Steiner Networks

J. Yancey

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

April 25, 2014

Outline

Outline

Outline

Steiner Network Problem

Steiner Network Problem

We are given:

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$
- a connectivity requirement function r mapping unordered pairs of vertices to Z^+

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$
- a connectivity requirement function r mapping unordered pairs of vertices to Z^+
- and a function $u : E \to \mathbb{Z}^+ \cup \{\infty\}$ stating an upper bound on the number of copies of edge e we are allowed to use

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$
- a connectivity requirement function r mapping unordered pairs of vertices to Z^+
- and a function $u : E \to \mathbb{Z}^+ \cup \{\infty\}$ stating an upper bound on the number of copies of edge e we are allowed to use
- if $u_e = \infty$, there is no upper bound for edge e.

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$
- a connectivity requirement function r mapping unordered pairs of vertices to Z^+
- and a function $u : E \to \mathbb{Z}^+ \cup \{\infty\}$ stating an upper bound on the number of copies of edge e we are allowed to use
- if $u_e = \infty$, there is no upper bound for edge e.

We want to find a minimum cost multigraph on the vertex set V that has $r(u, v)$ edge disjoint paths for each pair of vertices $u, v \in V$.

Steiner Network Problem

We are given:

- an undirected graph $G = (V, E)$
- a cost function defined over edges, $c : E \to \mathbb{Q}^+$
- a connectivity requirement function r mapping unordered pairs of vertices to Z^+
- and a function $u : E \to \mathbb{Z}^+ \cup \{\infty\}$ stating an upper bound on the number of copies of edge e we are allowed to use
- if $u_e = \infty$, there is no upper bound for edge e.

We want to find a minimum cost multigraph on the vertex set V that has $r(u, v)$ edge disjoint paths for each pair of vertices $u, v \in V$. Each copy of edge e used for constructing this graph will cost $c(e)$.

Outline

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem.

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u, v)|u \in S \text{ and } v \in \bar{S}\}$.

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \rightarrow \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u, v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u, v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \rightarrow \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u, v) | u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

$$\left\{ \right.$$

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u,v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \end{cases}$$

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u,v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e:e \in \delta(S)} x_e \geq f(S), S \subseteq V \end{cases}$$

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u,v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e:e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \in \mathbb{Z}^+, e \in E \text{ and } u_e = \infty \end{cases}$$

Inter programming formulation

First we define a *cut requirement function*, $f : 2^V \to \mathbb{Z}^+$, as we did for the Steiner forest problem. For every $S \subseteq V, f(S)$ is defined to be the largest connectivity requirement separated by the cut $(S, \bar{S})$, i.e., $f(S) = max\{r(u,v)|u \in S \text{ and } v \in \bar{S}\}$.

Integer Program

Our integer program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e:e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \in \mathbb{Z}^+, e \in E \text{ and } u_e = \infty \\ & x_e \in \{0, 1, \cdots, u_e\}, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{1}$$

LP-relaxation

LP-relaxation

Our LP program is:

LP-relaxation

Our LP program is:

$$\left\{ \right.$$

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ & \\ & \\ & \end{cases}$$

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e : e \in \delta(S)} x_e \geq f(S), S \subseteq V \end{cases}$$

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e : e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \end{cases}$$

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e : e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e: e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

Notes

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e: e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

Notes

- As shown in previous lectures, certain **NP**-hard problems, such as vertex cover, admit LP-relaxations having the property that they always have a half-integral optimal solution.

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e: e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

Notes

- As shown in previous lectures, certain **NP**-hard problems, such as vertex cover, admit LP-relaxations having the property that they always have a half-integral optimal solution.
- Rounding up all halves to 1 in such a solution leads to a factor 2 approximation algorithm.

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e: e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

Notes

- As shown in previous lectures, certain **NP**-hard problems, such as vertex cover, admit LP-relaxations having the property that they always have a half-integral optimal solution.
- Rounding up all halves to 1 in such a solution leads to a factor 2 approximation algorithm.
- Does relaxation have this property?

