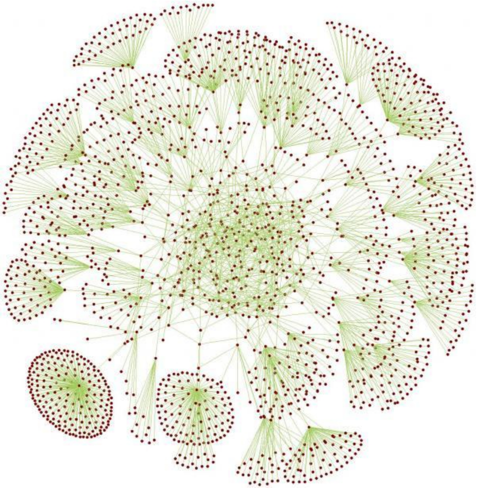
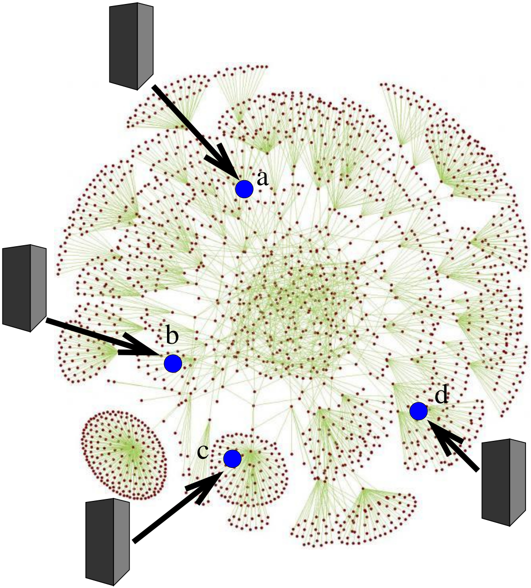
Vertex Sparsification

