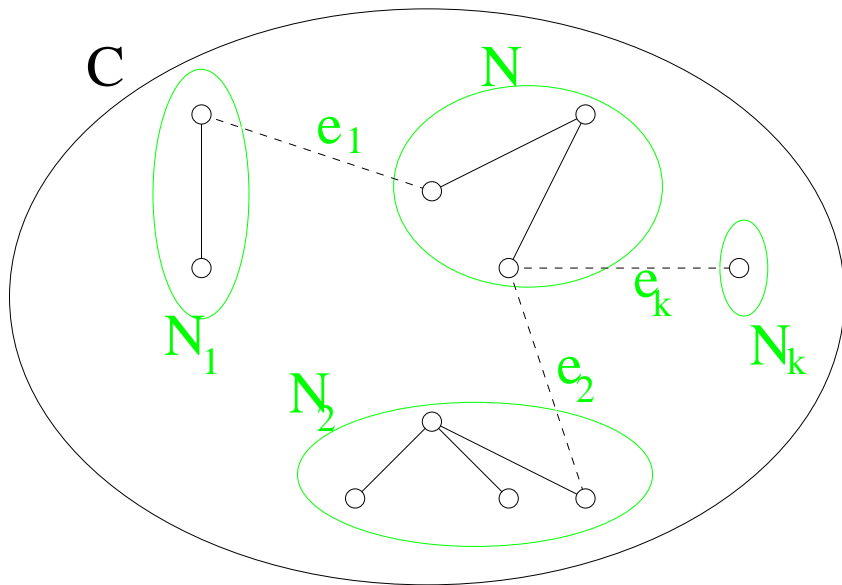
Lemma 2.2: For each connected component N of F' , $f(N) = 0$.



- Since $e_i \notin F'$, we know that $f(N_i) = 0$
- $C - N = \bigcup_{i=1}^k N_i$. By the disjointness property $f(\bigcup_{i=1}^k N_i) = 0$, hence $f(C - N) = 0$

IP solution feasibility: Lemma 2.2, cont.

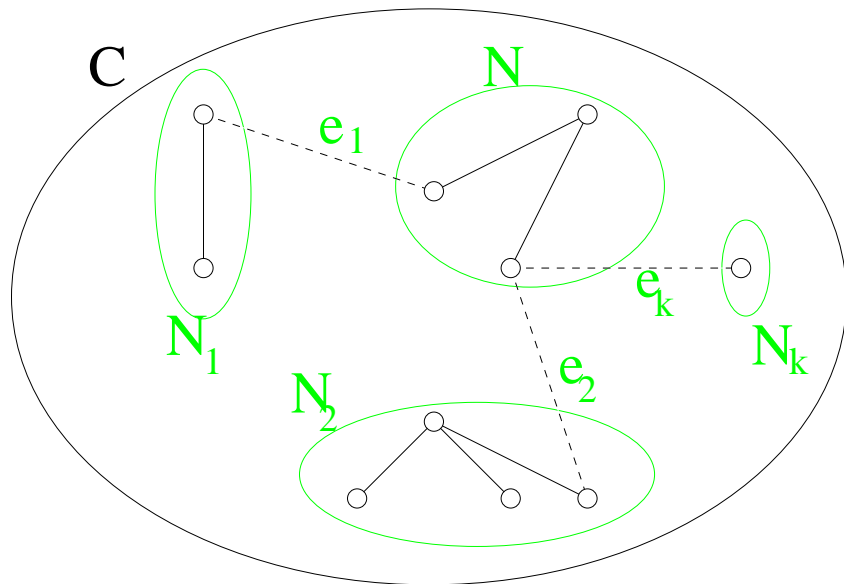
Lemma 2.2: For each connected component N of F' , $f(N) = 0$.



- Since $e_i \notin F'$, we know that $f(N_i) = 0$
- $C - N = \bigcup_{i=1}^k N_i$. By the disjointness property $f(\bigcup_{i=1}^k N_i) = 0$, hence $f(C - N) = 0$
- By the construction of F , $f(C) = 0$.

IP solution feasibility: Lemma 2.2, cont.

Lemma 2.2: For each connected component N of F' , $f(N) = 0$.



