my vertex sparsifier

```
   a
   |
   b -- c
   |
   d
```

Akamai
my vertex sparsifier
How to Use a Vertex Sparsifier

Question

What if you believe me, that there is a good vertex sparsifier?
Question

What if you believe me, that there is a good vertex sparsifier?

i.e. we can represent the relevant communication properties on a graph with only 4 nodes!
How to Use a Vertex Sparsifier

Question

*What if you believe me, that there is a good vertex sparsifier?*

I.e. we can represent the relevant communication properties on a graph with only 4 nodes!

**Applications of Vertex Sparsification:**
How to Use a Vertex Sparsifier

Question
What if you believe me, that there is a good vertex sparsifier?

i.e. we can represent the relevant communication properties on a graph with only 4 nodes!

Applications of Vertex Sparsification:

- Save SPACE (store a much smaller network)
How to Use a Vertex Sparsifier

Question

What if you believe me, that there is a good vertex sparsifier?

i.e. we can represent the relevant communication properties on a graph with only 4 nodes!

Applications of Vertex Sparsification:

- Save **SPACE** (store a much smaller network)
- Save **TIME** (run algorithms on a much smaller network)
How to Use a Vertex Sparsifier

**Question**

*What if you believe me, that there is a good vertex sparsifier?*

i.e. we can represent the relevant communication properties on a graph with only 4 nodes!

**Applications of Vertex Sparsification:**

- Save **SPACE** (store a much smaller network)
- Save **TIME** (run algorithms on a much smaller network)
- **UNIFIES** many rounding algorithms for graph partitioning
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning

- Vertex Sparsification
  - Definitions
  - Zero-Sum Game

- Graph Partitioning, Revisited
- Learning via Polynomials
Outline

- Introduction
  - **Minimum Congestion Routing**
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
\[ \vec{f} = \begin{bmatrix} x & \leftrightarrow & b \\ 0 & \leftrightarrow & c \\ y & \leftrightarrow & a \\ z & \leftrightarrow & d \\ 0 & \leftrightarrow & b \\ 0 & \leftrightarrow & d \\ 0 & \leftrightarrow & c \\ \end{bmatrix} \]
\[ \vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{c} a \quad \leftrightarrow \quad b \\ a \quad \leftrightarrow \quad c \\ a \quad \leftrightarrow \quad d \\ b \quad \leftrightarrow \quad c \\ b \quad \leftrightarrow \quad d \\ c \quad \leftrightarrow \quad d \end{array} \]
\[ f = \begin{bmatrix}
  x \\
  0 \\
  y \\
  z \\
  0 \\
  0
\end{bmatrix} \]

\[
\text{congestion} = \frac{\text{total rate}}{\text{bandwidth}}
\]
\[
\vec{f} = \begin{bmatrix}
    x \\
    0 \\
    y \\
    z \\
    0 \\
    0
\end{bmatrix} \\
\text{congestion} = \frac{\text{total rate}}{\text{bandwidth}}
\]
\[
\vec{f} = \begin{bmatrix}
x \\ 0 \\ y \\ z \\ 0 \\ 0 \\ 0 \\
\end{bmatrix} \quad \begin{array}{c}
a \leftrightarrow b \\ a \leftrightarrow c \\ a \leftrightarrow d \\ b \leftrightarrow c \\ b \leftrightarrow d \\ c \leftrightarrow d \\
\end{array}
\]

congestion = \frac{\text{total rate}}{\text{bandwidth}}
\( \vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \)

congestion = \( \frac{\text{total rate}}{\text{bandwidth}} \)
min congestion $\leftrightarrow$ max throughput
(min max)

congestion = bandwidth
\((\text{min max})\)

\[
\vec{f} = \begin{bmatrix}
    x \\
    0 \\
    y \\
    z \\
    0 \\
    0
\end{bmatrix}
\begin{align*}
    a &\rightarrow b \\
    a &\rightarrow c \\
    a &\rightarrow d \\
    b &\rightarrow c \\
    b &\rightarrow d \\
    c &\rightarrow d
\end{align*}

congestion = \frac{\text{total rate}}{\text{bandwidth}}
Question

Can we find a communication network on just the terminals, so that minimum congestion routing is approximately preserved?
What to Preserve, and How Well?