Ankur Moitra, IAS

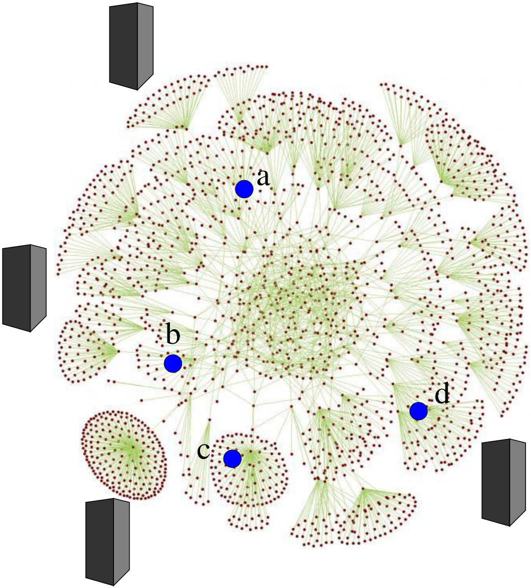
February 15th, 2012



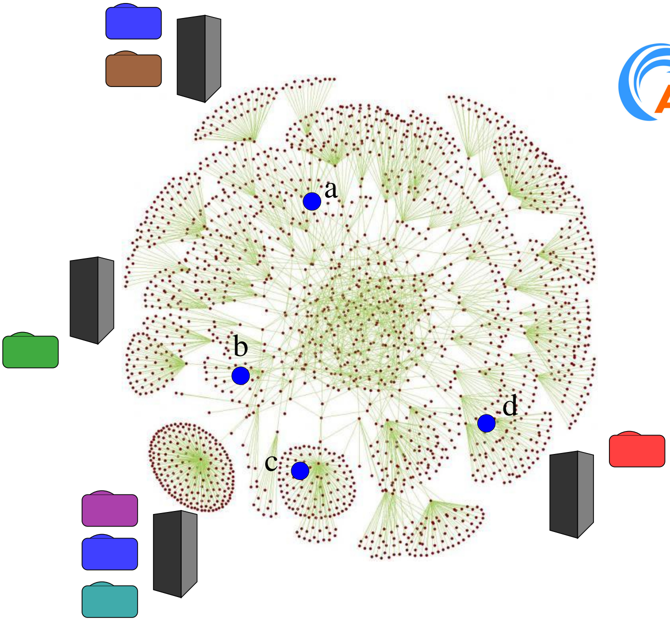
Akamai



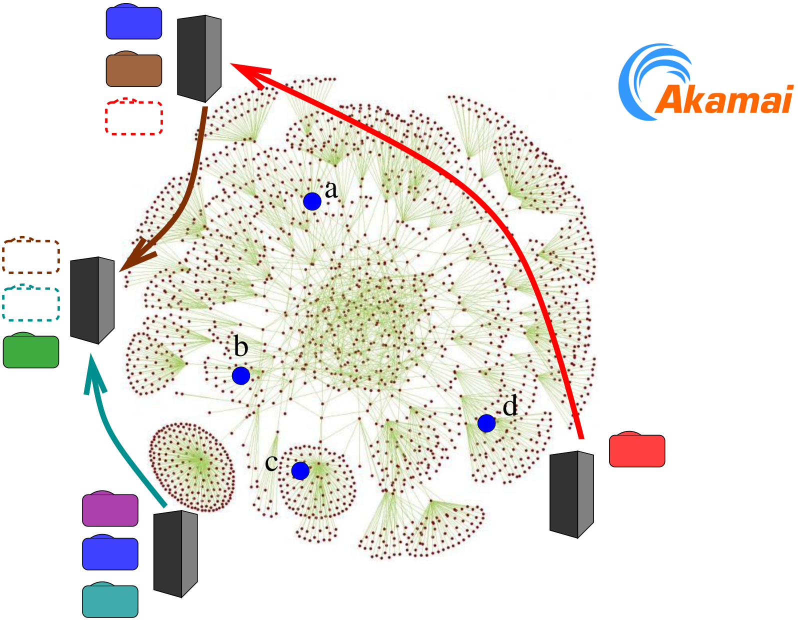
Akamai



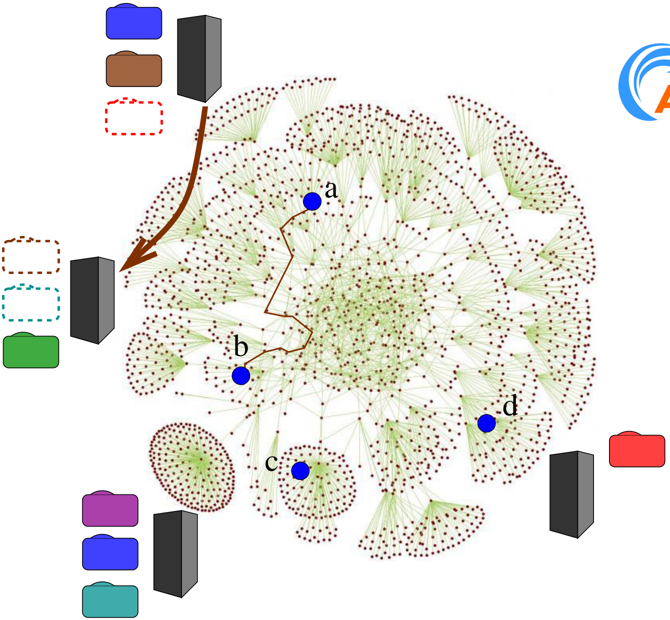
Akamai

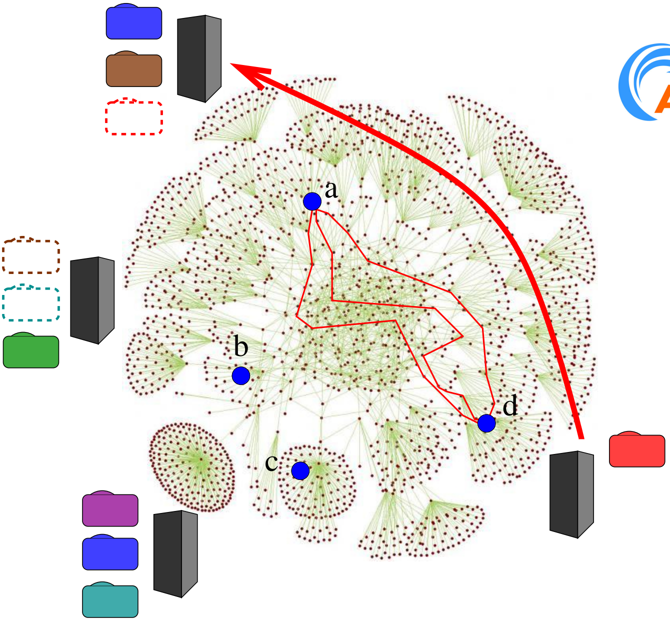


Akamai

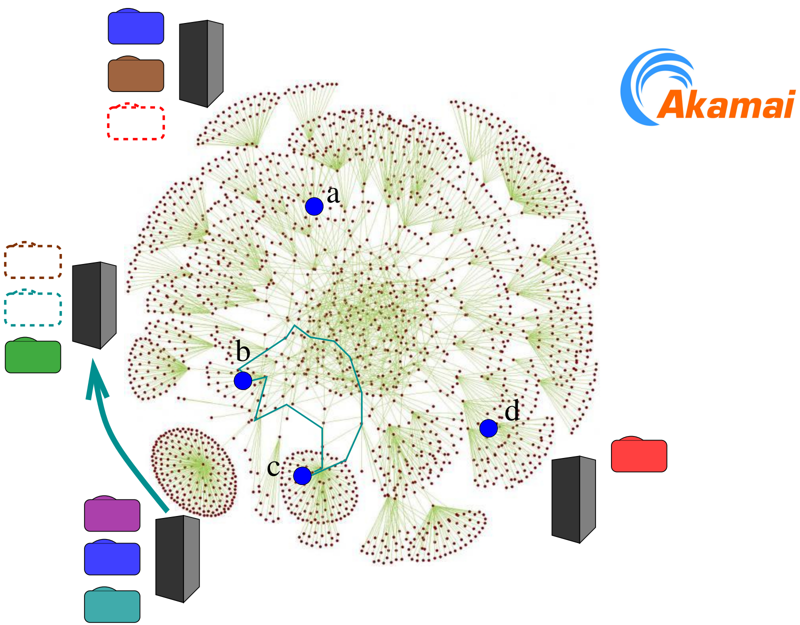


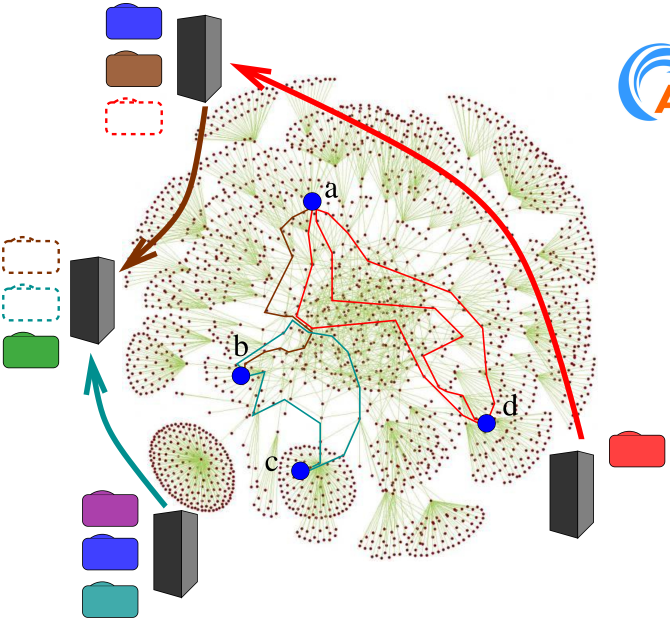
Akamai
a
b
c
d

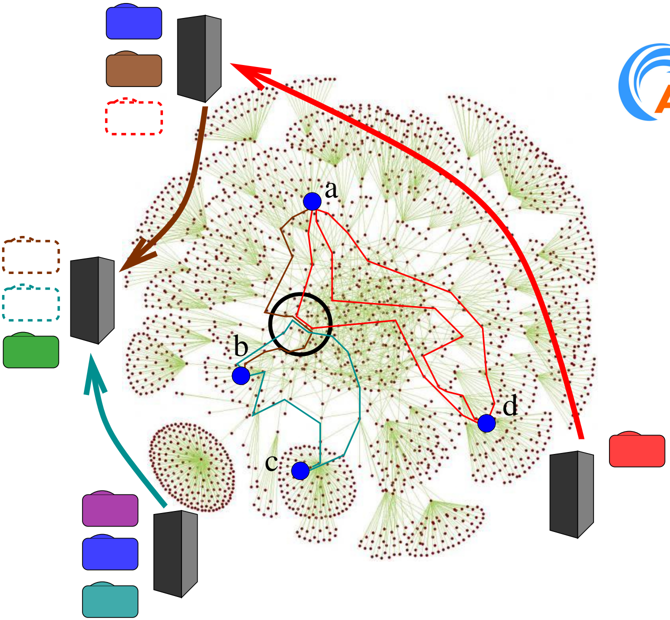

a
b
c
d



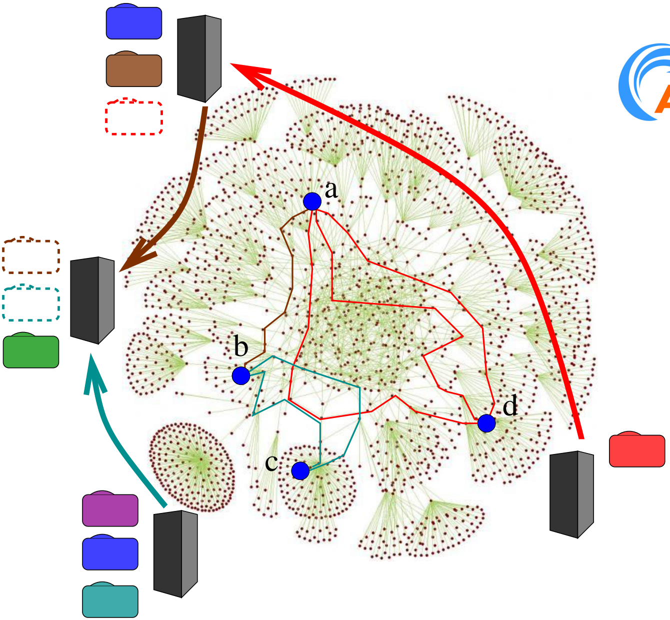
Akamai

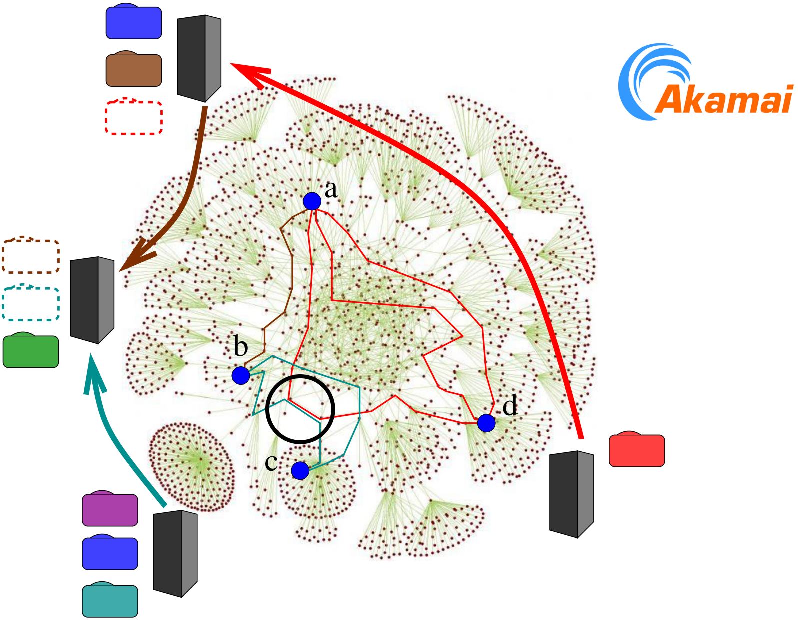

Akamai
a
b
c
d


a
b
c
d

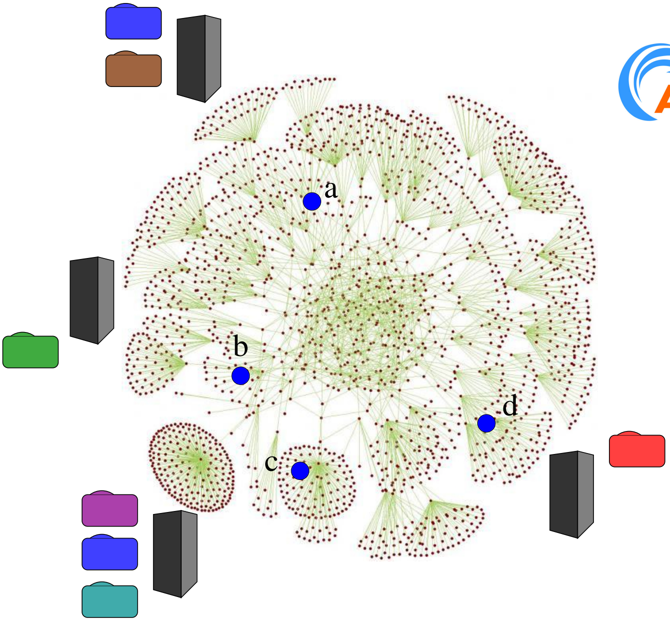

a
b
c
d

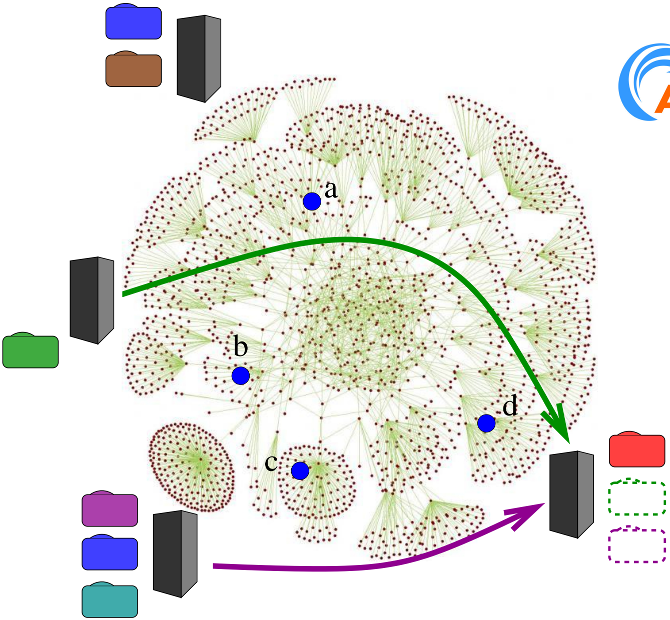
Akamai



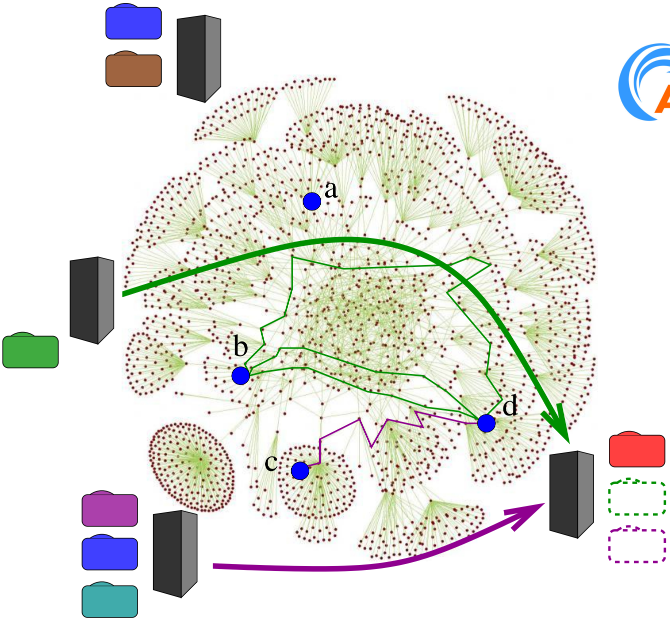

Akamai
a
b
c
d

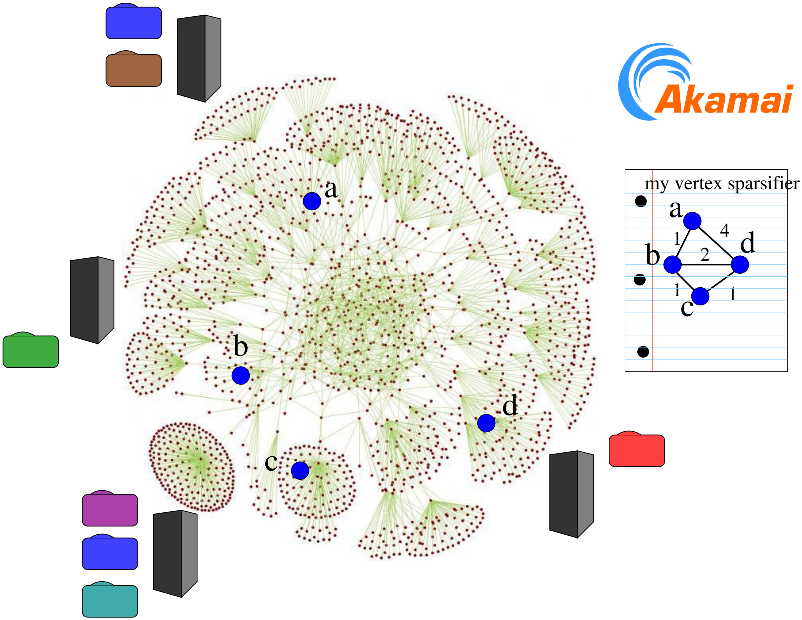
Akamai

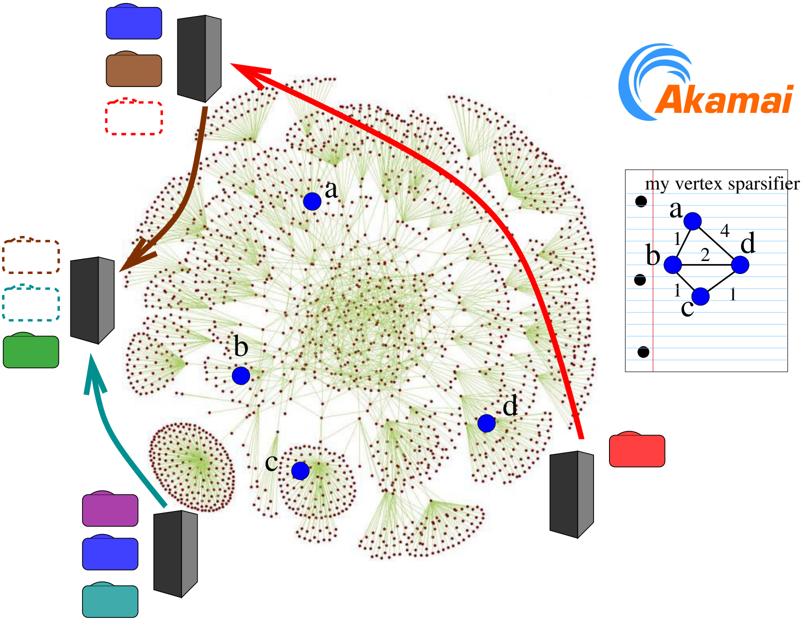


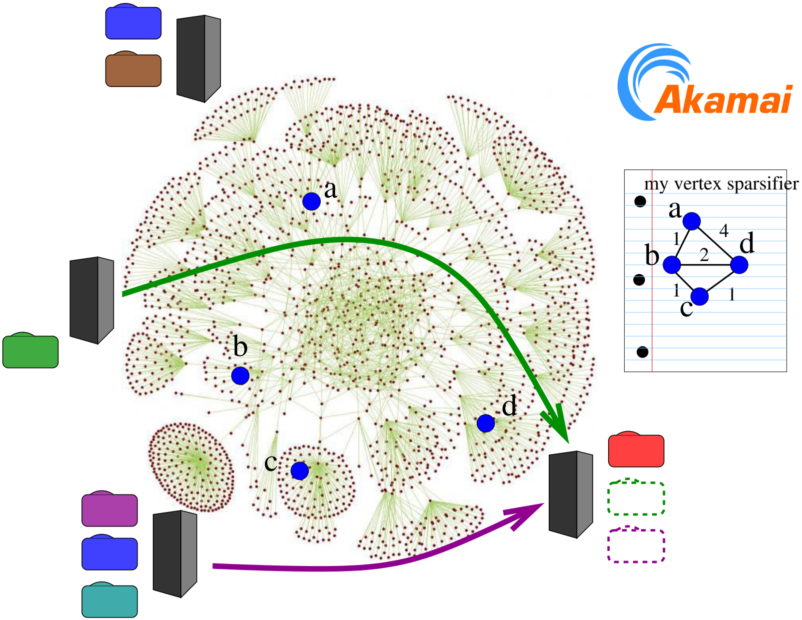


Akamai


a
b
c
d


a
b
c
d
my vertex sparsifier
a
b
c
d
1
4
2
1
1


a
b
c
d
Akamai
my vertex sparsifier
a
b
c
d
1
4
2
1
1



Akamai
my vertex sparsifier
a
b
c
d
1
4
2
1
1

How to Use a Vertex Sparsifier

Question

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

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!

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:

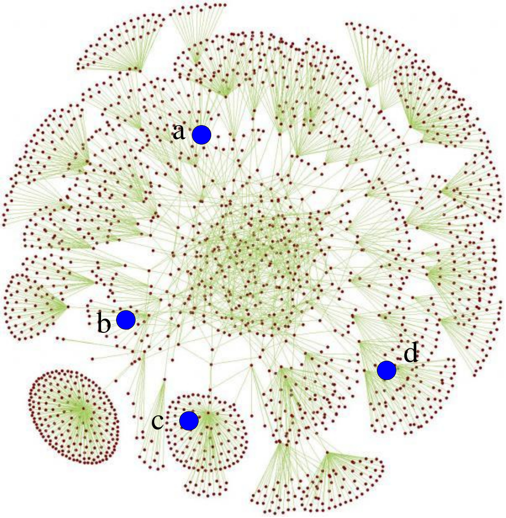
- Save **SPACE** (store a much smaller network)
- Save **TIME** (run algorithms on a much smaller network)
- **UNIFIES** many rounding algorithms for graph partitioning

Outline

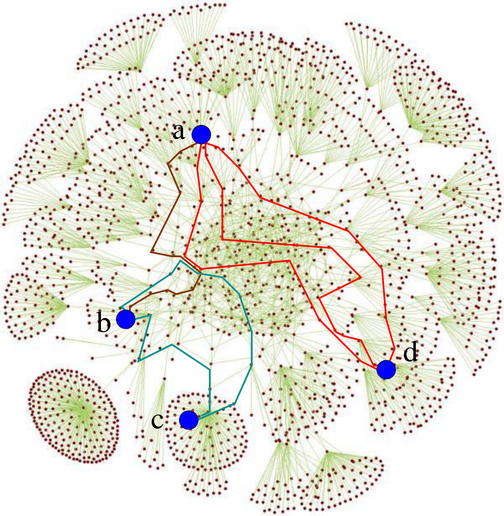
- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials

Outline

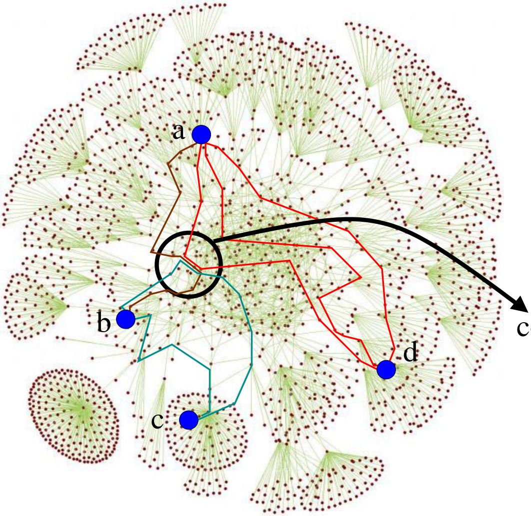
- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials



$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

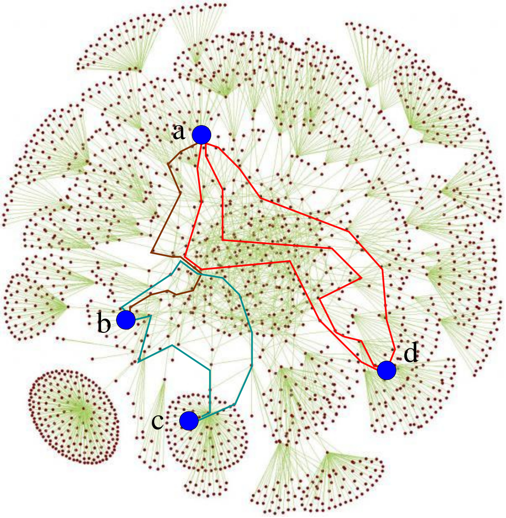


$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{matrix} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{matrix}$$