- Since $e_i \notin F'$, we know that $f(N_i) = 0$
- $C - N = \bigcup_{i=1}^k N_i$. By the disjointness property $f(\bigcup_{i=1}^k N_i) = 0$, hence $f(C - N) = 0$
- By the construction of F , $f(C) = 0$.
- Therefore, from observation 2.1 $f(N) = 0$.

IP solution feasibility: Theorem 2.3

Theorem 2.3: The incidence vector x' , which sets x_e to 1 iff $e \in F'$, is a feasible solution to (IP).

Assume in contradiction there are some $S \subseteq V$ such that $f(S) = 1$ and $x'(\delta(S)) = 0$. Denote by N_1, \dots, N_p components of F' .

IP solution feasibility: Theorem 2.3

Theorem 2.3: The incidence vector x' , which sets x_e to 1 iff $e \in F'$, is a feasible solution to (IP).

Assume in contradiction there are some $S \subseteq V$ such that $f(S) = 1$ and $x'(\delta(S)) = 0$. Denote by N_1, \dots, N_p components of F' .

- Since $x'(\delta(S)) = 0$ it must hold that for every N_i , either $N_i \cap S = \emptyset$ or $N_i \cap S = N_i$

IP solution feasibility: Theorem 2.3

Theorem 2.3: The incidence vector x' , which sets x_e to 1 iff $e \in F'$, is a feasible solution to (IP).

Assume in contradiction there are some $S \subseteq V$ such that $f(S) = 1$ and $x'(\delta(S)) = 0$. Denote by N_1, \dots, N_p components of F' .

- Since $x'(\delta(S)) = 0$ it must hold that for every N_i , either $N_i \cap S = \emptyset$ or $N_i \cap S = N_i$
- Therefore, we can write that $S = N_{i_1} \cup \dots \cup N_{i_k}$.

IP solution feasibility: Theorem 2.3

Theorem 2.3: The incidence vector x' , which sets x_e to 1 iff $e \in F'$, is a feasible solution to (IP).

Assume in contradiction there are some $S \subseteq V$ such that $f(S) = 1$ and $x'(\delta(S)) = 0$. Denote by N_1, \dots, N_p components of F' .

- Since $x'(\delta(S)) = 0$ it must hold that for every N_i , either $N_i \cap S = \emptyset$ or $N_i \cap S = N_i$
- Therefore, we can write that $S = N_{i_1} \cup \dots \cup N_{i_k}$.
- However, by the *Lemma 2.2* $f(N_i) = 0$ for every i , therefore, by the disjointness of f , $f(S) = 0$, contradicting our assumption.

IP solution feasibility: Theorem 2.3

Theorem 2.3: The incidence vector x' , which sets x_e to 1 iff $e \in F'$, is a feasible solution to (IP).

Assume in contradiction there are some $S \subseteq V$ such that $f(S) = 1$ and $x'(\delta(S)) = 0$. Denote by N_1, \dots, N_p components of F' .

- Since $x'(\delta(S)) = 0$ it must hold that for every N_i , either $N_i \cap S = \emptyset$ or $N_i \cap S = N_i$
- Therefore, we can write that $S = N_{i_1} \cup \dots \cup N_{i_k}$.
- However, by the *Lemma 2.2* $f(N_i) = 0$ for every i , therefore, by the disjointness of f , $f(S) = 0$, contradicting our assumption.

We conclude that no such S exists, and therefore the solution, found by the algorithm, is feasible.

The Algorithm

(IP) solution feasibility

The approximation factor

The approximation factor

Theorem 2.3: the algorithm is a $(2 - \frac{2}{|A|})$ -approximation

How each of the sums change?