Question

*Can we find a communication network on just the terminals, so that minimum congestion routing is approximately preserved?*

i.e. for all routing requests $\vec{f}$, $\text{cong}_G(\vec{f}) \approx \text{cong}_H(\vec{f})$
What to Preserve, and How Well?

Question

*Can we find a communication network on just the terminals, so that minimum congestion routing is approximately preserved?*

i.e. for all routing requests $\vec{f}$, $\text{cong}_G(\vec{f}) \approx \text{cong}_H(\vec{f})$

**Quality:** $\left( \max_{\vec{f}} \frac{\text{cong}_G(\vec{f})}{\text{cong}_H(\vec{f})} \right) \left( \max_{\vec{f}} \frac{\text{cong}_H(\vec{f})}{\text{cong}_G(\vec{f})} \right)$
What to Preserve, and How Well?

**Question**

*Can we find a communication network on just the terminals, so that minimum congestion routing is approximately preserved?*

i.e. for all routing requests $\vec{f}$, $\text{cong}_G(\vec{f}) \approx \text{cong}_H(\vec{f})$

**Quality:** $\left( \max_{\vec{f}} \frac{\text{cong}_G(\vec{f})}{\text{cong}_H(\vec{f})} \right) \left( \max_{\vec{f}} \frac{\text{cong}_H(\vec{f})}{\text{cong}_G(\vec{f})} \right)$

**Question**

*Should good quality vertex sparsifiers exist?*
Results (Good Vertex Sparsifiers Exist!)

\( K \) is the set of terminals (data centers):
Results (Good Vertex Sparsifiers Exist!)

\( K \) is the set of terminals (data centers):

- Can compute a vertex sparsifier of "quality" \( O\left(\frac{\log |K|}{\log \log |K|}\right) \) in general networks
Results (Good Vertex Sparsifiers Exist!)

$K$ is the set of terminals (data centers):

- Can compute a vertex sparsifier of "quality" $O\left(\frac{\log |K|}{\log \log |K|}\right)$ in general networks
- Can compute a vertex sparsifier of "quality" constant if the original network is

$$\{\text{planar, bounded treewidth, padded decomposition property, ...} \}$$
Results (Good Vertex Sparsifiers Exist!)

$K$ is the set of terminals (data centers):

- Can compute a vertex sparsifier of "quality" $O\left(\frac{\log |K|}{\log \log |K|}\right)$ in general networks.
- Can compute a vertex sparsifier of "quality" constant if the original network is \{planar, bounded treewidth, padded decomposition property, ...\} in quadratic time (approximately a single minimum congestion solve).
Results (Good Vertex Sparsifiers Exist!)

\( K \) is the set of terminals (data centers):

- Can compute a vertex sparsifier of "quality" \( O\left(\frac{\log |K|}{\log \log |K|}\right) \) in general networks
- Can compute a vertex sparsifier of "quality" **constant** if the original network is \{planar, bounded treewidth, padded decomposition property, ...\} in quadratic time (approximately a **single** minimum congestion solve)

**Examples:** road networks (planar), internet graph backbone (bounded treewidth), social networks (p.d.p.)
Results (Good Vertex Sparsifiers Exist!)

\( K \) is the set of terminals (data centers):

- Can compute a vertex sparsifier of "quality" \( O\left(\frac{\log |K|}{\log \log |K|}\right) \) in general networks
- Can compute a vertex sparsifier of "quality" constant if the original network is \( \{\) planar, bounded treewidth, padded decomposition property, \( \} \) in quadratic time (approximately a single minimum congestion solve)

Examples: road networks (planar), internet graph backbone (bounded treewidth), social networks (p.d.p.)

\( (\text{Makarychev, Makarychev}): \tilde{\Omega}\left(\sqrt{\log |K|}\right) \) "quality" is necessary
1. Moitra, ”Approximation algorithms with guarantees independent of the graph size”, FOCS 2009

2. Leighton, Moitra, ”Extensions and limits to vertex sparsification”, STOC 2010

3. Charikar, Leighton, Li, Moitra, ”Vertex sparsifiers and abstract rounding algorithms”, FOCS 2010

4. Makarychev, Makarychev, ”Metric extension operators, vertex sparsifiers and Lipschitz extendability”, FOCS 2010

5. Englert, Gupta, Krauthgamer, Räcke, Talgam-Cohen, Talwar, ”Vertex sparsifiers: new results from old techniques”, APPROX 2010

6. Chuzhoy. ”On vertex sparsifiers with Steiner nodes”, STOC 2012
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Observation

*What we really want are good routing schemes in the original network!*

---

**What we really want are good routing schemes in the original network!**
Observation

What we really want are good routing schemes in the original network!