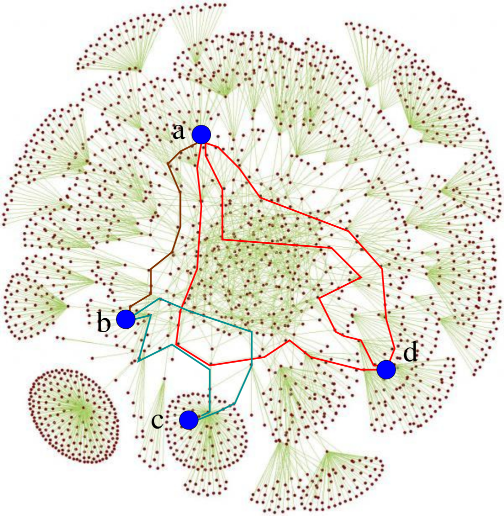
$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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



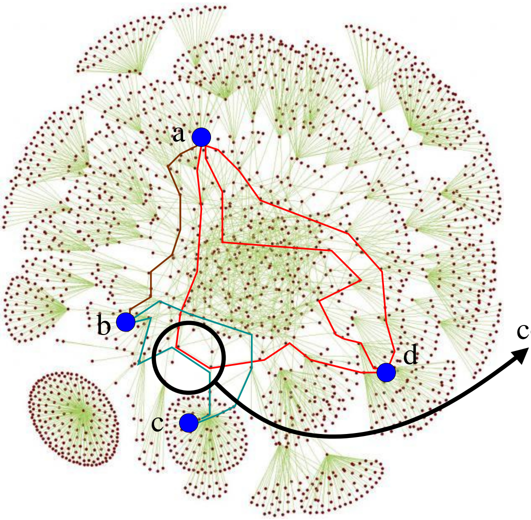
$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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



$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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

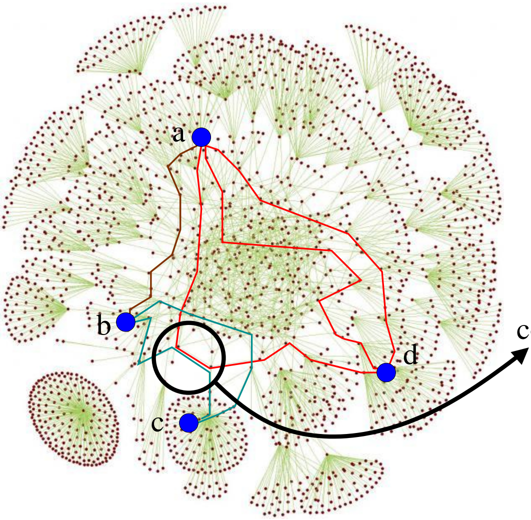


$$\vec{f} = \begin{array}{c} \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \end{array} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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

min congestion $\longleftrightarrow$ max throughput

(min max)



$$\vec{f} = \begin{array}{c} \\ \end{array}\left[\begin{array}{c} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{array}\right] \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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

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?*

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 $\vec{f}$, $cong_G(\vec{f}) \approx cong_H(\vec{f})$

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 $\vec{f}$, $cong_G(\vec{f}) \approx cong_H(\vec{f})$

$$\textbf{Quality}: \left(\max_{\vec{f}} \frac{cong_G(\vec{f})}{cong_H(\vec{f})}\right)\left(\max_{\vec{f}} \frac{cong_H(\vec{f})}{cong_G(\vec{f})}\right)$$

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 $\vec{f}$, $cong_G(\vec{f}) \approx cong_H(\vec{f})$

$$\textbf{Quality}: \left(\max_{\vec{f}} \frac{cong_G(\vec{f})}{cong_H(\vec{f})}\right)\left(\max_{\vec{f}} \frac{cong_H(\vec{f})}{cong_G(\vec{f})}\right)$$

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" $O(\frac{\log |K|}{\log \log |K|})$ in general networks*

Results (Good Vertex Sparsifiers Exist!)