Why inactive component can not be a leaf in H : construction

Why inactive component can not be a leaf in H : proof

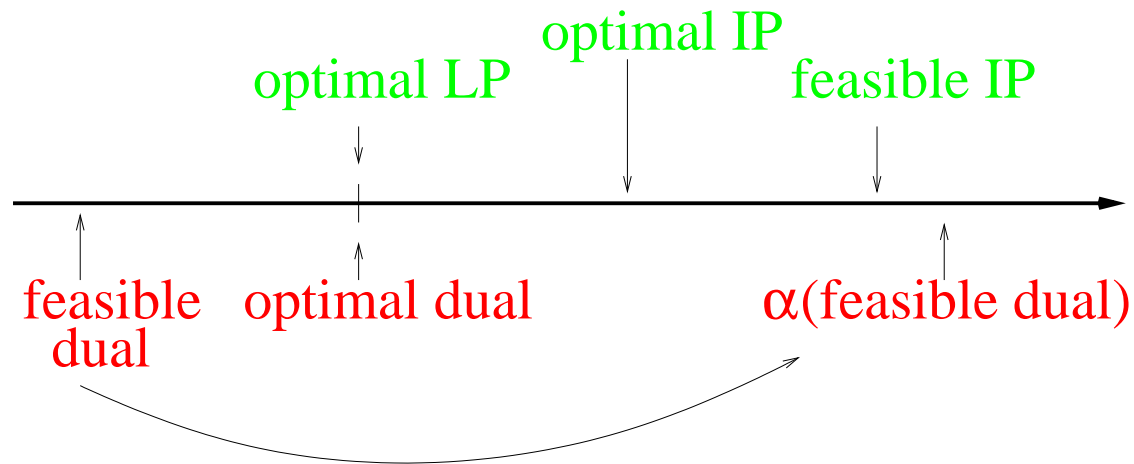
Sum of the degrees of all active components in H

Implementation

Additions

The approximation factor

The approximation factor



- The value of the feasible dual problem solution is the value of LB by the end of the execution.
- The value of the (IP) solution is the sum of all the edges weights in F' .

We will show that at every algorithm step the contribution to the value of the (IP) solution is bounded by the $(2 - \frac{2}{|A|})$ times the contribution to the dual problem solution, and therefore, at the end, the (IP) solution is at most $(2 - \frac{2}{|A|})$ times greater than the dual problem solution.

Theorem 2.3: the algorithm is a $(2 - \frac{2}{|A|})$ -approximation

Let us denote by Z_{LP}^* the value of the optimal solution of the Linear Program and by Z_{IP}^* the value of the optimal solution to the Integer Program.

Theorem 2.3: The algorithm produces a set of edges F' and a value LB such that

$$\sum_{e \in F'} c_e \leq (2 - \frac{2}{|A|})LB = (2 - \frac{2}{|A|}) \sum_{S \subset V} y_S \leq (2 - \frac{2}{|A|})Z_{LP}^* \leq (2 - \frac{2}{|A|})Z_{IP}^*$$

Hence the algorithm is a $(2 - \frac{2}{|A|})$ -approximation algorithm for the constrained forest problem for any proper function f .

Since $\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S$ we can write that

$\sum_{e \in F'} c_e = \sum_{S \subset V} y_S |F' \cap \delta(S)|$. To prove the theorem we will show that in each iteration:

$$\Delta \sum_{S \subset V} y_S |F' \cap \delta(S)| \leq \Delta (2 - \frac{2}{|A|}) \sum_{S \subset V} y_S$$

How each of the sums change?

- Lets us denote by \mathcal{C}^1 all the active components of \mathcal{C} in the current iteration

In each iteration loop the left-hand side of the inequality increase by

$$\sum_{C \in \mathcal{C}^1} \varepsilon |F' \cap \delta(S)| = \varepsilon \sum_{C \in \mathcal{C}^1} |F' \cap \delta(S)|$$