There is a **CANONICAL** mapping of flows in $H$ to flows in $G$: 

Observation

What we really want are good routing schemes in the original network!

There is a CANONICAL mapping of flows in $H$ to flows in $G$:

Claim

A good vertex sparsifier can be SIMULATED with low overhead, in the original network
my vertex sparsifier

K = \{a, b, c, d\}

my vertex sparsifier

\begin{align*}
    a & \quad b & \quad c & \quad d \\
    1 & \quad 2 & \quad 4 & \\
    1 & \quad 1 & &
\end{align*}
my vertex sparsifier

K = \{a,b,c,d\}
my vertex sparsifier

\[ K = \{a, b, c, d\} \]
my vertex sparsifier

\[ K = \{a, b, c, d\} \]
my vertex sparsifier

\[ K = \{a, b, c, d\} \]
my vertex sparsifier

K = \{a,b,c,d\}

\begin{array}{c}
\text{my vertex sparsifier} \\
\begin{array}{ccc}
\bullet & a & \bullet \\
\bullet & b & \bullet \\
\bullet & c & \bullet \\
\end{array}
\end{array}
my vertex sparsifier

\[ f = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} \]

\[ f = \begin{bmatrix} a \leftrightarrow b \\ 0 \leftrightarrow c \\ a \leftrightarrow d \\ b \leftrightarrow c \\ 0 \leftrightarrow d \\ 0 \leftrightarrow d \end{bmatrix} \]

\[ K = \{a, b, c, d\} \]
$f = \text{my vertex sparsifier}$

$K = \{a,b,c,d\}$

$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix}$$

$\begin{align*}
\text{x} & \rightarrow \text{b} \\
\text{0} & \rightarrow \text{c} \\
\text{y} & \rightarrow \text{d} \\
\text{z} & \rightarrow \text{c} \\
\text{0} & \rightarrow \text{d} \\
\text{0} & \rightarrow \text{d} \\
\end{align*}$
$\vec{f} = \begin{bmatrix} x \\ y \\ z \\ 0 \\ 0 \end{bmatrix}$

$K = \{a, b, c, d\}$

`my vertex sparsifier`
Observation
What we really want are good routing schemes in the original network!

There is a **CANONICAL** mapping of flows in $H$ to flows in $G$:

Claim
A good vertex sparsifier can be **SIMULATED** with low overhead, in the original network
Observation

What we really want are good routing schemes in the original network!

There is a **CANONICAL** mapping of flows in $H$ to flows in $G$:

Claim

A good vertex sparsifier can be **SIMULATED** with low overhead, in the original network

For each routing request, run off-the-shelf algorithm on a 4 node network (instead of on a gigantic one)
Solving a Sequence of Routing Problems

Observation
What we really want are good routing schemes in the original network!

There is a **CANONICAL** mapping of flows in $H$ to flows in $G$:

Claim
A good vertex sparsifier can be **SIMULATED** with low overhead, in the original network
Observation

What we really want are good routing schemes in the original network!

There is a CANONICAL mapping of flows in $H$ to flows in $G$:

Claim

A good vertex sparsifier can be SIMULATED with low overhead, in the original network

COMPARE: Räcke’s oblivious routing scheme is $\Theta(\log |V|)$-competitive; ours is $O(\log |K|)$
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Graph Partitioning Made Easy

Graph Partitioning

- **Goal**: cut few edges, disconnect terminals according to some constraints
Graph Partitioning Made Easy

Graph Partitioning

- **Goal**: *cut few edges, disconnect terminals according to some constraints*
- **Diverse** *set of problems and applications*

---

Charikar, Leighton, Li, Moitra: Vertex sparsifiers yield many known approximation guarantees as a special case, and give new ones too!
Graph Partitioning Made Easy

Graph Partitioning

- **Goal**: cut few edges, disconnect terminals according to some constraints
- **Diverse** set of problems and applications

Can use Min-Cut Max-Flow Theorem to prove:

Claim

*Preserve flows* $\Rightarrow$ *Preserve cuts*
Graph Partitioning

- **Goal**: cut few edges, disconnect terminals according to some constraints
- **Diverse** set of problems and applications