K is the set of terminals (data centers):

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

$$\left\{ \textit{planar, bounded treewidth, padded decomposition property, } \ldots \right\}$$

Results (Good Vertex Sparsifiers Exist!)

K is the set of terminals (data centers):

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

$$\left\{ \textit{planar, bounded treewidth, padded decomposition property, } \ldots \right\}$$

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

$$\left\{ \textit{planar, bounded treewidth, padded decomposition property, } \ldots \right\}$$

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

$$\left\{ \textit{planar, bounded treewidth, padded decomposition property, } \ldots \right\}$$

in quadratic time (approximately a **single** *minimum congestion solve)*

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

(Makarychev, Makarychev): $\tilde{\Omega}(\sqrt{\log |K|})$ *"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

Solving a Sequence of Routing Problems

Observation

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

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:

Solving a Sequence of Routing Problems

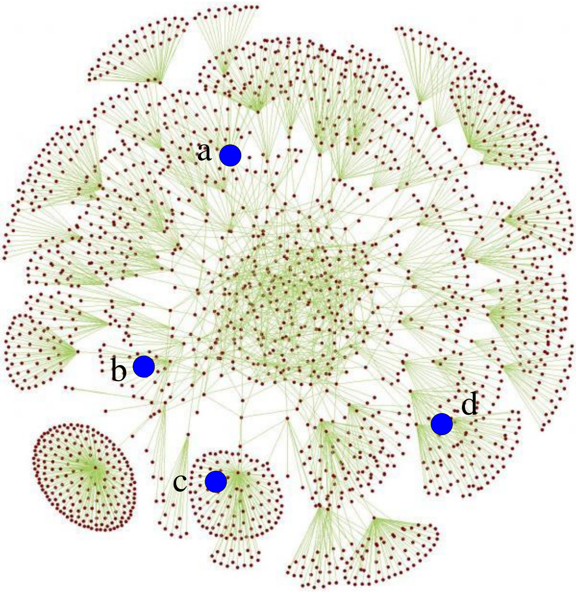
Observation

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

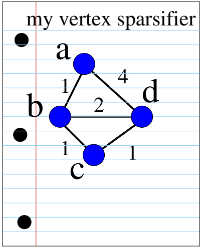
There is a **CANONICAL** mapping of flows in H to flows in G:

