



How bad is Selfish Routing?

Bounding and Handling Selfishness

Silvia Schwarze, University of Kaiserslautern

Dirk Stegemann, University of Mannheim

Silke Wagner, University of Karlsruhe

Overview

- The Price of Anarchy (Silvia)
- Designing of Networks for Selfish Users (Dirk)
- Stackelberg Routing (Silke)



The Price of Anarchy

*Can the negative effect of selfish behavior
be bounded?*

The Price of Anarchy

- Introduction
- The Model
- Bounding the Price of Anarchy
- Conclusions

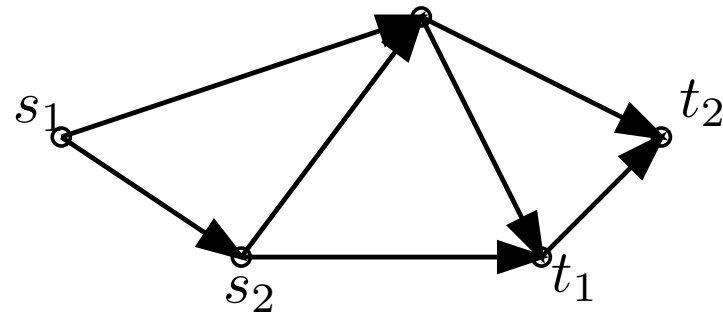
The Price of Anarchy

- Introduction
- The Model
- Bounding the Price of Anarchy
- Conclusions

Introduction

Given:

- Network
- OD-pairs
- flow rate
- infinitely many users
- load-dependent latency



- ## Problem:
- Selfish users act independently, considering own benefit, not overall social benefit

Introduction

- congestion of paths is produced by all users
- selfish user chooses way that minimizes the latency from origin to destination
- system ends up with an equilibrium situation, which is usual not optimal

Introduction

- congestion of paths is produced by all users
- selfish user chooses way that minimizes the latency from origin to destination
- system ends up with an equilibrium situation, which is usual not optimal

Question:

How much does the performance of the network suffer from selfish behavior?

The Price of Anarchy

- Introduction
- **The Model**
- Bounding The Price of Anarchy
- Conclusions

Preliminaries

- Directed Network $G = (V, E)$
parallel edges allowed, no self-loops
- k origin-destination pairs $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ with flow rate r_i
- \mathcal{P}_i : set of $s_i - t_i$ paths, $\mathcal{P} = \cup_i \mathcal{P}_i$
- flow: $f : \mathcal{P} \rightarrow \mathcal{R}^+$ for a fixed flow: $f_e = \sum_{P:e \in P} f_P$
- feasible flow: $\sum_{P \in \mathcal{P}_i} f_P = r_i \forall i$

Preliminaries

- latency functions: $l_e(f_e)$
Assumption: $l_e(f_e)$ nonnegative, differentiable, nondecreasing
- instance: (G, r, l)
- latency of a path: $l_P(f_P) = \sum_{e \in P} l_e(f_e)$
- Cost of a flow:

$$C(f) = \sum_{P \in \mathcal{P}} l_P(f_P) f_P = \sum_{e \in E} l_e(f_e) f_e$$

Nash Flow

Definition (1)

A feasible flow f is at a **Nash equilibrium** if $l_{P_1}(f) \leq l_{P_2}(\tilde{f}), \forall P_1, P_2 \in \mathcal{P}_i$ with $f_{P_1} > 0, i = 1, \dots, k$ and $\forall \delta \in (0, f_{P_1}]$, where

$$\tilde{f}_P = \begin{cases} f_P - \delta & \text{if } P = P_1 \\ f_P + \delta & \text{if } P = P_2 \\ f_P & \text{if } P \notin \{P_1, P_2\} \end{cases} .$$

Nash Flow

Proposition (1)