While the right-hand side increases by

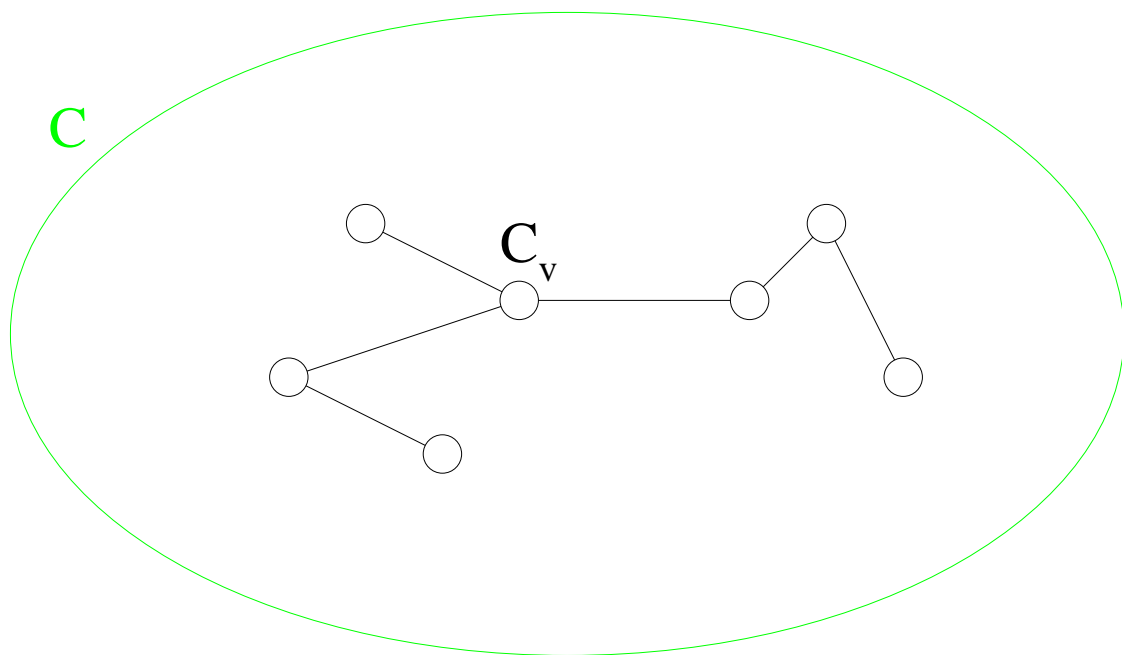
$$\varepsilon \cdot |\mathcal{C}^1|$$

Let us build a new graph, H . We present each segment of \mathcal{C} as a vertex in graph, and each edge of F' that connects two different segments of the current iteration becomes an edge in H . Now we remove all vertexes of degree 0, that correspond to inactive components of \mathcal{C} . We need to prove that the average degree of an active component is at most $(2 - \frac{2}{|A|})$.

Why inactive component can not be a leaf in H : construction

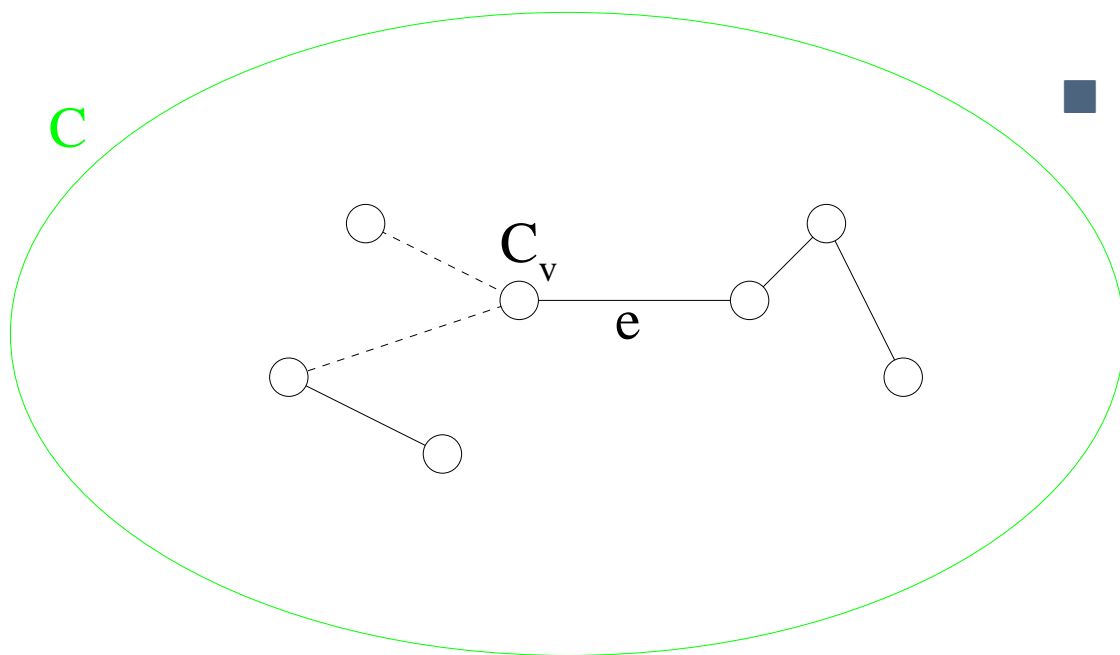
Notice that H is a forest. Assume in contradiction there is a passive component C_v that turns to a leaf in H .