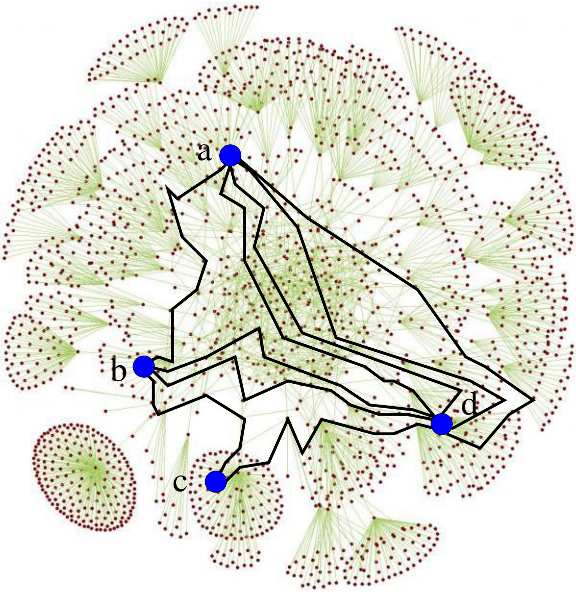
Claim

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

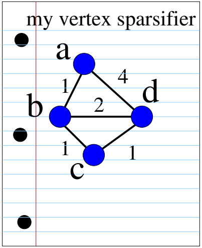


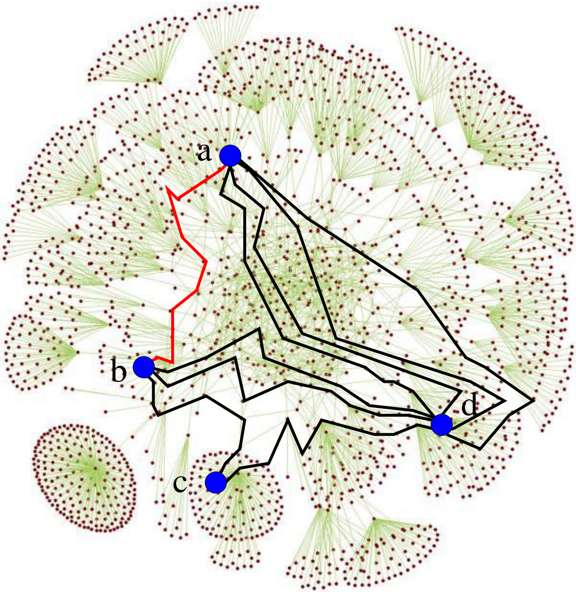
K = {a,b,c,d}



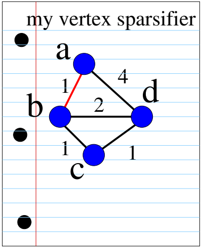


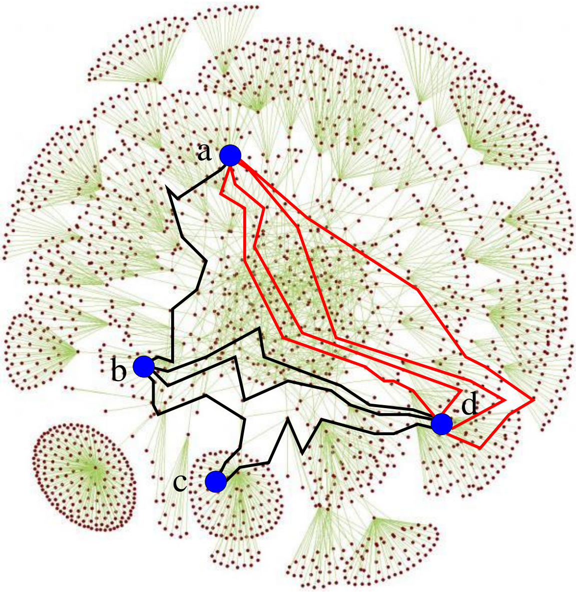
$K = \{a,b,c,d\}$



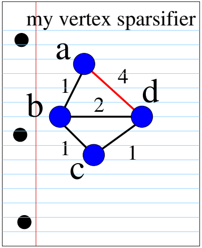


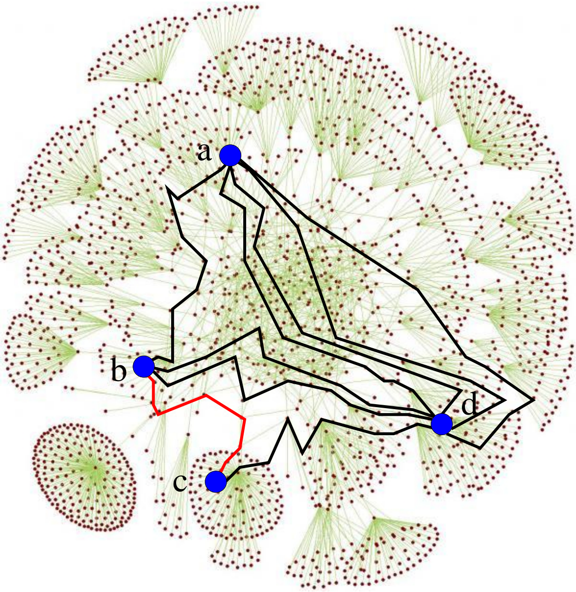
K = {a,b,c,d}



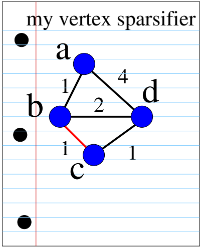


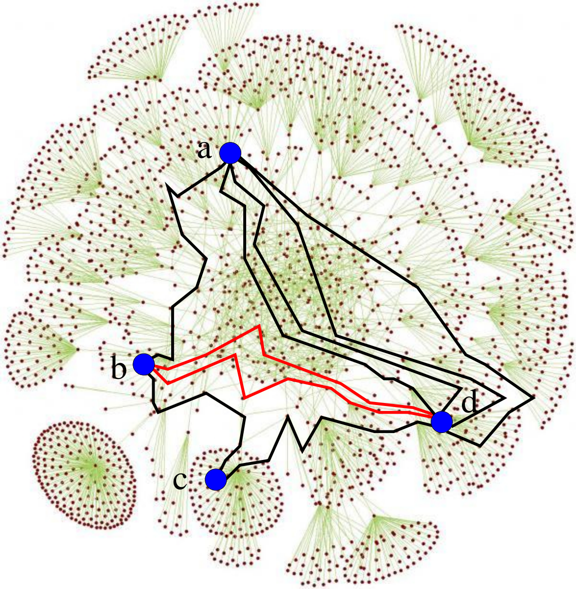
$K = \{a,b,c,d\}$



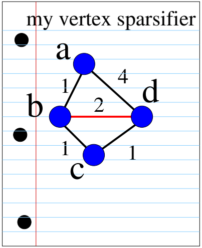


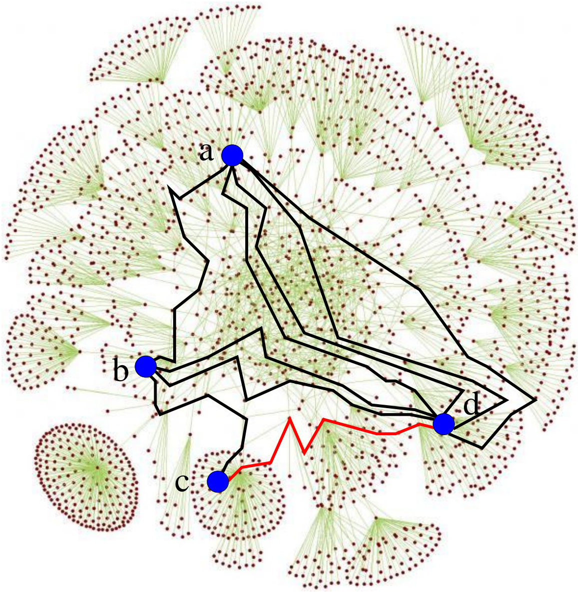
K = {a,b,c,d}



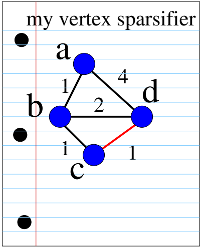


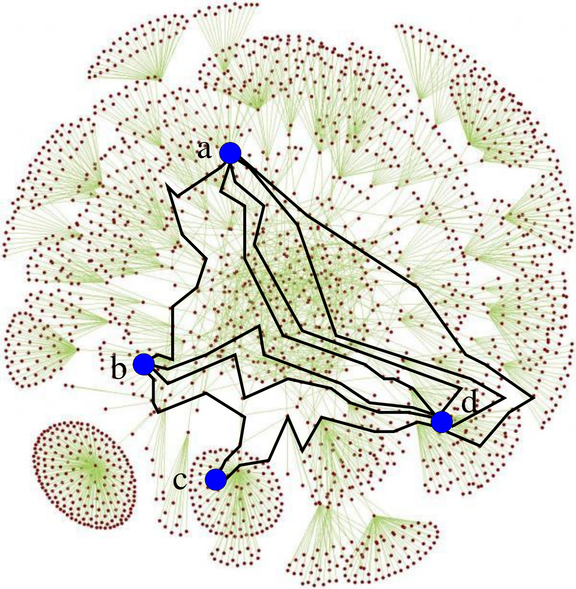
$K = \{a,b,c,d\}$





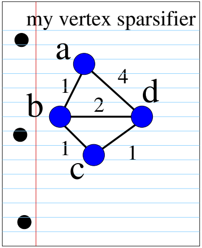
K = {a,b,c,d}

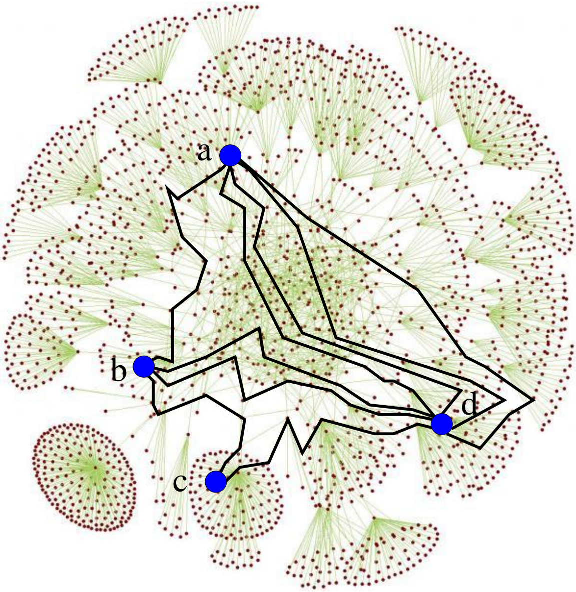




$$\vec{f} = \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

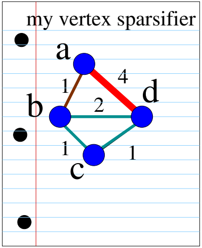
$$K = \{a,b,c,d\}$$

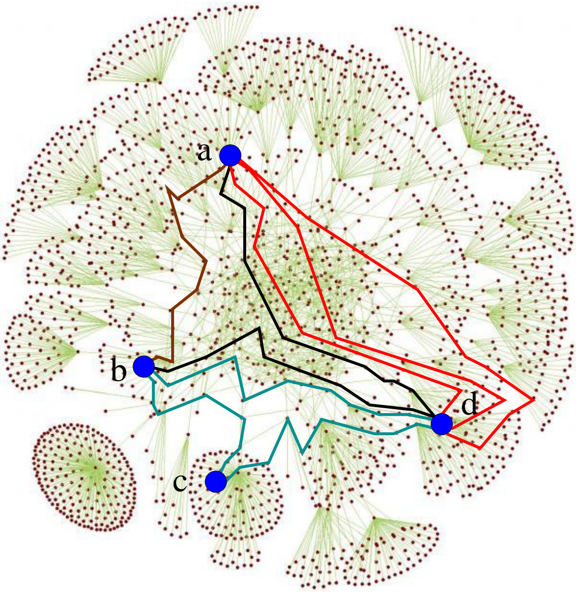




$$\vec{f} = \begin{array}{c} \begin{bmatrix} x \\ 0 \\ y \\ z \\ 0 \\ 0 \end{bmatrix} \end{array} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

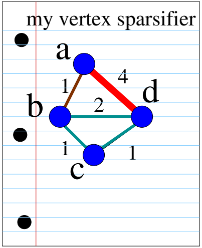
$$K = \{a,b,c,d\}$$





$$\vec{f} = \begin{array}{c} \left[\begin{array}{c} \color{brown}{x} \\ 0 \\ \color{red}{y} \\ \color{teal}{z} \\ 0 \\ 0 \end{array}\right] \end{array} \begin{array}{l} a \longleftrightarrow b \\ a \longleftrightarrow c \\ a \longleftrightarrow d \\ b \longleftrightarrow c \\ b \longleftrightarrow d \\ c \longleftrightarrow d \end{array}$$

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



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*

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*

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*

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*

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*

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 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*

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

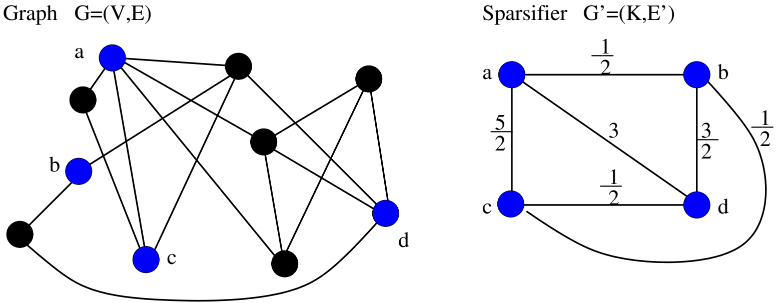
Outline

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

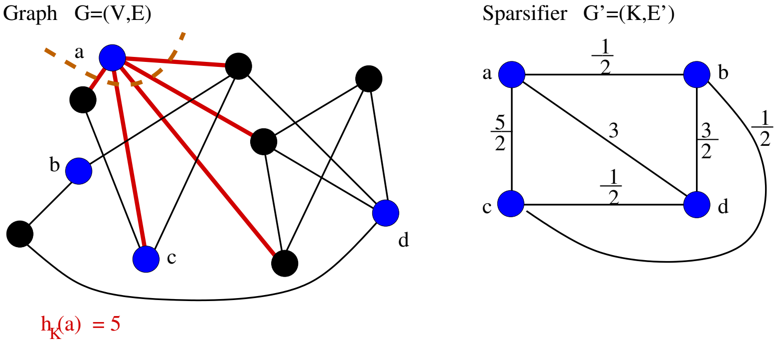
Outline

- 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

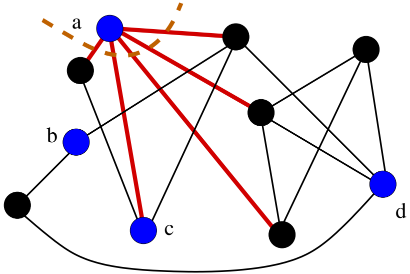


General Approach: Cut Sparsifiers



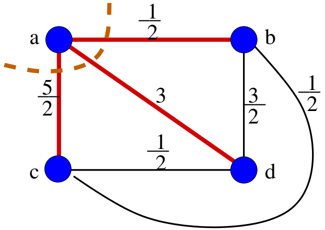
General Approach: Cut Sparsifiers

Graph $G=(V,E)$



$h_K(a) = 5$

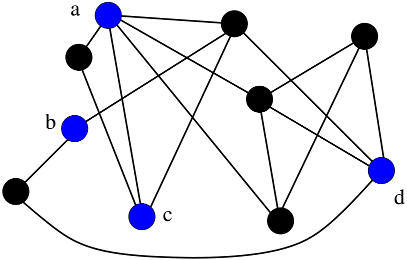
Sparsifier $G'=(K,E')$



$h'(a) = 6$

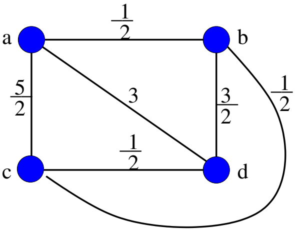
General Approach: Cut Sparsifiers

Graph $G=(V,E)$



$h_K(a) = 5$

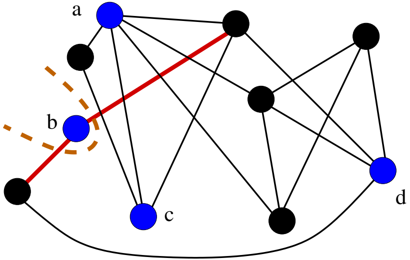
Sparsifier $G'=(K,E')$



$h'(a) = 6$

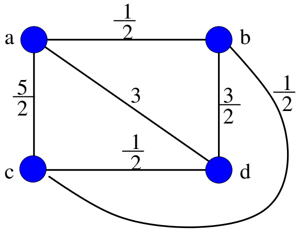
General Approach: Cut Sparsifiers

Graph G=(V,E)



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

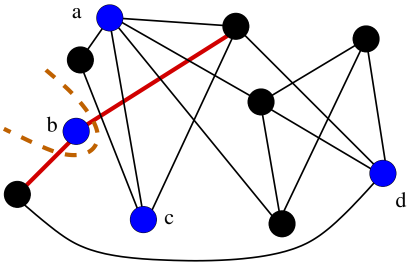
Sparsifier G'=(K,E')



$h'(a) = 6$

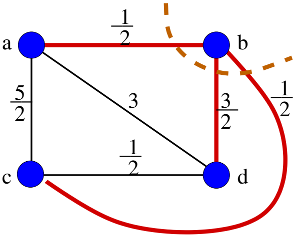
General Approach: Cut Sparsifiers

Graph G=(V,E)



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

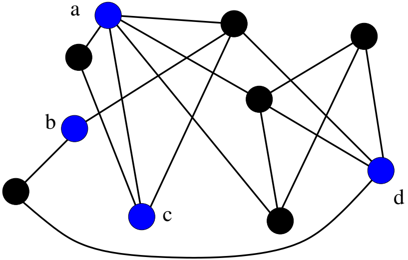
Sparsifier G'=(K,E')