LP-relaxation

Our LP program is:

$$\begin{cases} \text{minimize} & \sum_{e \in E} c_e x_e \\ \text{subject to} & \sum_{e: e \in \delta(S)} x_e \geq f(S), S \subseteq V \\ & x_e \geq 0, e \in E \text{ and } u_e = \infty \\ & u_e \geq x_e \geq 0, e \in E \text{ and } u_e \neq \infty \end{cases} \tag{2}$$

Notes

- As shown in previous lectures, certain **NP**-hard problems, such as vertex cover, admit LP-relaxations having the property that they always have a half-integral optimal solution.
- Rounding up all halves to 1 in such a solution leads to a factor 2 approximation algorithm.
- Does relaxation have this property? The following lemma shows that the answer is ”no”

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.
- Therefore, any feasible solution must have cost at least 5.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.
- Therefore, any feasible solution must have cost at least 5.
- Hence, the solution given above is in fact optimal.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.
- Therefore, any feasible solution must have cost at least 5.
- Hence, the solution given above is in fact optimal.
- Any solution with $x_e = 1$ for some edge e must have cost exceeding 5, since additional edges are required to connect the endpoints of e to the rest of the graph.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.
- Therefore, any feasible solution must have cost at least 5.
- Hence, the solution given above is in fact optimal.
- Any solution with $x_e = 1$ for some edge e must have cost exceeding 5, since additional edges are required to connect the endpoints of e to the rest of the graph.
- Therefore, any half integral solution of cost 5 would have to pick, to the extent of one half each, the edges of a Hamiltonian cycle.

Lemma 1

Consider the Petersen graph with a connectivity requirement of 1 between each pair of vertices and with each edge of unit cost. Relaxation does not have a half-integral optimal solution for this instance.

Proof.

- Consider the fractional solution $x_e = 1/3$ for each edge e.
- Since the Petersen graph is 3-edge connected, this is a feasible solution.
- The cost of this solution is 5.
- In any feasible solution, the sum of edge variables incident at any vertex must be at least 1, to allow connectivity to the other vertices.
- Therefore, any feasible solution must have cost at least 5.
- Hence, the solution given above is in fact optimal.
- Any solution with $x_e = 1$ for some edge e must have cost exceeding 5, since additional edges are required to connect the endpoints of e to the rest of the graph.
- Therefore, any half integral solution of cost 5 would have to pick, to the extent of one half each, the edges of a Hamiltonian cycle.
- Since the Petersen graph has no Hamiltonian cycles, there is no half-integral optimal solution.

□

Outline

Extreme point solution

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.
- Thick edges are picked to the extent of 1/2,

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.
- Thick edges are picked to the extent of 1/2, thin edges to the extent of 1/4,

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.
- Thick edges are picked to the extent of 1/2, thin edges to the extent of 1/4, and the missing edges is not picked.

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.
- Thick edges are picked to the extent of 1/2, thin edges to the extent of 1/4, and the missing edges is not picked.
- The isomophism group of the Petersen graph is edge-transitive, and there are 15 related extreme point solutions.

Extreme point solution

- Let us say that an extreme point solution for an LP is a feasible solution that cannot be written as the convex combination of two feasible solutions.
- The solution $x_e = 1/3$, for each edge e, is not an extreme point solution.
- An extreme point optimal solution is shown in figure 1.
- Thick edges are picked to the extent of 1/2, thin edges to the extent of 1/4, and the missing edges is not picked.
- The isomophism group of the Petersen graph is edge-transitive, and there are 15 related extreme point solutions.
- The solution $x_e = 1/3$ for each edge e is the average of these.

