On Hardness of Approximation

Selected Topics in Algorthms

ΑΛΜΑ, ΣΗΜΜΥ

Outline

1. Reducing from a hard to approximate problem

2. Reducing from an NP-hard problem

General Idea

Proving hardness of α-approximating an optimization problem B:

- Pick any "hard" problem A
- Construct in polytime an instance of B
- Make sure that solution of B gives back in polytime a solution of A
- Make sure that given an α-approximation algorithm for B gives back
  - a Yes or No answer for the instance of A (hard decision problem) or
  - an $f(\alpha)$-approximate solution for A (hard to $f(\alpha)$-approximate problem)

General Idea

Proving hardness of α-approximating an optimization problem B:

- Pick any "hard" problem A
- Construct in polytime an instance of B
- Make sure that solution of B gives back in polytime a solution of A
- Make sure that given an α-approximation algorithm for B gives back
  - a Yes or No answer for the instance of A (hard decision problem) or
  - an $f(\alpha)$-approximate solution for A (hard to $f(\alpha)$-approximate problem)

General Idea

Proving hardness of α-approximating an optimization problem B:

- Pick any "hard" problem A
- Construct in polytime an instance of B
- Make sure that solution of B gives back in polytime a solution of A
- Make sure that given an α-approximation algorithm for B gives back
  - a Yes or No answer for the instance of A (hard decision problem) or
  - an $f(\alpha)$-approximate solution for A (hard to $f(\alpha)$-approximate problem)

General Idea

Proving hardness of α-approximating an optimization problem B:

- Pick any "hard" problem A
- Construct in polytime an instance of B
- Make sure that solution of B gives back in polytime a solution of A
- Make sure that given an α-approximation algorithm for B gives back
  - a Yes or No answer for the instance of A (hard decision problem) or
  - an $f(\alpha)$-approximate solution for A (hard to $f(\alpha)$-approximate problem)

General Idea

Proving hardness of α-approximating an optimization problem B:

- Pick any "hard" problem A
- Construct in polytime an instance of B
- Make sure that solution of B gives back in polytime a solution of A
- Make sure that given an α-approximation algorithm for B gives back
  - a Yes or No answer for the instance of A (hard decision problem) or
  - an $f(\alpha)$-approximate solution for A (hard to $f(\alpha)$-approximate problem)

Outline

1. Reducing from a hard to approximate problem

2. Reducing from an NP-hard problem

Hard to $\frac{7}{8}$-Approximate Problem: MaxE3Sat

MaxE3Sat

input: Set of m clauses with exactly 3 literals

output: Assignment to the variables that maximizes the number of satisfied clauses

Facts:

- simple randomized algorithm returns $\frac{7}{8}m$ clauses satisfied in expectation $\Rightarrow$ optimal solution $k^* \geq \frac{7}{8}m$
- Unless P=NP, there is no $\frac{7}{8} + \epsilon$-approximation algorithm

Hard to $\frac{7}{8}$-Approximate Problem: MaxE3Sat

MaxE3Sat

input: Set of m clauses with exactly 3 literals

output: Assignment to the variables that maximizes the number of satisfied clauses

Facts:

- simple randomized algorithm returns $\frac{7}{8}m$ clauses satisfied in expectation $\Rightarrow$ optimal solution $k^* \geq \frac{7}{8}m$
- Unless P=NP, there is no $\frac{7}{8} + \epsilon$-approximation algorithm

to prove NP-Hard to Approximate Problem: Max2Sat

Max2Sat

input: Set of m clauses with at most 2 literals

output: Assignment to the variables that maximizes the number of satisfied clauses

to prove NP-Hard to Approximate Problem: Max2Sat

Max2Sat

input: Set of m clauses with at most 2 literals

output: Assignment to the variables that maximizes the number of satisfied clauses

Reducing MaxE3Sat to Max2Sat

The reduction

- Let A be an instance of MaxE3Sat with m clauses
- For each clause say $c_i = x_{i1} \vee x_{i2} \vee x_{i3}$ create the following 10 clauses:
  - x_{i1}, x_{i2}, x_{i3}
  - $\overline{x_{i1}} \vee \overline{x_{i2}}, \overline{x_{i2}} \vee \overline{x_{i3}}, \overline{x_{i3}} \vee \overline{x_{i1}}$
  - y_i, where y_i is a new variable corresponding to clause c_i
  - $x_{i1} \vee \overline{y_i}, x_{i2} \vee \overline{y_i}, x_{i3} \vee \overline{y_i}$
- Combine all clauses to create an instance B of Max2Sat

Facts:

- If c_i is satisfied then 7 out of 10 clauses can be satisfied
- If c_i is not satisfied then at most 6 out of 10 clauses can be satisfied
- If k^* is the optimal number of satisfied clauses for A then $7k^* + 6(m - k^*)$ is the optimal number of satisfied clauses for B

Reducing MaxE3Sat to Max2Sat

The reduction

- Let A be an instance of MaxE3Sat with m clauses
- For each clause say $c_i = x_{i1} \vee x_{i2} \vee x_{i3}$ create the following 10 clauses:
  - x_{i1}, x_{i2}, x_{i3}
  - $\overline{x_{i1}} \vee \overline{x_{i2}}$, $\overline{x_{i2}} \vee \overline{x_{i3}}$, $\overline{x_{i3}} \vee \overline{x_{i1}}$
  - y_i, where y_i is a new variable corresponding to clause c_i
  - $x_{i1} \vee \overline{y_i}$, $x_{i2} \vee \overline{y_i}$, $x_{i3} \vee \overline{y_i}$
- Combine all clauses to create an instance B of Max2Sat