$h'(a) = 6$
$h'(b) = 2.5$

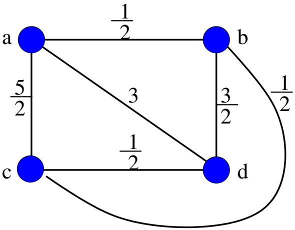
General Approach: Cut Sparsifiers

Graph G=(V,E)



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

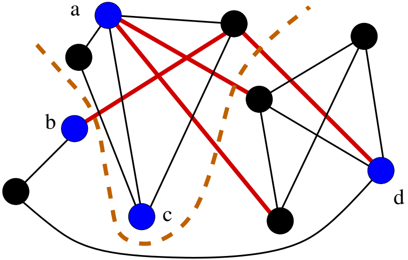
Sparsifier G'=(K,E')



$h'(a) = 6$
$h'(b) = 2.5$

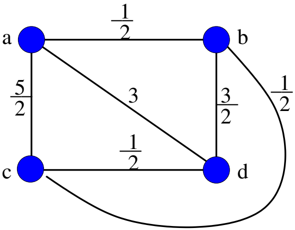
General Approach: Cut Sparsifiers

Graph $G=(V,E)$



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

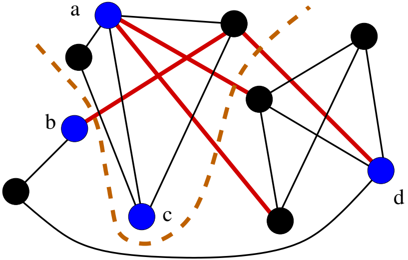
Sparsifier $G'=(K,E')$



$h'(a) = 6$
$h'(b) = 2.5$

General Approach: Cut Sparsifiers

Graph G=(V,E)

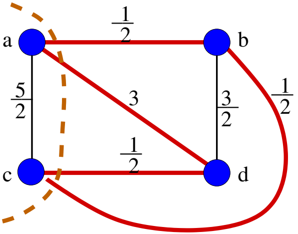


$h_K(a) = 5$

$h_K(b) = 2$

$h_K(ac) = 4$

Sparsifier G'=(K,E')



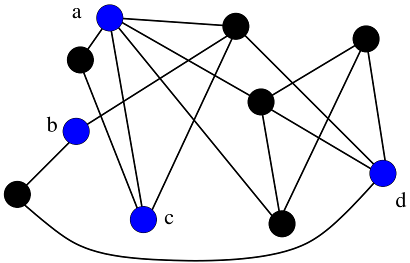
h'(a) = 6

h'(b) = 2.5

h'(ac) = 4.5

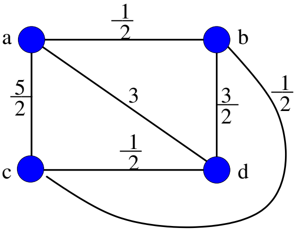
General Approach: Cut Sparsifiers

Graph $G=(V,E)$



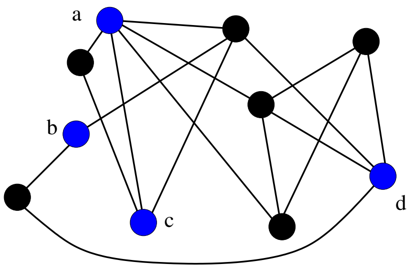
$h_K(a) = 5$ $\quad h_K(d) = 4$ $\quad h_K(ad) = 5$
$h_K(b) = 2$ $\quad h_K(ab) = 7$
$h_K(c) = 3$ $\quad h_K(ac) = 4$

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

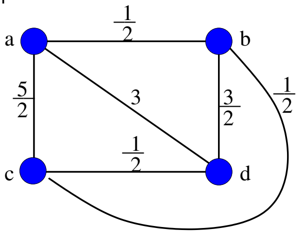


$h'(a) = 6$ $\quad h'(d) = 5$ $\quad h'(ad) = 5$
$h'(b) = 2.5$ $\quad h'(ab) = 7.5$
$h'(c) = 3.5$ $\quad h'(ac) = 4.5$

General Approach: Cut Sparsifiers



$h_K(a) = 5$ $h_K(d) = 4$ $h_K(ad) = 5$
$h_K(b) = 2$ $h_K(ab) = 7$
$h_K(c) = 3$ $h_K(ac) = 4$

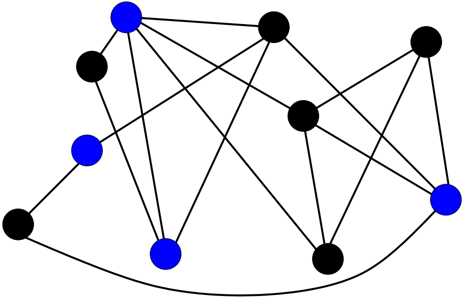


$h'(a) = 6$ $h'(d) = 5$ $h'(ad) = 5$
$h'(b) = 2.5$ $h'(ab) = 7.5$
$h'(c) = 3.5$ $h'(ac) = 4.5$

A Useful Primitive

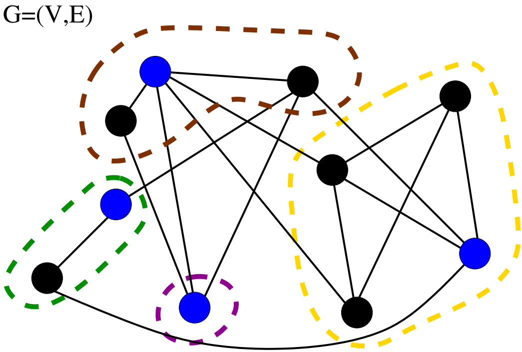
Definition

G=(V,E)



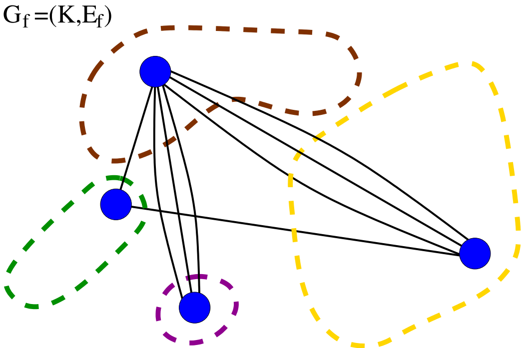
A Useful Primitive

Definition



A Useful Primitive

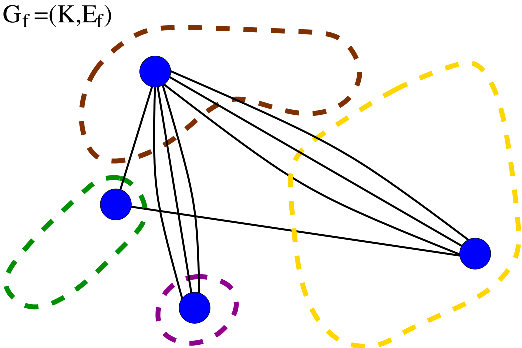
Definition



A Useful Primitive

Definition

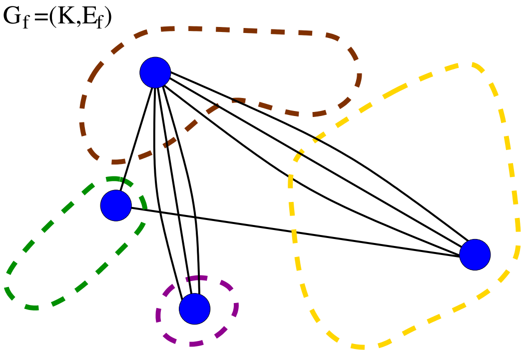
Let $f : V \to K$, is a 0-extension if for all $a \in K$, $f(a) = a$.