Can use Min-Cut Max-Flow Theorem to prove:

**Claim**

Preserve flows $\Rightarrow$ Preserve cuts

*(Charikar, Leighton, Li, Moitra)*: Vertex sparsifiers yield many known approximation guarantees as a special case, and give new ones too!
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

Sparsifier $G'=(K,E')$
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

Sparsifier $G’=(K,E’)$

$h_K(a) = 5$
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

Sparsifier $G'=(K,E')$

$h_K(a) = 5$  

$h'(a) = 6$
General Approach: Cut Sparsifiers

Graph $G = (V, E)$

Sparsifier $G' = (K, E')$

$h_K(a) = 5$

$h'(a) = 6$
General Approach: Cut Sparsifiers

Graph \( G = (V, E) \)

Sparsifier \( G' = (K, E') \)

\[
\begin{align*}
  h_K(a) &= 5 \\
  h_K(b) &= 2 \\
  h'(a) &= 6
\end{align*}
\]
General Approach: Cut Sparsifiers

Graph $G = (V, E)$

$h_K(a) = 5$
$h_K(b) = 2$

Sparsifier $G' = (K, E')$

$h'(a) = 6$
$h'(b) = 2.5$
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

Sparsifier $G'=(K,E')$

$h_K(a) = 5$
$h_K(b) = 2$

$h'(a) = 6$
$h'(b) = 2.5$
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

- $h_K(a) = 5$
- $h_K(b) = 2$
- $h_K(ac) = 4$

Sparsifier $G'=(K,E')$

- $h'(a) = 6$
- $h'(b) = 2.5$
General Approach: Cut Sparsifiers

Graph $G=(V,E)$

- $h_K(a) = 5$
- $h_K(b) = 2$
- $h_K(ac) = 4$

Sparsifier $G'=(K,E')$

- $h'(a) = 6$
- $h'(b) = 2.5$
- $h'(ac) = 4.5$
General Approach: Cut Sparsifiers

Graph $G = (V, E)$

$\begin{align*}
  h_K(a) &= 5 \\
  h_K(b) &= 2 \\
  h_K(c) &= 3 \\
  h_K(d) &= 4 \\
  h_K(ab) &= 7 \\
  h_K(ac) &= 4
  \end{align*}$

Sparsifier $G' = (K, E')$

$\begin{align*}
  h'(a) &= 6 \\
  h'(b) &= 2.5 \\
  h'(c) &= 3.5 \\
  h'(d) &= 5 \\
  h'(ab) &= 7.5 \\
  h'(ac) &= 4.5
  \end{align*}$
General Approach: Cut Sparsifiers

\[ h(a) = 5 \quad h'(a) = 6 \]
\[ h(b) = 2 \]
\[ h'(b) = 2.5 \]
\[ h(c) = 3 \]
\[ h'(c) = 3.5 \]
\[ h(ab) = 7 \]
\[ h'(ab) = 7.5 \]
\[ h(ac) = 4 \]
\[ h'(ac) = 4.5 \]
\[ h(ad) = 5 \]
\[ h'(ad) = 5 \]

Quality = \[ \frac{5}{4} \]
A Useful Primitive

Definition

Let $f : V \rightarrow K$ is a 0-extension if for all $a \in K$, $f(a) = a$. 

$G = (V, E)$
A Useful Primitive

Definition

Let $f : V \rightarrow K$ be a 0-extension if for all $a \in K$, $f(a) = a$. 

$G = (V, E)$
A Useful Primitive

Definition

\[ G_f = (K, E_f) \]
A Useful Primitive

Definition

Let $f : V \rightarrow K$, is a 0-extension if for all $a \in K$, $f(a) = a$. 

$G_f = (K, E_f)$
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier

$G_f = (K, E_f)$
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier

$G_f = (K, E_f)$

$K - A$

$A$
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier

$G = (V,E)$

$K - A$

$A$
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier
A Useful Primitive

Lemma

$G_f$ is a Cut Sparsifier
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Proof Outline

1. Define a Zero-Sum Game
2. The Best Response is a 0-Extension Problem
3. Construct a Feasible Solution for the Linear Programming Relaxation
4. Round the solution to get a Valid Response

[Fakcharoenphol, Harrelson, Rao, Talwar 2003]
[Calinescu, Karloff, Rabani 2001]
Proof Outline