Facts:

- If c_i is satisfied then 7 out of 10 clauses can be satisfied
- If c_i is not satisfied then at most 6 out of 10 clauses can be satisfied
- If k^* is the optimal number of satisfied clauses for A then $7k^* + 6(m - k^*)$ is the optimal number of satisfied clauses for B

Reducing MaxE3Sat to Max2Sat

The reduction

- Let A be an instance of MaxE3Sat with m clauses
- For each clause say $c_i = x_{i1} \vee x_{i2} \vee x_{i3}$ create the following 10 clauses:
  - x_{i1}, x_{i2}, x_{i3}
  - $\overline{x_{i1}} \vee \overline{x_{i2}}, \overline{x_{i2}} \vee \overline{x_{i3}}, \overline{x_{i3}} \vee \overline{x_{i1}}$
  - y_i, where y_i is a new variable corresponding to clause c_i
  - $x_{i1} \vee \overline{y_i}, x_{i2} \vee \overline{y_i}, x_{i3} \vee \overline{y_i}$
- Combine all clauses to create an instance B of Max2Sat

Facts:

- If c_i is satisfied then 7 out of 10 clauses can be satisfied
- If c_i is not satisfied then at most 6 out of 10 clauses can be satisfied
- If k^* is the optimal number of satisfied clauses for A then $7k^* + 6(m - k^*)$ is the optimal number of satisfied clauses for B

Reducing MaxE3Sat to Max2Sat

The reduction

- Let A be an instance of MaxE3Sat with m clauses
- For each clause say $c_i = x_{i1} \vee x_{i2} \vee x_{i3}$ create the following 10 clauses:
    - x_{i1}, x_{i2}, x_{i3}
    - $\overline{x_{i1}} \vee \overline{x_{i2}}, \overline{x_{i2}} \vee \overline{x_{i3}}, \overline{x_{i3}} \vee \overline{x_{i1}}$
    - y_i, where y_i is a new variable corresponding to clause c_i
    - $x_{i1} \vee \overline{y_i}, x_{i2} \vee \overline{y_i}, x_{i3} \vee \overline{y_i}$
- Combine all clauses to create an instance B of Max2Sat

Facts:

- If c_i is satisfied then 7 out of 10 clauses can be satisfied
- If c_i is not satisfied then at most 6 out of 10 clauses can be satisfied
- If k^* is the optimal number of satisfied clauses for A then $7k^* + 6(m - k^*)$ is the optimal number of satisfied clauses for B

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say Alg

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using Alg to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall \alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall \alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Using an Approximation Algorithm for Max2Sat

To show: α-approximation for B gives $f(\alpha)$-approximation for A.

Assume we have an α-approximation algotithm for Max2Sat, say *Alg*

- From instance A of MaxE3Sat construct instance B of Max2Sat
- Solve B using *Alg* to take an approximate solution of B that satisfes say X clauses.
- Call $\overline{k}$ the number of corresponding satisfied clauses in A
- It should be $\overline{k} \geq X - 6m$ (else $7\overline{k} + 6(m - \overline{k}) < X$)
- $\overline{k} \geq \alpha(7k^* + 6(m - k^*)) - 6m = \alpha k^* + 6(a-1)m \geq \alpha k^* + 6(a-1)\frac{8}{7}k^*$
- $\overline{k} \geq (\frac{55}{7}\alpha - \frac{48}{7})k^*$

$\frac{7}{8}$ hardness of approximation for A implies α hardness of approximation for B, $\forall\alpha : \frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8} \Rightarrow \forall\alpha : \alpha > \frac{433}{440}$

Outline

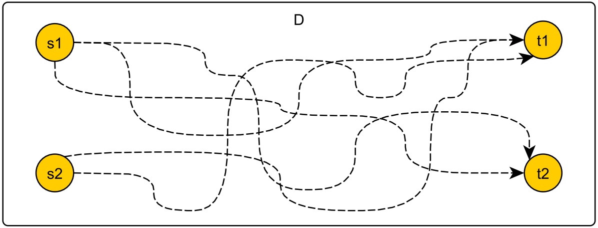
1. Reducing from a hard to approximate problem

2. Reducing from an NP-hard problem

NP Hard Problem: 2DDP

input: A directed Network D and two pairs s_1, t_1 and s_2, t_2.

output: Answer to the question: Are there 2 vertex disjoint paths joining s_1 to t_1 and s_2 to t_2



to Prove NP Hard to Approximate: BestSubnet

Abstract description of BestSubnet:

input: A directed Network G, with source s, target t and traffic rate r

output: Find Subnetwork that minimizes cost at Equilibrium

Approximate version:

output: Find Subnetwork with Equilibrium cost that approximates the Best Subnetwork's cost at equilibrium

to Prove NP Hard to Approximate: BestSubnet

Abstract description of BestSubnet:

input: A directed Network G, with source s, target t and traffic rate r

output: Find Subnetwork that minimizes cost at Equilibrium

Approximate version:

output: Find Subnetwork with Equilibrium cost that approximates the Best Subnetwork's cost at equilibrium

Reducing 2DDP to Approximating Best Subnetwork

Given an instance of 2DDP we construct a "base" network.