- Initially this component C_v was part of some component C of F .



Why inactive component can not be a leaf in H : construction

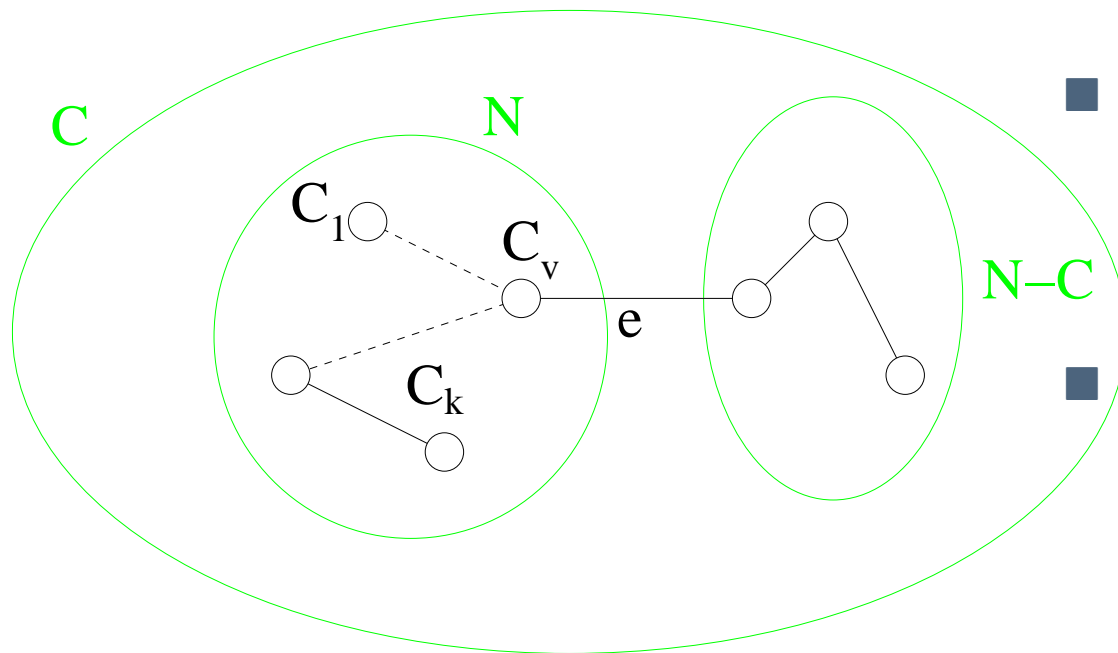
Notice that H is a forest. Assume in contradiction there is a passive component C_v that turns to a leaf in H .



- Initially this component C_v was part of some component C of F .
- Some of the edges adjacent to C_v in F was removed, making C_v a leaf. Let us denote the only edge that left in F' as e

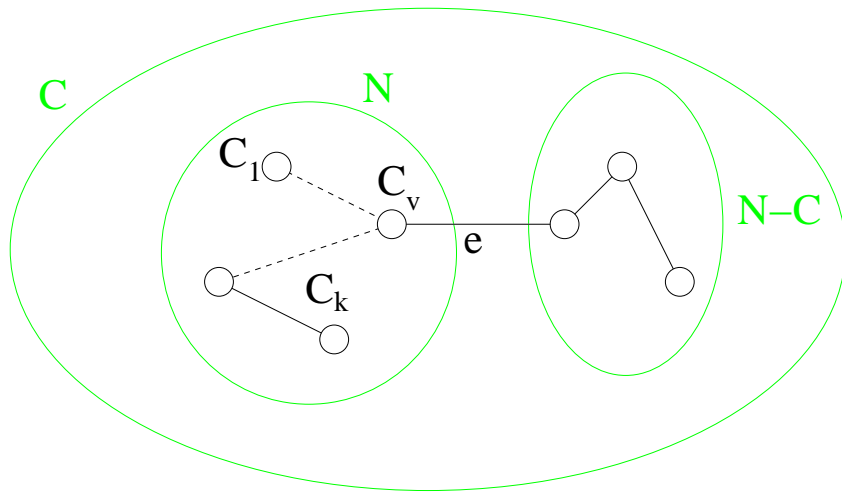
Why inactive component can not be a leaf in H : construction