A Useful Primitive

Lemma

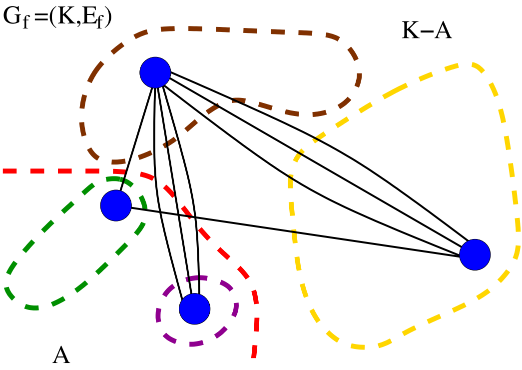
G_f is a Cut Sparsifier



A Useful Primitive

Lemma

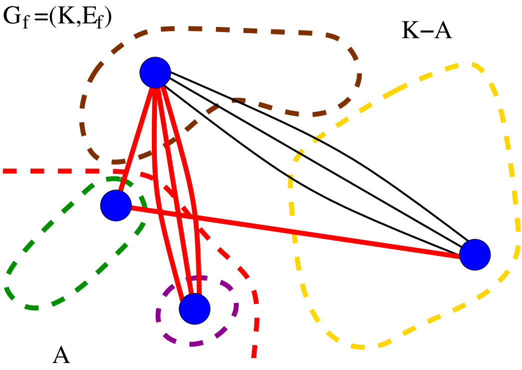
G_f is a Cut Sparsifier



A Useful Primitive

Lemma

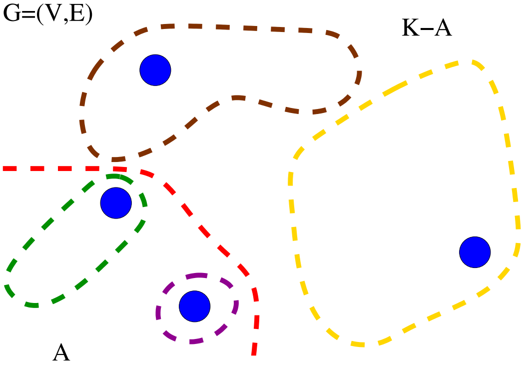
G_f is a Cut Sparsifier



A Useful Primitive

Lemma

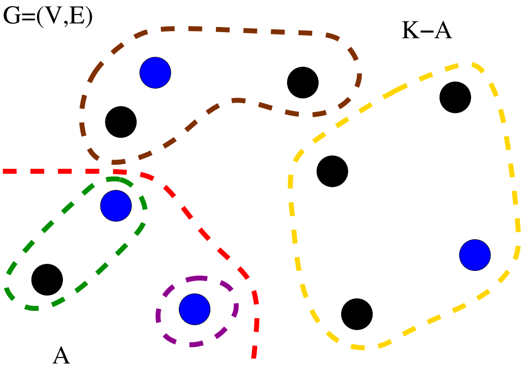
G_f is a Cut Sparsifier



A Useful Primitive

Lemma

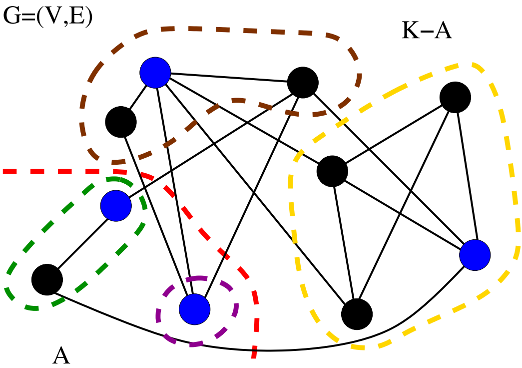
G_f is a Cut Sparsifier



A Useful Primitive

Lemma

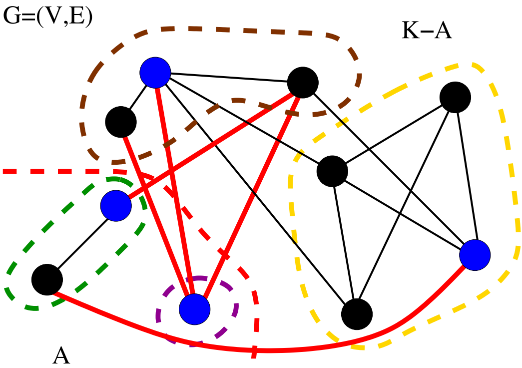
G_f is a Cut Sparsifier



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

Outline

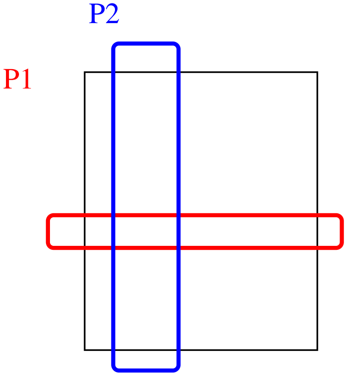
- Introduction
  - Minimum Congestion Routing
  - Application: Routing
  - Application: Graph Partitioning
- Vertex Sparsification
  - Definitions
  - Zero-Sum Game
- Graph Partitioning, Revisited
- Learning via Polynomials