- Yes instance $\rightarrow \exists$ subnetwork with worst Equilibrium cost$= r/4$
- No instance $\rightarrow \forall$ subnetwork worst Equilibrium cost $\geq r/3$

Reduction provides a 4/3 gap for Best Subnetwork Problem
or else:

There exists a $(4/3 - \epsilon)$-approximation algorithm

$$\Downarrow$$

In a Yes instance, returned solution cost $\leq (4/3 - \epsilon)r/4 < r/3$

Reducing 2DDP to Approximating Best Subnetwork

Given an instance of 2DDP we construct a "base" network.

- Yes instance $\rightarrow \exists$ subnetwork with worst Equilibrium cost$= r/4$
- No instance $\rightarrow \forall$ subnetwork worst Equilibrium cost $\geq r/3$

Reduction provides a 4/3 gap for Best Subnetwork Problem
or else:

There exists a $(4/3 - \epsilon)$-approximation algorithm

$$\Downarrow$$

In a Yes instance, returned solution cost $\leq (4/3 - \epsilon)r/4 < r/3$

Reducing 2DDP to Approximating Best Subnetwork

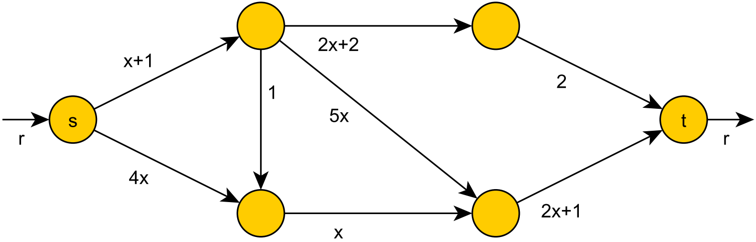
Given an instance of 2DDP we construct a "base" network.

- Yes instance $\rightarrow \exists$ subnetwork with worst Equilibrium cost$= r/4$
- No instance $\rightarrow \forall$ subnetwork worst Equilibrium cost $\geq r/3$

Reduction provides a $4/3$ gap for Best Subnetwork Problem
or else:

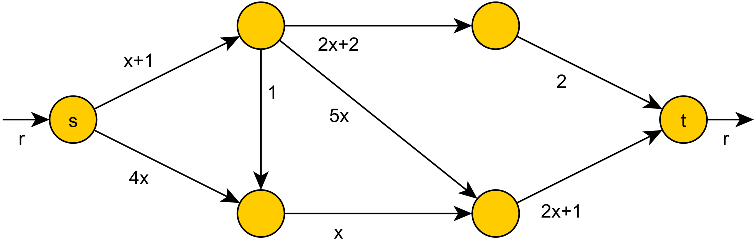
$$\text{There exists a } (4/3 - \epsilon)\text{-approximation algorithm}$$
$$\Downarrow$$
$$\text{In a Yes instance, returned solution cost} \leq (4/3 - \epsilon)r/4 < r/3$$

Routing Instance



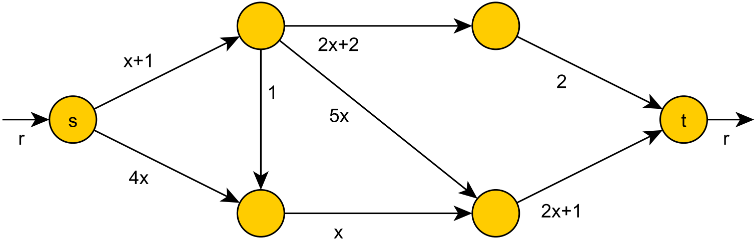
- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

Routing Instance



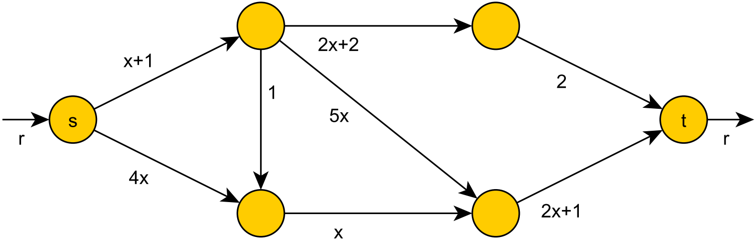
- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

Routing Instance



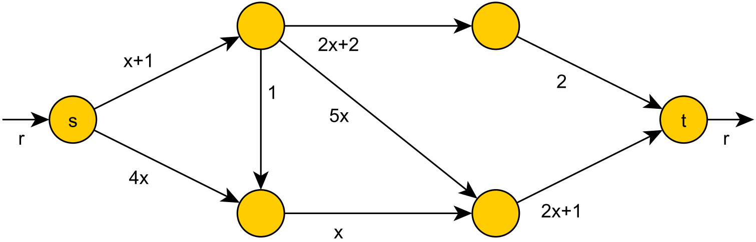
- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

Routing Instance



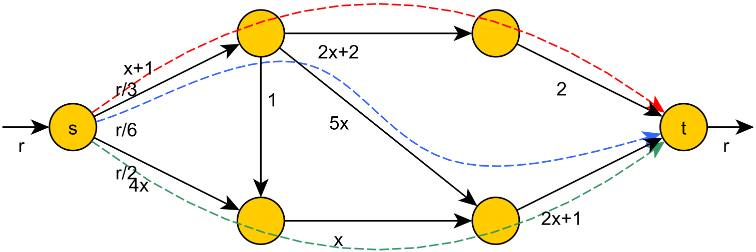
- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