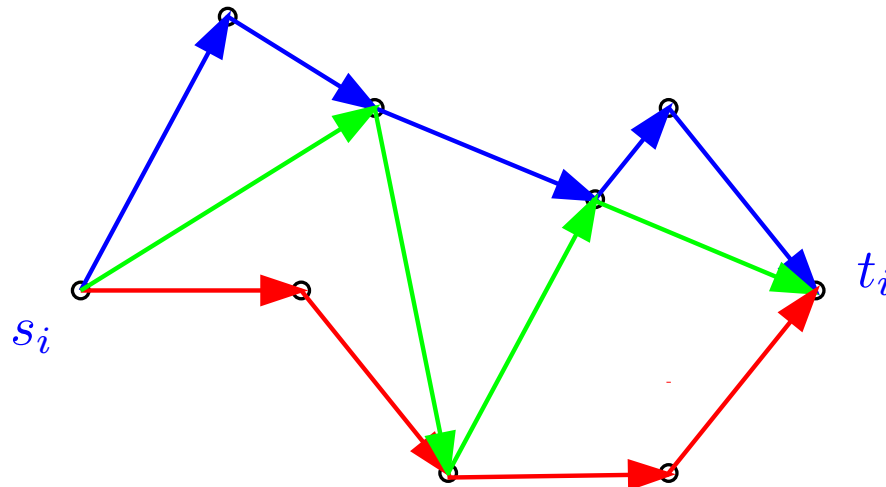
Wardrop's principle

A flow f feasible for (G, r, l) is at Nash equilibrium if and only if $l_{P_1}(f) \leq l_{P_2}(f) \forall P_1, P_2 \in \mathcal{P}_i$ with $f_{P_1} > 0, i = 1, \dots, k$.

Nash Flow

Observation (1)

f Nash flow \Rightarrow all $s_i - t_i$ paths with positive flow have the same latency $L_i(f)$



Optimal flow

Definition (2)

- $l(x)$ is **standard** if $l(x)x$ is convex on $[0, \infty)$
- Class of latency functions \mathcal{L} is **standard** if it contains a nonzero function and each $l(x) \in \mathcal{L}$ is standard.
- **marginal cost function** $l^*(x)$:

$$l^*(x) = \frac{d}{dx}(l(x)x) = l(x) + l'(x)x$$

Optimal flow

- Optimal flow: flow that minimizes $C(f)$

Optimal flow

- Optimal flow: flow that minimizes $C(f)$
- Formulation as NLP:

$$\text{minimize } \sum_{e \in E} l_e(f_e) f_e$$

$$\text{s.t. } \sum_{P \in \mathcal{P}_i} f_P = r_i \quad \forall i = 1, \dots, k$$

$$f_e = \sum_{P \in \mathcal{P}: e \in E} f_P \quad \forall e \in E$$

$$f_P \geq 0 \quad \forall P \in \mathcal{P}$$

Optimal flow

- Optimal flow: flow that minimizes $C(f)$
- Formulation as NLP:

$$\begin{aligned} & \text{minimize} && \sum_{e \in E} l_e(f_e) f_e \\ & \text{s.t.} && \sum_{P \in \mathcal{P}_i} f_P = r_i \quad \forall i = 1, \dots, k \\ & && f_e = \sum_{P \in \mathcal{P}: e \in E} f_P \quad \forall e \in E \\ & && f_P \geq 0 \quad \forall P \in \mathcal{P} \end{aligned}$$

- NLP is convex for standard latency functions.

Optimal Flow

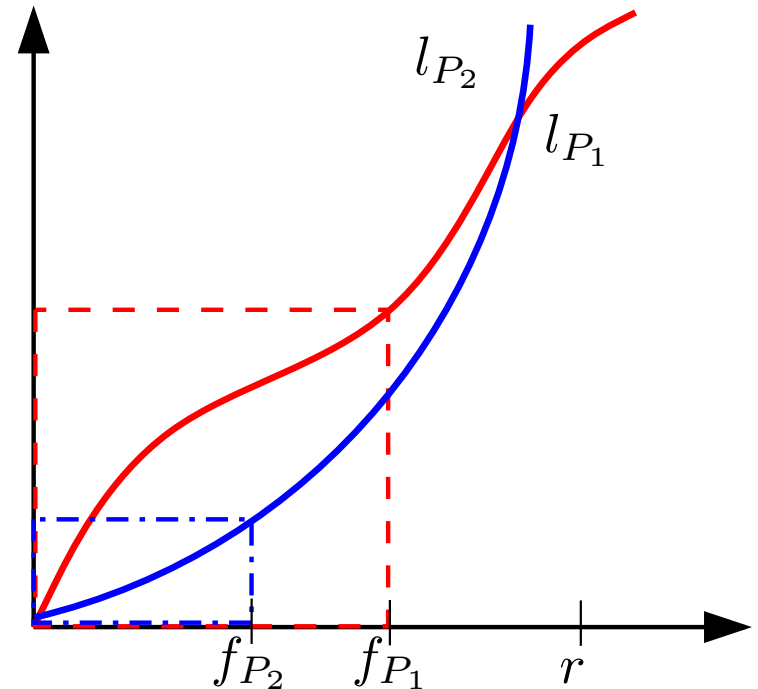
Lemma (1)

f is optimal for a convex NLP if and only if for every $P_1, P_2 \in \mathcal{P}_i$ with