Proof Outline

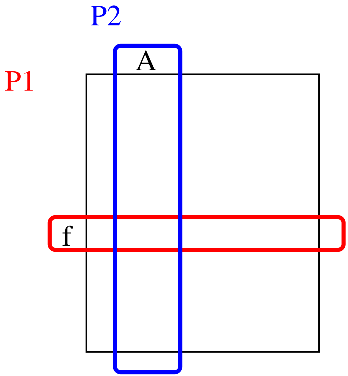
Proof Outline

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

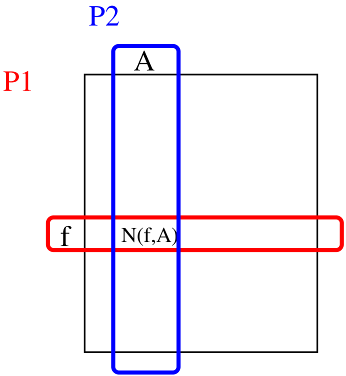
The Extension-Cut Game



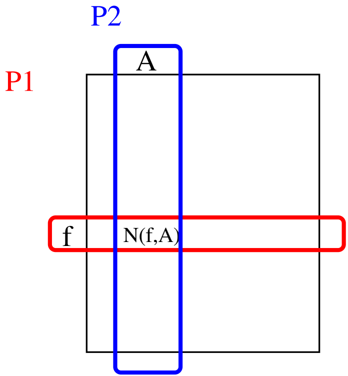
The Extension-Cut Game

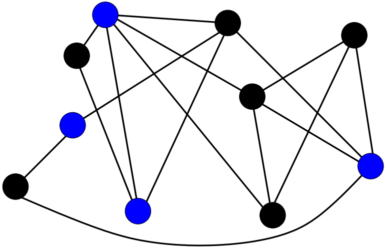


The Extension-Cut Game

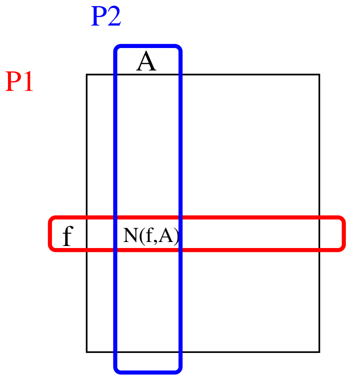


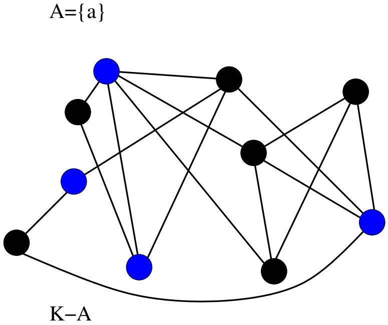
The Extension-Cut Game



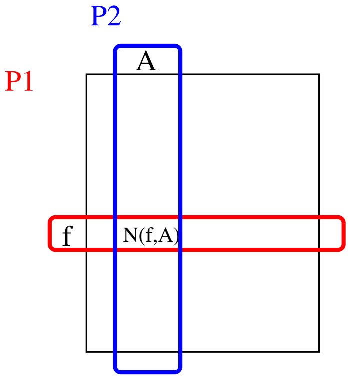


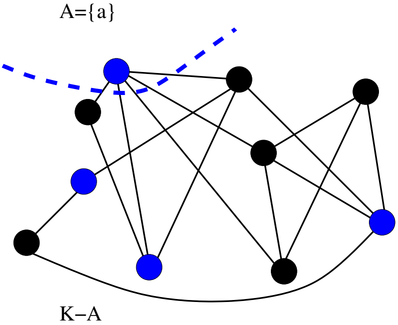
The Extension-Cut Game



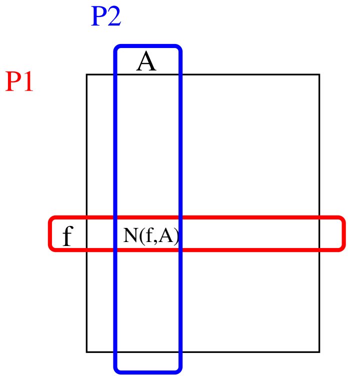


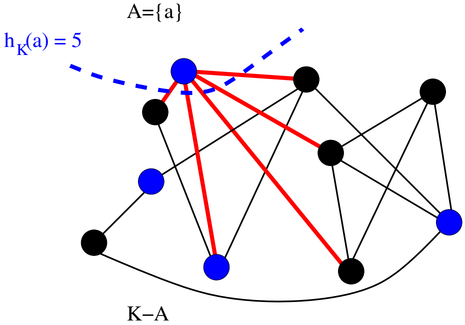
The Extension-Cut Game



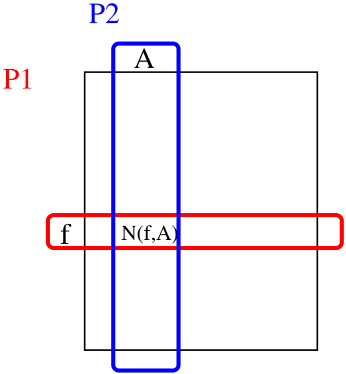


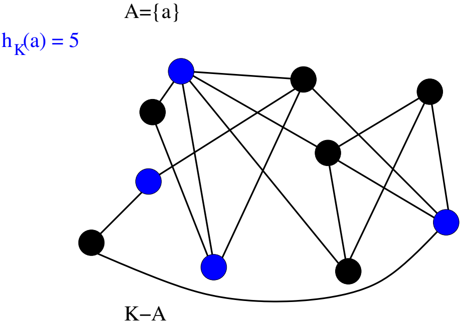
The Extension-Cut Game



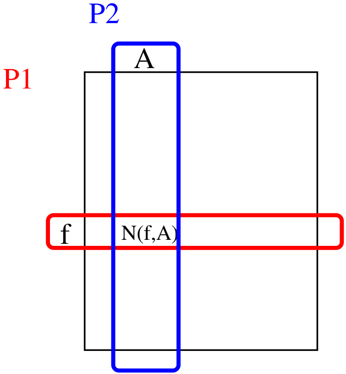


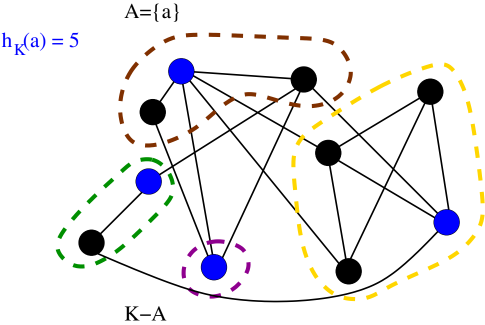
The Extension-Cut Game



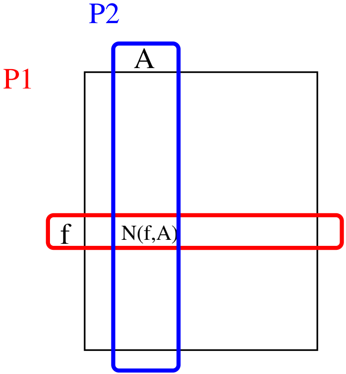


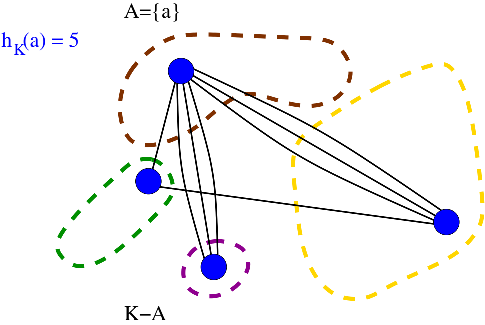
The Extension-Cut Game



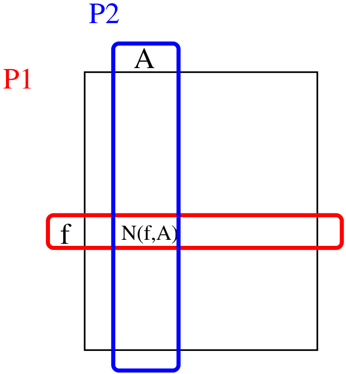


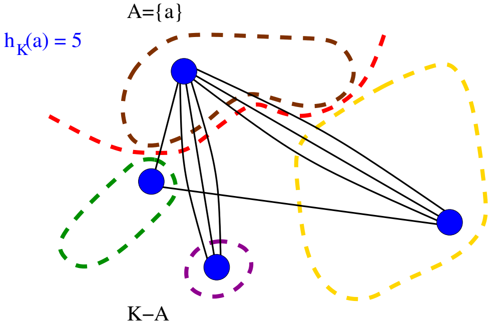
The Extension-Cut Game



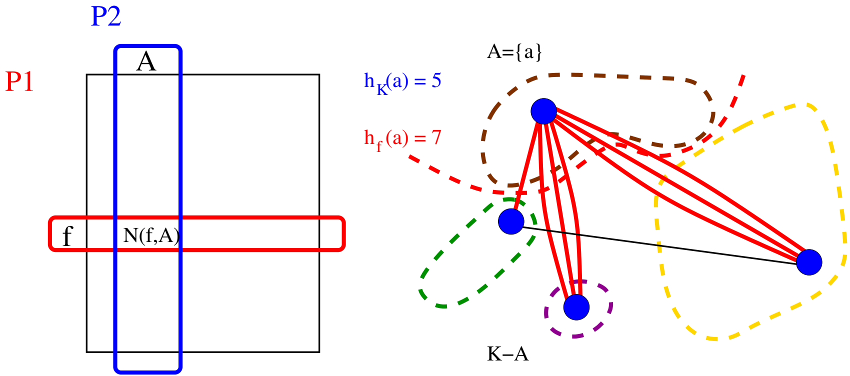


The Extension-Cut Game

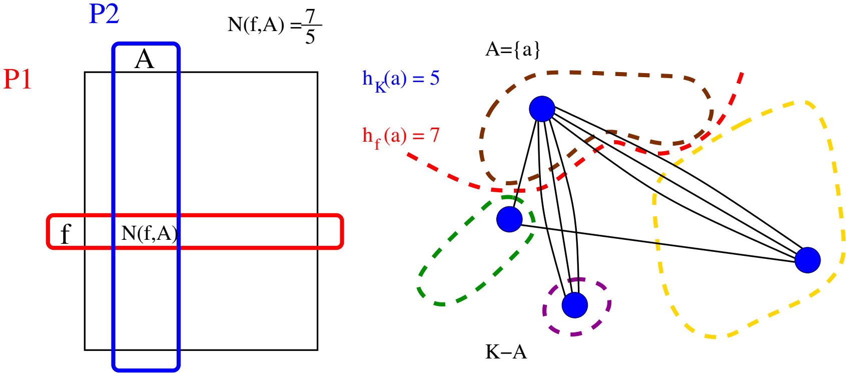




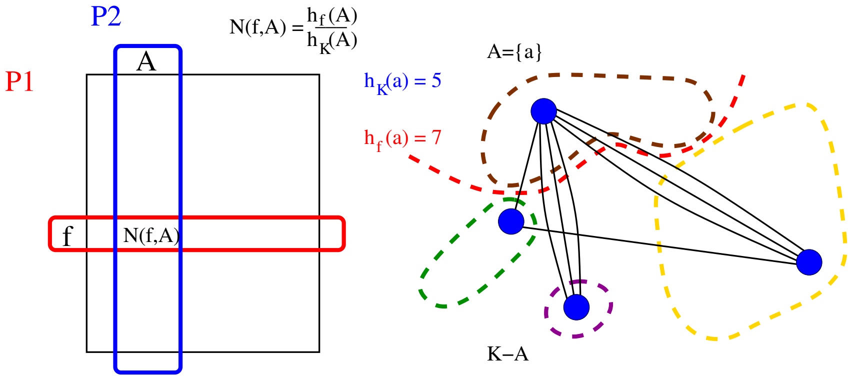
The Extension-Cut Game



The Extension-Cut Game



The Extension-Cut Game



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

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

Question

For every distribution λ on $A \subset K$, is there a **good** *response f for the extension player?*

Proof Outline

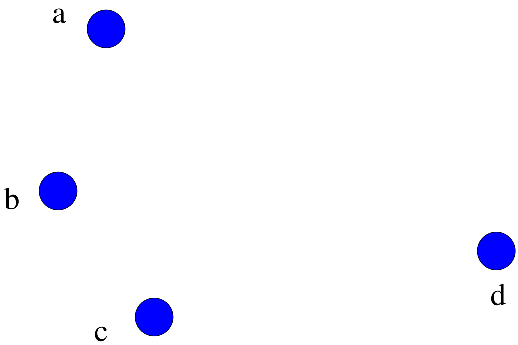
1. Define a **Zero-Sum Game**

Proof Outline

1. Define a **Zero-Sum Game**
2. The **Best Response** is a 0-Extension Problem

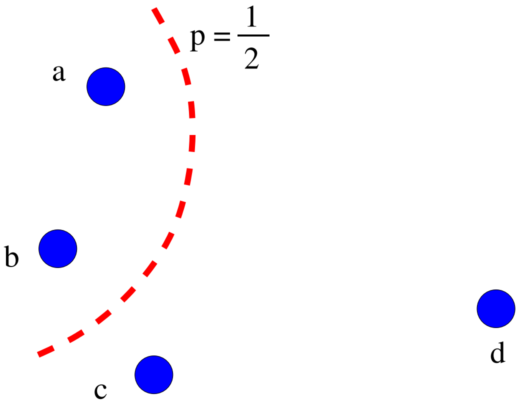
Best Response?

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



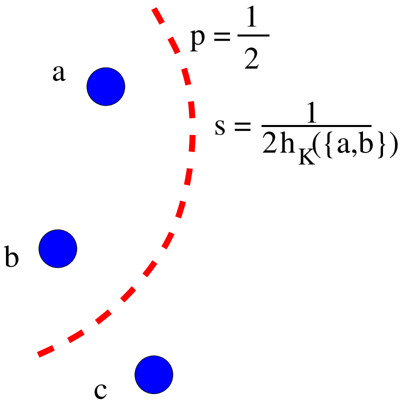
Best Response?

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



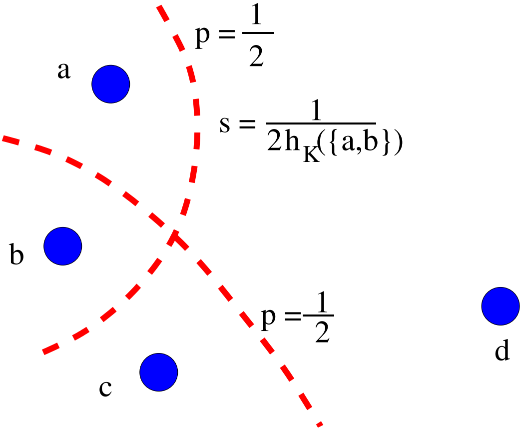
Best Response?

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



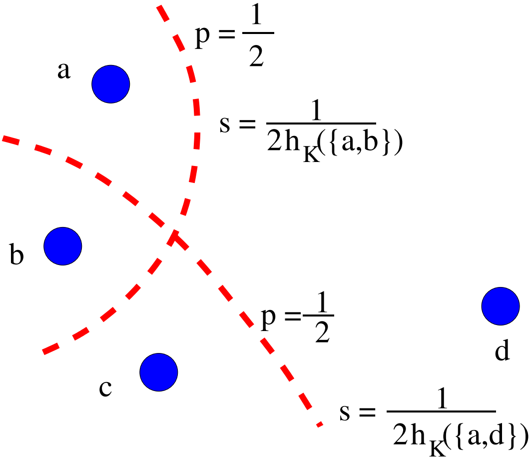
Best Response?

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



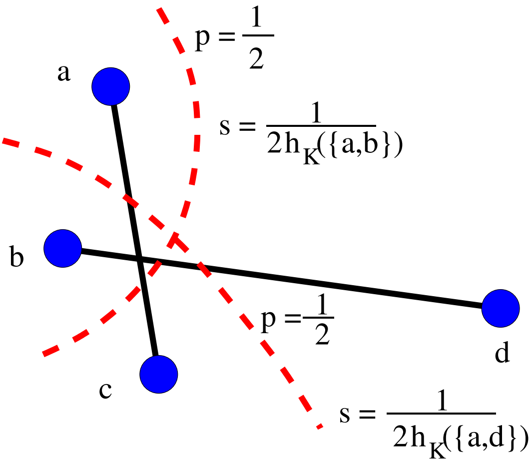
Best Response?

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



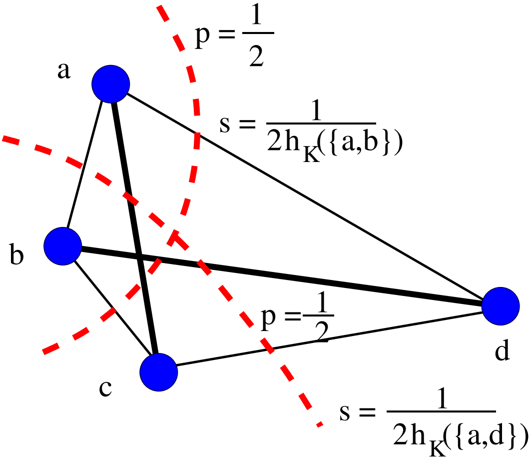
Best Response?

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



Best Response?

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



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

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

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

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)?

Fractional Graph Partitioning Problems

Definition

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

$$\begin{array}{rl} \min & \sum_{(u,v)\in E} c(u,v)d(u,v) \\ \text{s.t.} & \\ & d : V \times V \to \Re^{+} \text{ is a semi-metric} \\ & \ldots \end{array}$$

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):

$$\begin{array}{ll} \min & \sum_{(u,v)\in E} c(u,v)d(u,v) \\ \text{s.t.} & \\ & d : V \times V \to \Re^+ \text{ is a semi-metric} \\ & f(d\big|_K) \geq 1 \end{array}$$

Examples

Consider the (standard) fractional relaxations for:

Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:**

   Goal: Separate all pairs of demands, cutting few edges

Examples

Consider the (standard) fractional relaxations for:

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

Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:** $f(d\Big|_K) = \min_i d(s_i, t_i)$

   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:** $f(d\big|_K) = \min_i d(s_i, t_i)$
   Goal: Separate all pairs of demands, cutting few edges
2. **Sparsest Cut:** $f(d\big|_K) = \sum_i dem(i) d(s_i, t_i)$
   Goal: Find a cut with small ratio

Examples

Consider the (standard) fractional relaxations for:

1. **Multi-Cut:** $f(d\big|_K) = \min_i d(s_i, t_i)$
   Goal: Separate all pairs of demands, cutting few edges
2. **Sparsest Cut:** $f(d\big|_K) = \sum_i dem(i)d(s_i, t_i)$
   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:** $f(d\Big|_K) = \min_i d(s_i, t_i)$
   Goal: Separate all pairs of demands, cutting few edges
2. **Sparsest Cut:** $f(d\Big|_K) = \sum_i dem(i) d(s_i, t_i)$
   Goal: Find a cut with small ratio
3. **Requirement Cut:** $f(d\Big|_K) = \min_i \frac{\mathrm{MST}_{(R_i)}}{p_i}$
   Goal: Separate all sets R_i into at least p_i components, cutting few edges

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*

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