1. Define a Zero-Sum Game
The Extension-Cut Game
The Extension-Cut Game
The Extension-Cut Game

N(f, A)

P1

P2

A

f

N(f, A)
The Extension-Cut Game

P1

P2

A

N(f, A)

f
The Extension-Cut Game

P1

A

f

N(f,A)

P2

A={a}

K−A

A={a}

K−A
The Extension-Cut Game

$A = \{a\}$

$N(f, A)$

$K - A$

$P_2$

$P_1$

$A = \{a\}$

$K - A$
The Extension-Cut Game

\[ A = \{a\} \]

\[ f \]

\[ N(f, A) \]

\[ K \]

\[ h_{K(a)} = 5 \]

\[ A = \{a\} \]

\[ K - A \]
The Extension-Cut Game

A = \{a\}

h_K(a) = 5

P2

A

f

N(f,A)

P1

A = \{a\}

K - A

h_K(a) = 5

K - A
The Extension-Cut Game

\[ A = \{a\} \]

\[ N(f, A) \]

\[ h_{K}(a) = 5 \]

\[ K - A \]

\[ A = \{a\} \]

\[ K - A \]
The Extension-Cut Game

$A = \{a\}$

$N(f, A)$

$h_{K}(a) = 5$

$K-A$
The Extension-Cut Game

\[ A = \{a\} \]

\[ N(f, A) \]

\[ h_K(a) = 5 \]

\[ K - A \]
The Extension-Cut Game

$h_f(a) = 5$

$h_f(a) = 7$

$A = \{a\}$

$K - A$
The Extension-Cut Game

\[ N(f, A) = \frac{7}{5} \]

\( h_k(a) = 5 \)

\( h_f(a) = 7 \)
The Extension-Cut Game

\[ N(f,A) = \frac{h_f(A)}{h_K(A)} \]

\[ h_K(a) = 5 \]

\[ h_f(a) = 7 \]
Minmax

Theorem (von Neumann)

\[
\min_{\gamma} \max_{A} E_{f \leftarrow \gamma}[N(f, A)] = \max_{\lambda} \min_{f} E_{A \leftarrow \lambda}[N(f, A)]
\]
Minmax

Theorem (von Neumann)

\[
\min_{\gamma} \max_{A} E_{f \leftarrow \gamma}[N(f, A)] = \max_{\lambda} \min_{f} E_{A \leftarrow \lambda}[N(f, A)]
\]

Bound on game value implies that good Cut Sparsifiers exist!
Minmax

Theorem (von Neumann)

\[
\min_{\gamma} \max_{A} E_{f \leftarrow \gamma}[N(f, A)] = \max_{\lambda} \min_{f} E_{A \leftarrow \lambda}[N(f, A)]
\]

Bound on **game value** implies that good Cut Sparsifiers exist!

Let \( G' = \sum_f \gamma(f) G_f \) (no good response for the cut player)
Minmax

Theorem (von Neumann)

\[
\min_{\gamma} \max_{A} E_{f \leftarrow \gamma}[N(f, A)] = \max_{\lambda} \min_{f} E_{A \leftarrow \lambda}[N(f, A)]
\]

Bound on \textbf{game value} implies that good Cut Sparsifiers exist!

Let \( G' = \sum_f \gamma(f) G_f \) \( \text{(no good response for the cut player)} \)

Question

For every distribution \( \lambda \) on \( A \subset K \), is there a \textbf{good} response \( f \) for the extension player?
Proof Outline

1. Define a **Zero-Sum Game**

[Fakcharoenphol, Harrelson, Rao, Talwar 2003]

[Calinescu, Karloff, Rabani 2001]
Proof Outline

1. Define a **Zero-Sum Game**

2. The **Best Response** is a 0-Extension Problem

[Fakcharoenphol, Harrelson, Rao, Talwar 2003]
[Calinescu, Karloff, Rabani 2001]
Best Response?

Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$
Best Response?

Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$

$p = \frac{1}{2}$
Best Response?

Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}

\begin{align*}
p &= \frac{1}{2} \\
s &= \frac{1}{2h_K(\{a, b\})}
\end{align*}
Best Response?

Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$

\[ p = \frac{1}{2} \]

\[ s = \frac{1}{2h_K(\{a,b\})} \]

\[ p = \frac{1}{2} \]
Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$
Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$

$p = \frac{1}{2}$

$s = \frac{1}{2h_K(\{a, b\})}$

$p = \frac{1}{2}$

$s = \frac{1}{2h_K(\{a, d\})}$
Best Response?