Routing Instance



- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

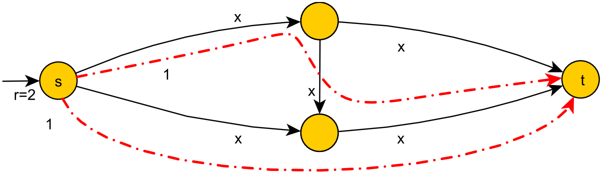
Routing Instance



- Traffic rate r to be routed through paths from s to t.
- Edges are associated with a cost function
- Depending on the routing and the resulting load of each edge,
  - each path has a cost
  - the network itself has an "overall cost"

Our Case: Bottleneck Routing Games

- The cost of each path is its *Bottleneck Cost*: the cost of its most costly edge
- The Overall cost is the *Bottleneck Cost of the Network*: the cost of the most costly edge (under use) on the Network

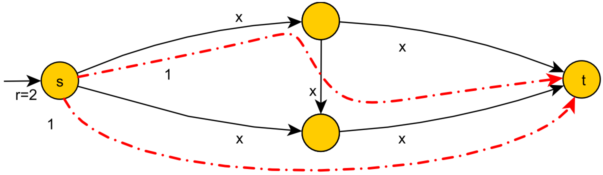


Paths: Upper path costs 1. Middle path costs 2. Lower path costs 2.

Network: The Bottleneck cost of the Network is 2

Our Case: Bottleneck Routing Games

- The cost of each path is its *Bottleneck Cost*: the cost of its most costly edge
- The Overall cost is the *Bottleneck Cost of the Network*: the cost of the most costly edge (under use) on the Network

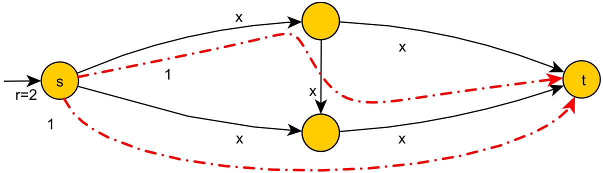


Paths: Upper path costs 1. Middle path costs 2. Lower path costs 2.

Network: The Bottleneck cost of the Network is 2

Our Case: Bottleneck Routing Games

- The cost of each path is its *Bottleneck Cost*: the cost of its most costly edge
- The Overall cost is the *Bottleneck Cost of the Network*: the cost of the most costly edge (under use) on the Network

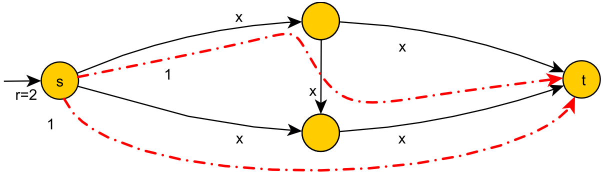


Paths: Upper path costs 1. Middle path costs 2. Lower path costs 2.

Network: The Bottleneck cost of the Network is 2

Our Case: Bottleneck Routing Games

- The cost of each path is its *Bottleneck Cost*: the cost of its most costly edge
- The Overall cost is the *Bottleneck Cost of the Network*: the cost of the most costly edge (under use) on the Network

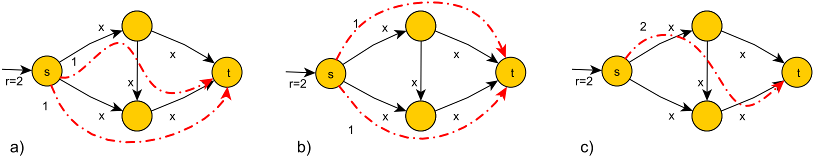


Paths: Upper path costs 1. Middle path costs 2. Lower path costs 2.

Network: The Bottleneck cost of the Network is 2

Equilibria-Nash Flows

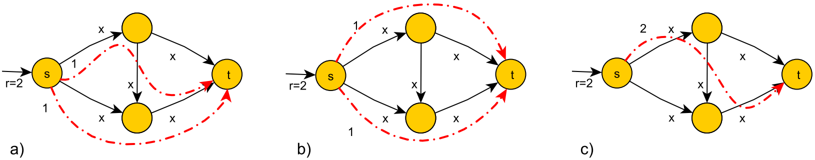
Nash Flow: A flow under which no player wishes to change path



- Not unique: There are optimal, bad and worst Nash flows
- At a Nash flow, edges with cost $\geq BC$ form a cut
- Worst Nash flow: flow is routed through few blocking paths

Equilibria-Nash Flows

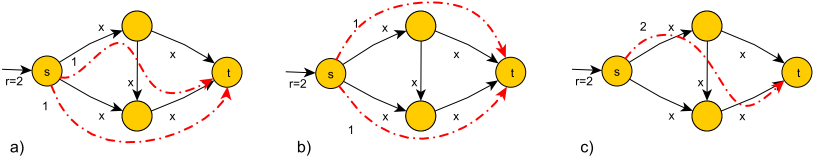
Nash Flow: A flow under which no player wishes to change path



- Not unique: There are optimal, bad and worst Nash flows
- At a Nash flow, edges with cost $\geq BC$ form a cut
- Worst Nash flow: flow is routed through few blocking paths

Equilibria-Nash Flows

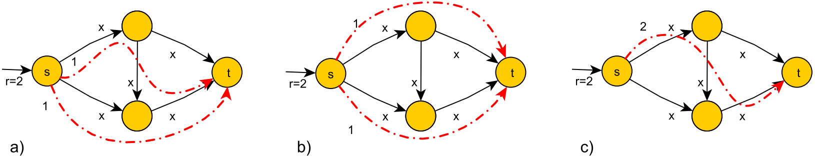
Nash Flow: A flow under which no player wishes to change path