Notice that H is a forest. Assume in contradiction there is a passive component C_v that turns to a leaf in H .



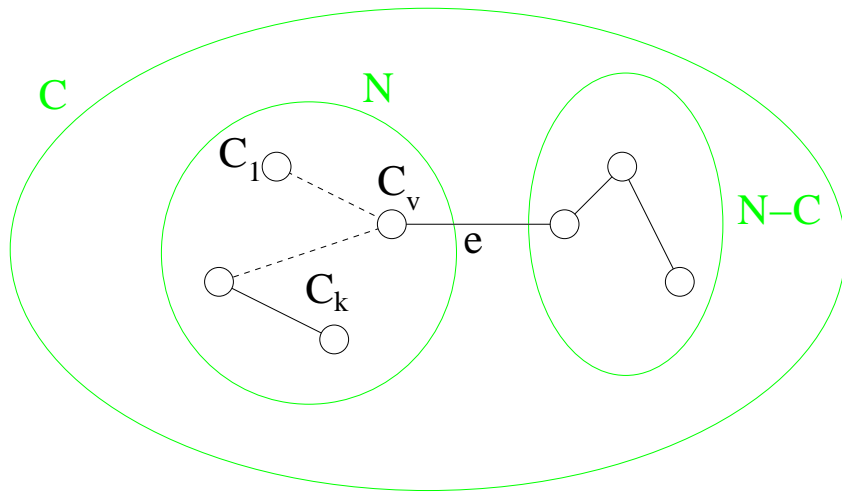
- Initially this component C_v was part of some component C of F .
- Some of the edges adjacent to C_v in F was removed, making C_v a leaf. Let us denote the only edge that left in F' as e
- If we remove e from the graph, C will be divided into two components. Denote the component C_v belongs to as N and the other as $C - N$. After the edges adjacent to C_v were removed from F , N is partitioned into components once again. Let us denote them by C_1, \dots, C_k . Note that C_v is a component by itself.

Why inactive component can not be a leaf in H : proof



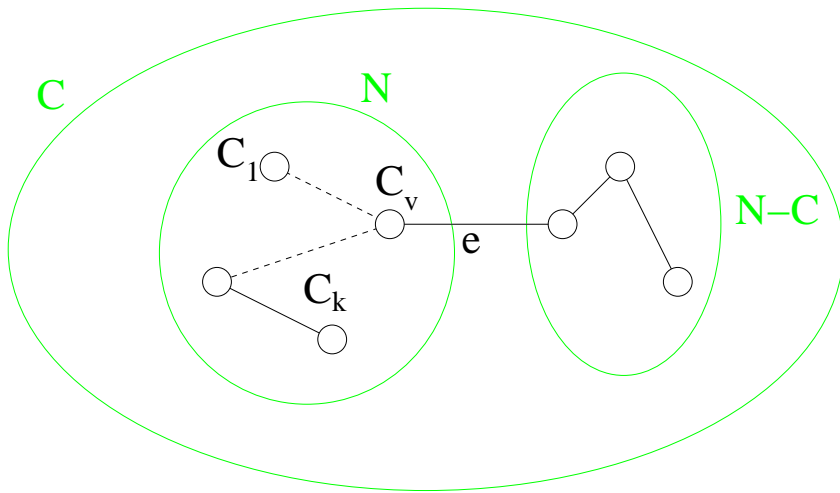
- Since the edges $(C_v, C_1), \dots, (C_v, C_k)$ are not in H , we know that $f(C_1) = \dots = f(C_k) = 0$ (by the construction of F').