Let $\mu = \frac{1}{2}\{a, b\} + \frac{1}{2}\{a, d\}$

\[
p = \frac{1}{2} \quad \text{and} \quad s = \frac{1}{2h_K(\{a, b\})}
\]

\[
p = \frac{1}{2} \quad \text{and} \quad s = \frac{1}{2h_K(\{a, d\})}
\]
Proof Outline

1. Define a **Zero-Sum Game**

2. The **Best Response** is a 0-Extension Problem
Proof Outline

1. Define a Zero-Sum Game
2. The Best Response is a 0-Extension Problem
3. Construct a Feasible Solution for the Linear Programming Relaxation
Proof Outline

1. Define a **Zero-Sum Game**

2. The **Best Response** is a 0-Extension Problem

3. Construct a **Feasible Solution** for the Linear Programming Relaxation

4. Round the solution to get a **Valid Response**

[Fakcharoenphol, Harrelson, Rao, Talwar '03]
[Calinescu, Karloff, Rabani '01]
Proof Outline

1. Define a **Zero-Sum Game**

2. The **Best Response** is a 0-Extension Problem

3. Construct a **Feasible Solution** for the Linear Programming Relaxation

4. Round the solution to get a **Valid Response**
   [Fakcharoenphol, Harrelson, Rao, Talwar ’03]
   [Calinescu, Karloff, Rabani ’01]
Summary, So Far

Non-constructive proof that good quality cut sparsifiers exist, through a zero sum game
Summary, So Far

Non-constructive proof that good quality cut sparsifiers exist, through a zero sum game

MORAL: Challenge someone else to prove you wrong (if he can’t, you’re right!)
Summary, So Far

Non-constructive proof that good quality cut sparsifiers exist, through a zero sum game

**MORAL**: Challenge someone else to prove you wrong (if he can’t, you’re right!)

This proof can be made constructive:
Summary, So Far

**Non-constructive** proof that good quality cut sparsifiers exist, through a zero sum game

**MORAL**: Challenge someone else to prove you wrong (if he can’t, you’re right!)

This proof can be made constructive:

- Solve a sequence of routing problems as fast as solving just one!
Summary, So Far

Non-constructive proof that good quality cut sparsifiers exist, through a zero sum game

**MORAL**: Challenge someone else to prove you wrong (if he can’t, you’re right!)

This proof can be made constructive:

- Solve a sequence of routing problems as fast as solving just one!
- Reduce all your graph partitioning problems to trees!
Outline

- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning

- Vertex Sparsification
  - Definitions
  - Zero-Sum Game

- Graph Partitioning, Revisited
- Learning via Polynomials
Outline

- Introduction
  - Minimum Congestion Routing
    - Application: Routing
    - Application: Graph Partitioning
  - Vertex Sparsification
    - Definitions
    - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials
Make the Problem Smaller AND Simpler

Recall, graph partitioning: cut few edges, disconnect terminals according to some constraints
Make the Problem Smaller AND Simpler

Recall, graph partitioning: cut few edges, disconnect terminals according to some constraints

Claim

*Graph partitioning problems are often easy to solve on trees*
Recall, graph partitioning: cut few edges, disconnect terminals according to some constraints

Claim

*Graph partitioning problems are often easy to solve on trees*

Question

*Can we use vertex sparsification to make the problem smaller and simpler?*

e.g. can we **ROUND** the graph to a tree (on just the terminals)?
Definition

We call an optimization problem a Fractional Graph Partitioning Problem if it can be written as

\[
\min \sum_{(u,v) \in E} c(u, v) d(u, v)
\]

s.t.

\[
d : V \times V \to \mathbb{R}^+ \text{ is a semi-metric}
\]

...
Fractional Graph Partitioning Problems

Definition

We call an optimization problem a Fractional Graph Partitioning Problem if it can be written as (for some monotone increasing function $f$):

$$\min \sum_{(u,v) \in E} c(u, v)d(u, v)$$

s.t.

$$d : V \times V \to \mathbb{R}^+ \text{ is a semi-metric}$$

$$f(d|_K) \geq 1$$
Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:**
   \[ f(d \mid K) = \min_i d(s_i, t_i) \]
   **Goal:** Separate all pairs of demands, cutting few edges