- Not unique: There are optimal, bad and worst Nash flows
- At a Nash flow, edges with cost $\geq BC$ form a cut
- Worst Nash flow: flow is routed through few blocking paths

Equilibria-Nash Flows

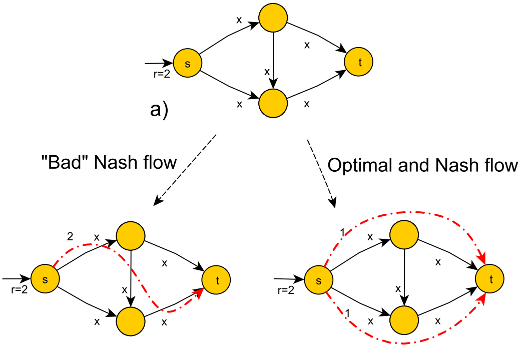
Nash Flow: A flow under which no player wishes to change path



- Not unique: There are optimal, bad and worst Nash flows
- At a Nash flow, edges with cost $\geq BC$ form a cut
- Worst Nash flow: flow is routed through few blocking paths

Braess Paradox

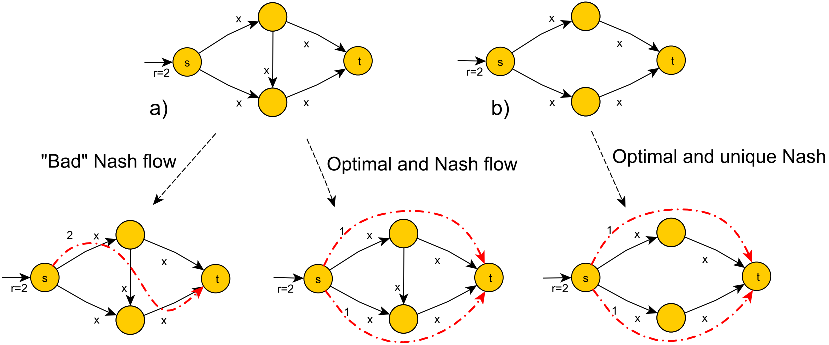
Network (a) has worst Nash flow $BC = 2$



Braess Paradox: Network's performance may improve by removing edges

Braess Paradox

Network (a) has worst Nash flow $BC = 2$ while Network (b) has worst Nash flow $BC = 1$



Braess Paradox: Network's performance may improve by removing edges

Network Design Hardness Results

Best Subnetwork: Find subnetwork for which worst Nash flow cost is minimized.

- It is NP-hard to approximate Best Subnetwork within a factor of 4/3
- It is NP-hard to approximate Best Subnetwork within a factor of $O(n^{0.121-\epsilon})$

Reductions from 2DDP (2 Directed Disjoint Paths Problem)

Network Design Hardness Results

Best Subnetwork: Find subnetwork for which worst Nash flow cost is minimized.

- It is NP-hard to approximate Best Subnetwork within a factor of $4/3$
- It is NP-hard to approximate Best Subnetwork within a factor of $O(n^{0.121-\epsilon})$

Reductions from 2DDP (2 Directed Disjoint Paths Problem)

Network Design Hardness Results

Best Subnetwork: Find subnetwork for which worst Nash flow cost is minimized.

- It is NP-hard to approximate Best Subnetwork within a factor of $4/3$
- It is NP-hard to approximate Best Subnetwork within a factor of $O(n^{0.121-\epsilon})$

Reductions from 2DDP (2 Directed Disjoint Paths Problem)

Network Design Hardness Results

Best Subnetwork: Find subnetwork for which worst Nash flow cost is minimized.

- It is NP-hard to approximate Best Subnetwork within a factor of $4/3$
- It is NP-hard to approximate Best Subnetwork within a factor of $O(n^{0.121-\epsilon})$

Reductions from 2DDP (2 Directed Disjoint Paths Problem)

Network Design Hardness Results

Best Subnetwork: Find subnetwork for which worst Nash flow cost is minimized.

- It is NP-hard to approximate Best Subnetwork within a factor of $4/3$
- It is NP-hard to approximate Best Subnetwork within a factor of $O(n^{0.121-\epsilon})$

Reductions from 2DDP (2 Directed Disjoint Paths Problem)

Reducing 2DDP to Approximating Best Subnetwork

Given an instance of 2DDP we construct the "base" network.

- Yes instance $\rightarrow \exists$ subgraph with worst Nash flow cost$= r/4$
- No instance $\rightarrow \forall$ subgraph worst Nash flow cost $\geq r/3$

Reduction provides a 4/3 gap for Best Subnetwork Problem or else:

There exists a $(4/3 - \epsilon)$-approximation algorithm

$\Downarrow$

In a Yes instance, returned solution cost $\leq (4/3 - \epsilon)r/4 < r/3$

Reducing 2DDP to Approximating Best Subnetwork

Given an instance of 2DDP we construct the "base" network.

- Yes instance $\rightarrow \exists$ subgraph with worst Nash flow cost$= r/4$
- No instance $\rightarrow \forall$ subgraph worst Nash flow cost $\geq r/3$

Reduction provides a 4/3 gap for Best Subnetwork Problem
or else:

There exists a $(4/3 - \epsilon)$-approximation algorithm

$$\Downarrow$$

In a Yes instance, returned solution cost $\leq (4/3 - \epsilon) r/4 < r/3$

Reducing 2DDP to Approximating Best Subnetwork

Given an instance of 2DDP we construct the "base" network.