Why inactive component can not be a leaf in H : proof



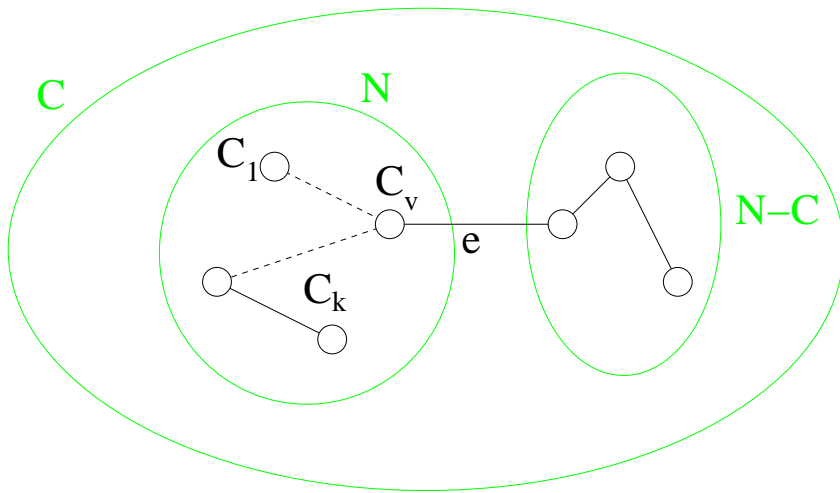
- Since the edges $(C_v, C_1), \dots, (C_v, C_k)$ are not in H , we know that $f(C_1) = \dots = f(C_k) = 0$ (by the construction of F').
- Therefore, by disjointness of f , $f(\cup_{i=1}^k C_i) = 0$.

Why inactive component can not be a leaf in H : proof



- Since the edges $(C_v, C_1), \dots, (C_v, C_k)$ are not in H , we know that $f(C_1) = \dots = f(C_k) = 0$ (by the construction of F').
- Therefore, by disjointness of f , $f(\cup_{i=1}^k C_i) = 0$.
- $f(C_v) = 0$ by assumption, hence, again by disjointness of f , $f(N) = 0$.

Why inactive component can not be a leaf in H : proof



- Since the edges $(C_v, C_1), \dots, (C_v, C_k)$ are not in H , we know that $f(C_1) = \dots = f(C_k) = 0$ (by the construction of F').
- Therefore, by disjointness of f , $f(\cup_{i=1}^k C_i) = 0$.
- $f(C_v) = 0$ by assumption, hence, again by disjointness of f , $f(N) = 0$.
- We know, by construction of F that $f(C) = 0$, therefore, by Observation 2.1, $f(C - N) = 0$. But in this case e must not be in F' . Contradiction.

Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.

Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.
- The degree of a vertex that corresponds to inactive component is at least 2.

Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.
- The degree of a vertex that corresponds to inactive component is at least 2.
- Therefore, the sum of the degrees of all active components is at most $2(|\mathcal{C}| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|)$

number of inactive components



Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.
- The degree of a vertex that corresponds to inactive component is at least 2.
- Therefore, the sum of the degrees of all active components is at most $2(|\mathcal{C}| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|) = 2(|\mathcal{C}^1| + |\mathcal{C}| - |\mathcal{C}^1| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|)$

Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.
- The degree of a vertex that corresponds to inactive component is at least 2.
- Therefore, the sum of the degrees of all active components is at most
$$2(|\mathcal{C}| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|) = 2(|\mathcal{C}^1| + |\mathcal{C}| - |\mathcal{C}^1| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|) = 2|\mathcal{C}^1| - 2$$

Sum of the degrees of all active components in H

- Since H is a forest the sum of all degrees of all the vertexes is at most $2(|\mathcal{C}| - 1)$.
- The degree of a vertex that corresponds to inactive component is at least 2.
- Therefore, the sum of the degrees of all active components is at most $2(|\mathcal{C}| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|) = 2(|\mathcal{C}^1| + |\mathcal{C}| - |\mathcal{C}^1| - 1) - 2(|\mathcal{C}| - |\mathcal{C}^1|) = 2|\mathcal{C}^1| - 2$

We conclude that:

$$\varepsilon \sum_{C \in \mathcal{C}^1} |F' \cap \delta(S)| \leq \varepsilon(2|\mathcal{C}^1| - 2) = \varepsilon|\mathcal{C}^1|(2 - \frac{2}{|\mathcal{C}^1|}) \leq \varepsilon|\mathcal{C}^1|(2 - \frac{2}{|A|})$$

Since $|A| \geq |\mathcal{C}^1|$.