Extreme point solution cont.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.
- We obtain a 2-algorithm by rounding up these edges and iterating.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.
- We obtain a 2-algorithm by rounding up these edges and iterating.
- Let H be the set of edges picked by the algorithm at some point.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.
- We obtain a 2-algorithm by rounding up these edges and iterating.
- Let H be the set of edges picked by the algorithm at some point.
- Then, the residual requirement of cut $(S, \bar{S})$ is $f'(S) = f(S) - |\delta_H(S)|$, where $\delta_H(S)$ represents the set of edges of H crossing the cut $(S, \bar{S})$.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.
- We obtain a 2-algorithm by rounding up these edges and iterating.
- Let H be the set of edges picked by the algorithm at some point.
- Then, the residual requirement of cut $(S, \bar{S})$ is $f'(S) = f(S) - |\delta_H(S)|$, where $\delta_H(S)$ represents the set of edges of H crossing the cut $(S, \bar{S})$.
- In general, the residual cut requirement function, f', may not correspond to the cut requirement function for nay set of connectivity requirements.

Extreme point solution cont.

- Although the extreme point solution is not half-integral, it picks some edges to the extent of half.
- We show that this is a property of any extreme point solution to our LP.
- We obtain a 2-algorithm by rounding up these edges and iterating.
- Let H be the set of edges picked by the algorithm at some point.
- Then, the residual requirement of cut $(S, \bar{S})$ is $f'(S) = f(S) - |\delta_H(S)|$, where $\delta_H(S)$ represents the set of edges of H crossing the cut $(S, \bar{S})$.
- In general, the residual cut requirement function, f', may not correspond to the cut requirement function for nay set of connectivity requirements.
- We will need the following definitions to characterize it:

Remark

Remark

- Sometimes submodularity is defined only with the first condition.

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.
- If A and B don't cross then either they are disjoint or one of these sets is contained in the other.

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.
- If A and B don't cross then either they are disjoint or one of these sets is contained in the other.

Definitions

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.
- If A and B don't cross then either they are disjoint or one of these sets is contained in the other.

Definitions

Function $f : 2^V \to \mathbb{Z}^+$ is said to be submodular if $f(V) = 0$, and for every two sets $A, B \subseteq V$, the following two conditions hold:

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.
- If A and B don't cross then either they are disjoint or one of these sets is contained in the other.

Definitions

Function $f : 2^V \to \mathbb{Z}^+$ is said to be submodular if $f(V) = 0$, and for every two sets $A, B \subseteq V$, the following two conditions hold:

- $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$

Remark

- Sometimes submodularity is defined only with the first condition.
- We will need to work with the stronger definition given prior.
- Two subsets of V, A and B, are said to cross if each of the sets, $A-B, B-A$, and $A \cap B$, is nonempty.
- If A and B don't cross then either they are disjoint or one of these sets is contained in the other.

Definitions

Function $f : 2^V \to \mathbb{Z}^+$ is said to be submodular if $f(V) = 0$, and for every two sets $A, B \subseteq V$, the following two conditions hold:

- $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$
- $f(A)_f(B) \geq f(A - B) + f(B - A)$

Lemma 2

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

- If sets A and B do not cross, then the two conditions given in the definition of submodular functions hold trivially.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

- If sets A and B do not cross, then the two conditions given in the definition of submodular functions hold trivially.
- Otherwise, edges having one endpoint in $A \cap B$ and the other in $A \bar{\cup} B$ (edge e_1 in figure 2) contribute to $\delta(A)$ and $\delta(B)$ but not to $\delta(A-B)$ or $\delta(B-A)$.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

- If sets A and B do not cross, then the two conditions given in the definition of submodular functions hold trivially.
- Otherwise, edges having one endpoint in $A \cap B$ and the other in $A \bar{\cup} B$ (edge e_1 in figure 2) contribute to $\delta(A)$ and $\delta(B)$ but not to $\delta(A-B)$ or $\delta(B-A)$.
- Similarly, edge e_2 below does not contribute to $\delta(A \cap B)$ or to $\delta(A \cup B)$.

Lemma 2

For any graph G on vertex set V, the function $|\delta_G(.)|$ is submodular.

Proof.