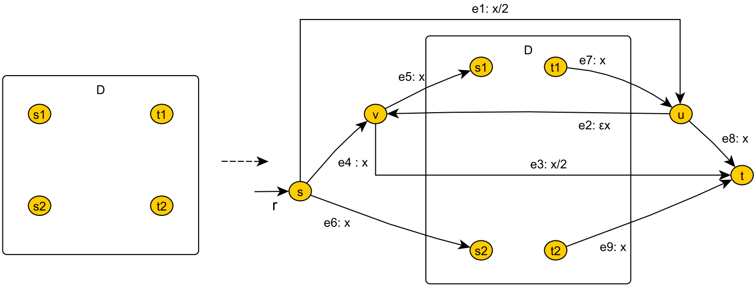
- Yes instance $\rightarrow \exists$ subgraph with worst Nash flow cost$= r/4$
- No instance $\rightarrow \forall$ subgraph worst Nash flow cost $\geq r/3$

Reduction provides a $4/3$ gap for Best Subnetwork Problem
or else:

$$\text{There exists a } (4/3 - \epsilon)\text{-approximation algorithm}$$
$$\Downarrow$$
$$\text{In a Yes instance, returned solution cost} \leq (4/3 - \epsilon) r/4 < r/3$$

Reducing 2DDP to Approximating Best Subnetwork

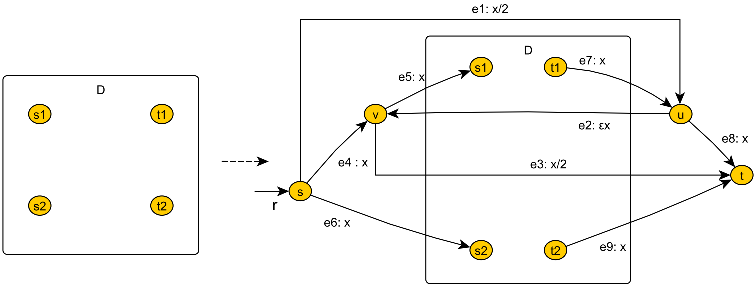
Given Network D we construct Network G by adding these external vertices and edges



- There are 2 paths from $\{s_1, s_2\}$ to $\{t_1, t_2\}$
- Optimal uses the "quick" path and two slow paths and achieve $BC = r/4$

Reducing 2DDP to Approximating Best Subnetwork

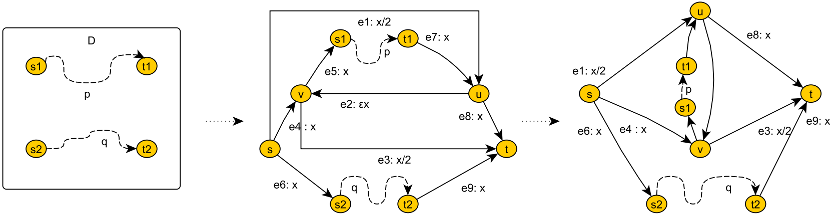
Given Network D we construct Network G by adding these external vertices and edges



- There are 2 paths from $\{s_1, s_2\}$ to $\{t_1, t_2\}$
- Optimal uses the "quick" path and two slow paths and achieve $BC = r/4$

D is a YES instance of 2DDP

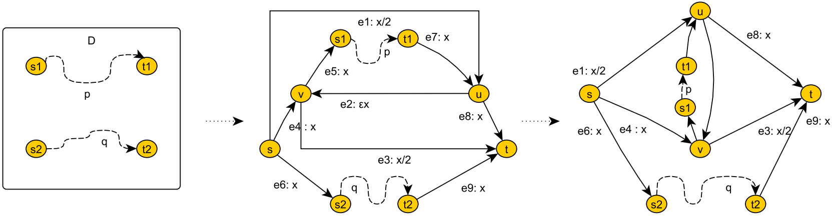
- Let p,q be the disjoint paths
- Keep external graph , p and q.



- Moving from u to v and vice versa is "free"
- Unique Nash flow uses the quick path and two "slow" ones: worst Nash flow $BC = r/4$

D is a YES instance of 2DDP

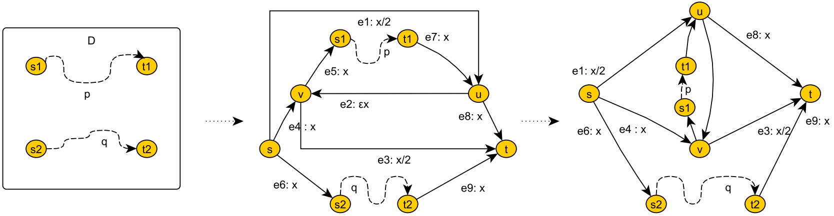
- Let p,q be the disjoint paths
- Keep external graph , p and q.