The Algorithm

(IP) solution feasibility

The approximation
factor

Implementation

The algorithm
implementation

Additions

Implementation

The algorithm implementation

- Computation of $f(S)$ can be done in $O(n)$ time for any problem of interest, so the computation need to be performed at most $O(n)$ times.

The algorithm implementation

- Computation of $f(S)$ can be done in $O(n)$ time for any problem of interest, so the computation need to be performed at most $O(n)$ times.
- Components can be maintained as a union-find structure, hence all merging takes at most $O(n\alpha(n, n))$ time. (α is the inverse Ackermann function, the function grows very slowly)

The algorithm implementation

- Computation of $f(S)$ can be done in $O(n)$ time for any problem of interest, so the computation need to be performed at most $O(n)$ times.
- Components can be maintained as a union-find structure, hence all merging takes at most $O(n\alpha(n, n))$ time. (α is the inverse Ackermann function, the function grows very slowly)
- To find a minimum edge and to check whether this edge connects two different components we need
 - $O(n\alpha(m, n))$ steps in each iteration under a naive approach, resulting the overall running time of $O(nm\alpha(m, n))$ for the main loop.
 - Managing a special data structure of edges and reducing the number of candidate edges, we can reduce the iteration complexity to $O(n \log n)$, and, therefore, the main loop complexity will become $O(n^2 \log n)$, which is preferable for dense graphs.

The algorithm implementation

- Computation of $f(S)$ can be done in $O(n)$ time for any problem of interest, so the computation need to be performed at most $O(n)$ times.
- Components can be maintained as a union-find structure, hence all merging takes at most $O(n\alpha(n, n))$ time. (α is the inverse Ackermann function, the function grows very slowly)
- To find a minimum edge and to check whether this edge connects two different components we need
 - $O(n\alpha(m, n))$ steps in each iteration under a naive approach, resulting the overall running time of $O(nm\alpha(m, n))$ for the main loop.
 - Managing a special data structure of edges and reducing the number of candidate edges, we can reduce the iteration complexity to $O(n \log n)$, and, therefore, the main loop complexity will become $O(n^2 \log n)$, which is preferable for dense graphs.
- Computation of F' from F takes $O(n)$ time.

The Algorithm

(IP) solution feasibility

The approximation factor

Implementation

Additions

A list of problems where the system is applicable

Extensions

Additions

A list of problems where the system is applicable

- Minimum spanning tree problem: the algorithm is accurate.
- The shortest path problem: the algorithm is accurate
- The generalized Steiner tree problem: approximation factor of $2 - \frac{2}{k}$ where $k = |\cup T_i|$
- The minimum weight perfect matching problem. Approximation factor of $2 - \frac{2}{n}$ when the edge costs obey the triangle inequality. However, the problem can be solved in polynomial time, but for practical purposes less accurate but faster algorithm is preferable.
- Point-to-Point connection problem. Approximation factor of $2 - \frac{2}{p}$ is achieved, where p is the number of pairs
- Exact partitioning problem. For instances that satisfy the triangle inequality approximation factor of $2 - \frac{2}{n}$ is achieved.

- Weakening the conditions for proper function enables using the algorithm for solving additional problems. However, algorithm should be changes appropriately. For example, if symmetry is not required, one of the vertexes should be defined as a root.
- Also a variation of disjointness is considered.

The same technique used in the algorithm can be applied to additional problems. Two examples are considered:

- Prize-Collecting Steiner tree and
- Prize collecting travelling salesman

As before, the problems are presented as Integer Programs, the dual problem is formulated and the algorithm that behaves in the same greedy way is proposed. In these problems there are two groups of constrains on the dual variable, one comes for the edge cost and the other for vertexes prizes. While making the former constrain tight leads to a merge between two segments, the other constrain de-constructs them. However, it is proven that the algorithm still achieves $(2 - \frac{2}{n-1})$ approximation.