- If sets A and B do not cross, then the two conditions given in the definition of submodular functions hold trivially.
- Otherwise, edges having one endpoint in $A \cap B$ and the other in $A \bar{\cup} B$ (edge e_1 in figure 2) contribute to $\delta(A)$ and $\delta(B)$ but not to $\delta(A-B)$ or $\delta(B-A)$.
- Similarly, edge e_2 below does not contribute to $\delta(A \cap B)$ or to $\delta(A \cup B)$.
- The remaining edges contribute equally to both sides of both conditions.

□

Weakly submodular

Weakly submodular

Function $f : 2^V \to \mathbb{Z}$ is said to be weakly submodular if $f(V) = 0$ and for every two sets $A, B \subseteq V$, at least one of the following holds:

Weakly submodular

Function $f : 2^V \to \mathbb{Z}$ is said to be weakly submodular if $f(V) = 0$ and for every two sets $A, B \subseteq V$, at least one of the following holds:

- $f(A) + f(B) \leq f(A - B) + f(A - B)$

Weakly submodular

Function $f : 2^V \to \mathbb{Z}$ is said to be weakly submodular if $f(V) = 0$ and for every two sets $A, B \subseteq V$, at least one of the following holds:

- $f(A) + f(B) \leq f(A - B) + f(A - B)$
- $f(A)_f(B) \leq f(A \cap B) + f(B \cup A)$

It is easy to check that the original cut requirement function is weakly supermodular;

Weakly submodular

Function $f : 2^V \to \mathbb{Z}$ is said to be weakly submodular if $f(V) = 0$ and for every two sets $A, B \subseteq V$, at least one of the following holds:

- $f(A) + f(B) \leq f(A - B) + f(A - B)$
- $f(A)_f(B) \leq f(A \cap B) + f(B \cup A)$

It is easy to check that the original cut requirement function is weakly supermodular; by the following lemma, so is the residual cut requirement function f'.

Lemma 3

Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is a weakly supermodular function, then so is the residual cut requirement function f'.

Lemma 3

Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is a weakly supermodular function, then so is the residual cut requirement function f'.

Proof.

Suppose $f(A) + f(B) \leq f(A - B) + f(B - A)$; the proof of the other case is similar.

Lemma 3

Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is a weakly supermodular function, then so is the residual cut requirement function f'.

Proof.

Suppose $f(A) + f(B) \leq f(A-B) + f(B-A)$; the proof of the other case is similar. By lemma 2 $|\delta_H(A)| + |\delta_H(B)| \geq |\delta_H(A-B)| + |\delta_H(B-A)|$.

Lemma 3

Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is a weakly supermodular function, then so is the residual cut requirement function f'.

Proof.

Suppose $f(A) + f(B) \leq f(A-B) + f(B-A)$; the proof of the other case is similar. By lemma 2 $|\delta_H(A)| + |\delta_H(B)| \geq |\delta_H(A-B)| + |\delta_H(B-A)|$. Subtracting, we get $f'(A) + f'(B) \leq f'(A-B) + f'(B-A)$. □

Lemma 3

Let H be a subgraph of G. If $f : 2^{V(G)} \to \mathbb{Z}^+$ is a weakly supermodular function, then so is the residual cut requirement function f'.

Proof.

Suppose $f(A) + f(B) \leq f(A-B) + f(B-A)$; the proof of the other case is similar. By lemma 2 $|\delta_H(A)| + |\delta_H(B)| \geq |\delta_H(A-B)| + |\delta_H(B-A)|$. Subtracting, we get $f'(A) + f'(B) \leq f'(A-B) + f'(B-A)$. □

Theorem 1

For any weakly supermodular function f, any extreme point solution, x, to our LP must pick some edge to the extent of at least a half, i.e., $x_e \geq 1/2$ for at least one edge e.

Iterated Rounding

1. **Initialization:** $H \leftarrow \emptyset : f' \leftarrow f$.
2. While $f' \not\equiv 0$, do:
   Find an extreme optimal solution, x, to LP with cut requirements given by f'.
   For each edge e such that $x_e \geq 1/2$, include $\lceil x_e \rceil$ copies of e in H, and decrement u_e by this amount.
   Update f' : for $S \subseteq V$, $f'(S) \leftarrow f(S) - |\delta_H(S)|$.
3. Output H.