- Moving from u to v and vice versa is "free"
- Unique Nash flow uses the quick path and two "slow" ones: worst Nash flow $BC = r/4$

D is a YES instance of 2DDP

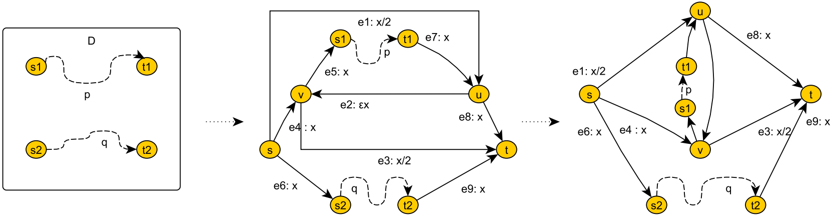
- Let p,q be the disjoint paths
- Keep external graph , p and q.



- Moving from u to v and vice versa is “free”
- Unique Nash flow uses the quick path and two “slow” ones: worst Nash flow $BC = r/4$

D is a YES instance of 2DDP

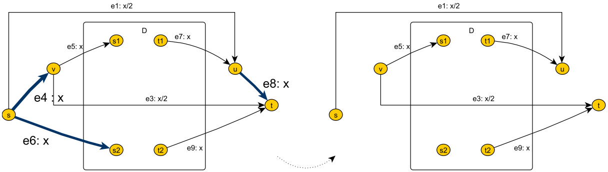
- Let p,q be the disjoint paths
- Keep external graph , p and q.



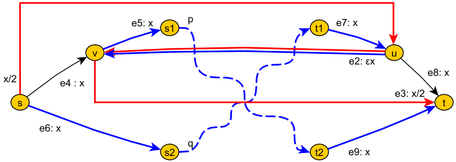
- Moving from u to v and vice versa is “free”
- Unique Nash flow uses the quick path and two “slow” ones: worst Nash flow $BC = r/4$

D is a NO instance of 2DDP

When e_2 is missing e_4, e_6 and e_8 form a "slow" cut



Paths p,q join $s_1 - t_2$ and $s_2 - t_1 \Rightarrow$ Nash flow that loads* only two paths



["slow" cut] or [1 "quick" and 1 "slow" path] $\Rightarrow$ *worst Nash BC* $\geq r/3$

Inapproximability of Best Subnetwork

With the "base" network we achieve a 4/3 gap for Best Subnetwork Problem

We can amplify gaps

- If network G provides gap γ
- G combined with base network provides gap $4/3\ \gamma$

Applying recursively for $k = log_{4/3} n$ times, we get a Network with $O(8^k n) = O(n^{8.23})$ vertices and edges providing gap n

Inapproximability of Best Subnetwork

With the "base" network we achieve a $4/3$ gap for Best Subnetwork Problem

We can amplify gaps

- If network G provides gap γ
- G combined with base network provides gap $4/3\ \gamma$

Applying recursively for $k = log_{4/3}n$ times, we get a Network with $O(8^k n) = O(n^{8.23})$ vertices and edges providing gap n

Inapproximability of Best Subnetwork

With the "base" network we achieve a 4/3 gap for Best Subnetwork Problem

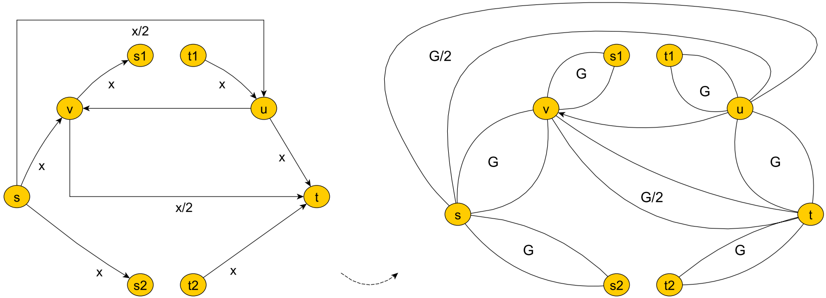
We can amplify gaps

- If network G provides gap γ
- G combined with base network provides gap $4/3\ \gamma$

Applying recursively for $k = log_{4/3} n$ times, we get a Network with $O(8^k n) = O(n^{8.23})$ vertices and edges providing gap n

Amplify the gap

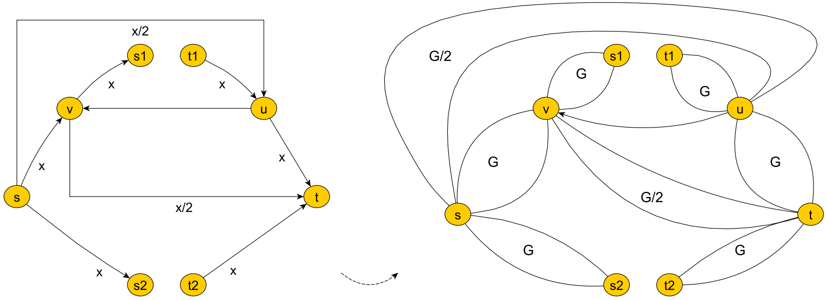
- If network G (with striclty increasing linear latencies) provides gap γ
- G combined with base network provides gap $(4/3)\ \gamma$



Amplifying a gap by 4/3 factor

Amplify the gap

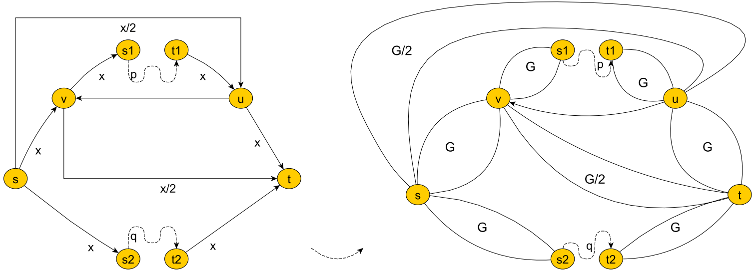
- If network G (with striclty increasing linear latencies) provides gap γ
- G combined with base network provides gap $(4/3)\ \gamma$