2. **Sparsest Cut:**
   \[ f(d \mid K) = \sum_i \text{dem}(i) d(s_i, t_i) \]
   **Goal:** Find a cut with small ratio

3. **Requirement Cut:**
   \[ f(d \mid K) = \min_i \text{MST}(R_i) p_i \]
   **Goal:** Separate all sets \( R_i \) into at least \( p_i \) components, cutting few edges
Consider the (standard) fractional relaxations for:

1. **Multi-Cut:**
   - **Goal:** Separate all pairs of demands, cutting few edges
Consider the (standard) fractional relaxations for:

1. **Multi-Cut**: \( f(d_{|K}) = \min_i d(s_i, t_i) \)
   
   **Goal**: Separate all pairs of demands, cutting few edges
Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:** \[ f(d|_K) = \min_i d(s_i, t_i) \]
   **Goal:** Separate all pairs of demands, cutting few edges

2. **Sparsest Cut:**
   **Goal:** Find a cut with small ratio
Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut**: \( f(d_{\mid K}) = \min_i d(s_i, t_i) \)
   *Goal*: Separate all pairs of demands, cutting few edges

2. **Sparsest Cut**: \( f(d_{\mid K}) = \sum_i \text{dem}(i)d(s_i, t_i) \)
   *Goal*: Find a cut with small ratio
Consider the (standard) fractional relaxations for:

1. **Multi-Cut:** \( f(d_{\mid K}) = \min_i d(s_i, t_i) \)
   **Goal:** Separate all pairs of demands, cutting few edges

2. **Sparsest Cut:** \( f(d_{\mid K}) = \sum_i dem(i) d(s_i, t_i) \)
   **Goal:** Find a cut with small ratio

3. **Requirement Cut:**
   **Goal:** Separate all sets \( R_i \) into at least \( p_i \) components, cutting few edges
Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:** $f(d_{|K}) = \min_i d(s_i, t_i)$
   **Goal:** Separate all pairs of demands, cutting few edges

2. **Sparsest Cut:** $f(d_{|K}) = \sum_i dem(i)d(s_i, t_i)$
   **Goal:** Find a cut with small ratio

3. **Requirement Cut:** $f(d_{|K}) = \min_i \frac{\text{MST}(R_i)}{p_i}$
   **Goal:** Separate all sets $R_i$ into at least $p_i$ components, cutting few edges
Theorem (Charikar, Leighton, Li, Moitra)

For any graph partitioning problem, the maximum integrality gap is at most $O(\log k)$ times the max integrality gap restricted to trees.
A Master Theorem

Theorem (Charikar, Leighton, Li, Moitra)

For any graph partitioning problem, the maximum integrality gap is at most $O(\log k)$ times the max integrality gap restricted to trees.

This yields many known integrality gaps for fractional graph partitioning problems (and new ones too):
A Master Theorem

Theorem (Charikar, Leighton, Li, Moitra)

*For any graph partitioning problem, the maximum integrality gap is at most $O(\log k)$ times the max integrality gap restricted to trees.*

This yields many known integrality gaps for fractional graph partitioning problems (and new ones too):

1. [Garg, Vazirani, Yannakakis]
A Master Theorem

Theorem (Charikar, Leighton, Li, Moitra)

For any graph partitioning problem, the maximum integrality gap is at most $O(\log k)$ times the max integrality gap restricted to trees.

This yields many known integrality gaps for fractional graph partitioning problems (and new ones too):

1. [Garg, Vazirani, Yannakakis]
2. [Linial, Londan, Rabinovich], [Aumann, Rabani]
A Master Theorem

Theorem (Charikar, Leighton, Li, Moitra)

For any graph partitioning problem, the maximum integrality gap is at most $O(\log k)$ times the max integrality gap restricted to trees.

This yields many known integrality gaps for fractional graph partitioning problems (and new ones too):

1. [Garg, Vazirani, Yannakakis]
2. [Linial, Londan, Rabinovich], [Aumann, Rabani]
3. [Gupta, Nagarajan, Ravi]
4. ...
Thanks! Any Questions?

- **Vertex sparsification**
  existence via an exponential-sized zero-sum game

- **Implications for routing**
  save space and time, when solving a sequence of problems

- **Implications for graph partitioning**
  general case can be reduced to trees (on the set of terminals)

- **Other**: Learning, Lattices, Convex Geometry