Note

Note

- The above algorithm achieves an approximation guarantee of factor 2 for an arbitrary weakly supermodular function f.

Note

- The above algorithm achieves an approximation guarantee of factor 2 for an arbitrary weakly supermodular function f.
- To establish a polynomial running time involves showing that an extreme optimal solution to LP can be found efficiently.

Note

- The above algorithm achieves an approximation guarantee of factor 2 for an arbitrary weakly supermodular function f.
- To establish a polynomial running time involves showing that an extreme optimal solution to LP can be found efficiently.
- If f is the original cut requirement function for some connectivity requirements, then a polynomial time implementation follows from the existence of a polynomial time separation oracle for each iteration.

Outline

Discussion

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.
- Given a solution x, construct a graph on vetex set V with capactiy x_e for each edge e.

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.
- Given a solution x, construct a graph on vetex set V with capactiy x_e for each edge e.
- Then, for each pair of vertices $u, v \in V$, check if this graph admits a flow of at least $r(u, v)$ from u to v.

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.
- Given a solution x, construct a graph on vetex set V with capactiy x_e for each edge e.
- Then, for each pair of vertices $u, v \in V$, check if this graph admits a flow of at least $r(u, v)$ from u to v.
- If not, we will get a violated cut, i.e., a cut $(S, \bar{S})$ such that $\delta_x(S) < f(S)$, where

$$\delta_x(S) = \sum_{e: e \in \delta(S)} x_e. \tag{3}$$

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.
- Given a solution x, construct a graph on vetex set V with capactiy x_e for each edge e.
- Then, for each pair of vertices $u, v \in V$, check if this graph admits a flow of at least $r(u, v)$ from u to v.
- If not, we will get a violated cut, i.e., a cut $(S, \bar{S})$ such that $\delta_x(S) < f(S)$, where

$$\delta_x(S) = \sum_{e: e \in \delta(S)} x_e. \tag{3}$$

- Let f' be the cut requirement function of a subsequent iteration.

Discussion

- For the first iteration, a separation oracle follows from a max-flow subroutine.
- Given a solution x, construct a graph on vetex set V with capactiy x_e for each edge e.
- Then, for each pair of vertices $u, v \in V$, check if this graph admits a flow of at least $r(u, v)$ from u to v.
- If not, we will get a violated cut, i.e., a cut $(S, \bar{S})$ such that $\delta_x(S) < f(S)$, where

$$\delta_x(S) = \sum_{e: e \in \delta(S)} x_e. \tag{3}$$

- Let f' be the cut requirement function of a subsequent iteration.
- Given a solution to LP for this function, say x', define x as follows: for each edge $e, x_e = x'_e + e_H$, where e_H is the number of copies of edge e in H.

Discussion cont.

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.
- Furthermore, this lemma also shows that there is no need to update f' explicitly after each iteration.

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.
- Furthermore, this lemma also shows that there is no need to update f' explicitly after each iteration.

Lemma 4

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.
- Furthermore, this lemma also shows that there is no need to update f' explicitly after each iteration.

Lemma 4

A cut $(S, \bar{S})$ is violated by solution x' under cut requirement function f' iff it is violated by solution x under cut requirement function f.

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.
- Furthermore, this lemma also shows that there is no need to update f' explicitly after each iteration.

Lemma 4

A cut $(S, \bar{S})$ is violated by solution x' under cut requirement function f' iff it is violated by solution x under cut requirement function f.

Proof.

Discussion cont.

- The following lemma shows that a separation oracle for the original function f leads to a separation oracle for f'.
- Furthermore, this lemma also shows that there is no need to update f' explicitly after each iteration.

Lemma 4

A cut $(S, \bar{S})$ is violated by solution x' under cut requirement function f' iff it is violated by solution x under cut requirement function f.

Proof.