YES instance of 2DDP

We keep the subgraph with

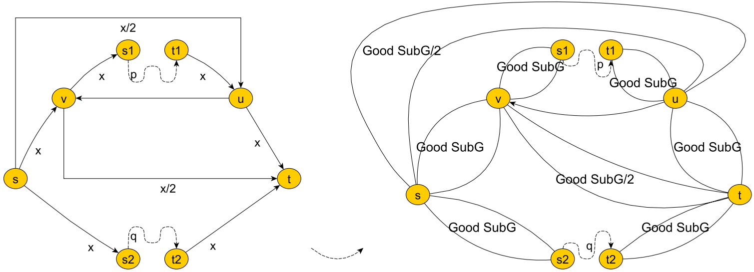
- the paths p,q and
- the good subgraphs in the copies of G



YES instance of 2DDP

We keep the subgraph with

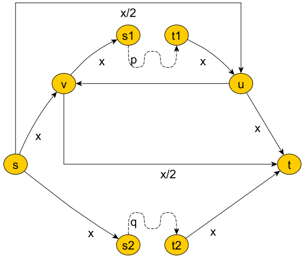
- the paths p,q and
- the good subgraphs in the copies of G

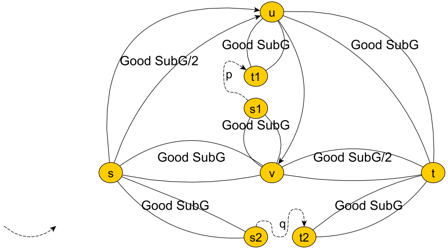


YES instance of 2DDP

We keep the subgraph with

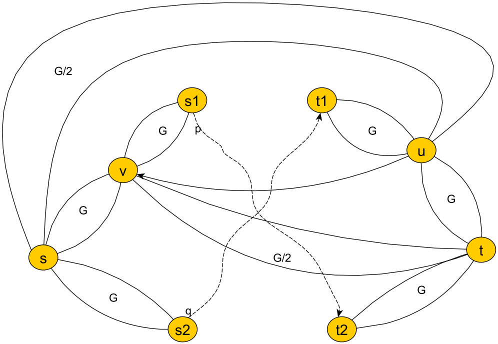
- the paths p,q and
- the good subgraphs in the copies of G





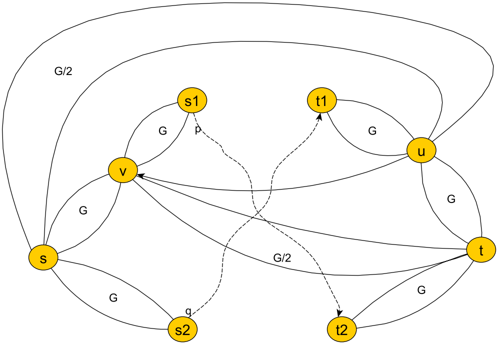
NO instance of 2DDP

- We have to check all subgraphs
- Any subgraph of the copies has few blocking paths



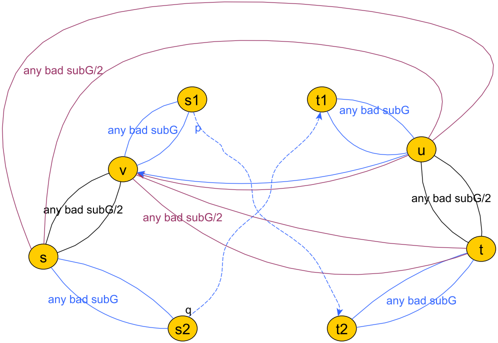
NO instance of 2DDP

- We have to check all subgraphs
- Any subgraph of the copies has few blocking paths



NO instance of 2DDP

- We have to check all subgraphs
- Any subgraph of the copies has few blocking paths
- Try to block all paths in a way similar with the base case



γ gap on each subnetwork and 4/3 because of the "base" network

Interesting Point

How do we solve 2DDP with "good" approximation Algorithm?

- We do not know how to compute worst Nash flow efficiently.
- In Yes instances, a solution of 2DDP exists inside the returned network
- There are polynomial many networks
- We can check in polynomial time each one

Interesting Point

How do we solve 2DDP with "good" approximation Algorithm?

- We do not know how to compute worst Nash flow efficiently.
- In Yes instances, a solution of 2DDP exists inside the returned network
- There are polynomial many networks
- We can check in polynomial time each one

Interesting Point

How do we solve 2DDP with "good" approximation Algorithm?

- We do not know how to compute worst Nash flow efficiently.
- In Yes instances, a solution of 2DDP exists inside the returned network
- There are polynomial many networks
- We can check in polynomial time each one

Interesting Point

How do we solve 2DDP with "good" approximation Algorithm?

- We do not know how to compute worst Nash flow efficiently.
- In Yes instances, a solution of 2DDP exists inside the returned network
- There are polynomial many networks
- We can check in polynomial time each one

Interesting Point

How do we solve 2DDP with "good" approximation Algorithm?

- We do not know how to compute worst Nash flow efficiently.
- In Yes instances, a solution of 2DDP exists inside the returned network
- There are polynomial many networks
- We can check in polynomial time each one