Notice that $\delta_x(S) = \delta_{x'}(S) + |\delta_H(S)|$. Since $f(S) = f'(S) + |\delta_H(S)|, \delta_x(S) \geq$ iff $\delta_{x'}(S) \geq f'(S)$. □

2-Approximation Guarantee

2-Approximation Guarantee

Lemma 4 implies that solution x' is feasible for the cut requirement function f' iff solution x is feasible for f.

2-Approximation Guarantee

Lemma 4 implies that solution x' is feasible for the cut requirement function f' iff solution x is feasible for f. Assuming Theorem 1, which we will provide below, let us show that our algorithm achieves an approximation guarantee of 2.

2-Approximation Guarantee

Lemma 4 implies that solution x' is feasible for the cut requirement function f' iff solution x is feasible for f. Assuming Theorem 1, which we will provide below, let us show that our algorithm achieves an approximation guarantee of 2.

Theorem 2

Our Algorithm achieves an approximation guarantee of 2 for the Steiner network problem.

Proof.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration. Since H is obtained by rounding up nonzero components of $\hat{x}$ and each of these components is $\geq /2$, $\text{cost}(H) \leq 2 \cdot \text{cost}(\hat{x})$.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration. Since H is obtained by rounding up nonzero components of $\hat{x}$ and each of these components is $\geq /2$, $\text{cost}(H) \leq 2 \cdot \text{cost}(\hat{x})$.

Let f' be the residual requirement function after the first iteration and H' be the set of edges picked in subsequent iterations for satisfying f'.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration. Since H is obtained by rounding up nonzero components of $\hat{x}$ and each of these components is $\geq /2$, $\text{cost}(H) \leq 2 \cdot \text{cost}(\hat{x})$.

Let f' be the residual requirement function after the first iteration and H' be the set of edges picked in subsequent iterations for satisfying f'. The key observation is that $x - \hat{x}$ is a feasible solution for f', and thus by the induction hypothesis, $\text{cost}(H') \leq 2 \cdot \text{cost}(x - \hat{x})$.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration. Since H is obtained by rounding up nonzero components of $\hat{x}$ and each of these components is $\geq /2$, $\text{cost}(H) \leq 2 \cdot \text{cost}(\hat{x})$.

Let f' be the residual requirement function after the first iteration and H' be the set of edges picked in subsequent iterations for satisfying f'. The key observation is that $x - \hat{x}$ is a feasible solution for f', and thus by the induction hypothesis, $\text{cost}(H') \leq 2 \cdot \text{cost}(x - \hat{x})$. Let us denote $H + H'$ the edges of H together with those of H'.

Proof.

By induction on the number of iterations executed by the algorithm when run with a weakly supermodular cut requirement function f, we will prove that the cost of the integral solution obtained is within a factor two of the cost of the optimal fractional solution. For the base case, if f requires one iteration, the claim follows, since the algorithm rounds up only edges e with $x_e \geq 1/2$.

For the induction step, assume that x is the extreme optimal solution obtained in the first iteration. Obtain $\hat{x}$ from x by zeroing out components that are strictly smaller than 1/2. By Theorem 1, $\hat{x} \neq 0$. Let H be the set of edges picked in the first iteration. Since H is obtained by rounding up nonzero components of $\hat{x}$ and each of these components is $\geq /2$, $\text{cost}(H) \leq 2 \cdot \text{cost}(\hat{x})$.

Let f' be the residual requirement function after the first iteration and H' be the set of edges picked in subsequent iterations for satisfying f'. The key observation is that $x - \hat{x}$ is a feasible solution for f', and thus by the induction hypothesis, $\text{cost}(H') \leq 2 \cdot \text{cost}(x - \hat{x})$. Let us denote $H + H'$ the edges of H together with those of H'. Clearly, $H + H'$ satisfies f. Now,

$$\begin{aligned} cost(H + H') &\leq cost(H) + cost(H') && (4) \\ &\leq 2 \cdot cost(\hat{x}) + 2 \cdot cost(x - \hat{x}) \leq 2 \cdot cost(x) && (5) \end{aligned}$$

□