$f_{P_1} > 0$ holds that

$$(l_{P_1}(f_{P_1})f_{P_1})' \leq (l_{P_2}(f_{P_2})f_{P_2})'$$

, $i = 1, \dots, k$



Optimal Flow

Conditions of Lemma 1 are similar to conditions for Nash flow:

$$l_{P_1}(f) \leq l_{P_2}(f) \text{ with } f_{P_1} > 0$$

Corollary (1)

$l(x)$ standard, $l^*(x) = (l(x)x)'$:

f feasible in (G, r, l) is optimal if and only if f is a Nash flow for (G, r, l^*) .

Nash Flow

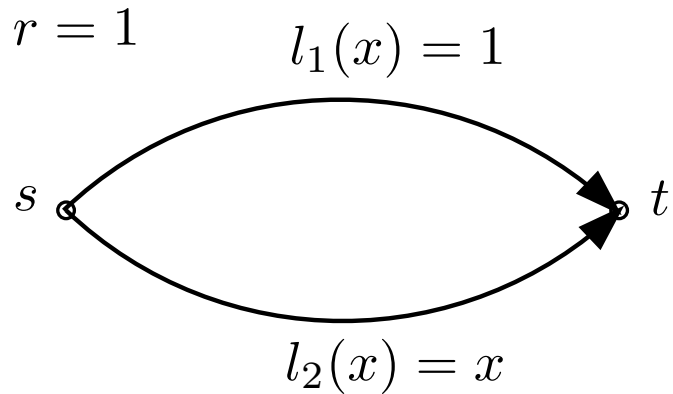
Observation (2)

Corollary 1

\Rightarrow *Existence and Uniqueness of Nash*

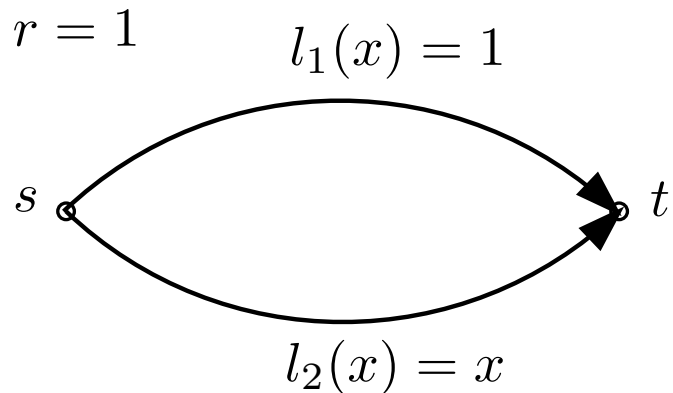
\Rightarrow *For an optimal flow the marginal costs have to be equal*

Pigou's Example



$$C(f) = 1 \cdot f_1 + f_2^2$$

Pigou's Example

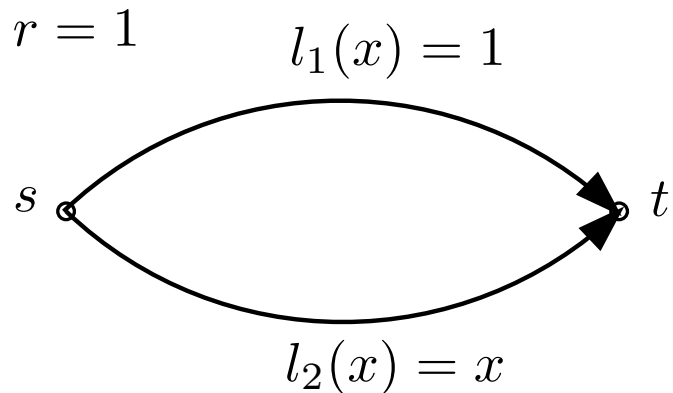


$$C(f) = 1 \cdot f_1 + f_2^2$$

Nash:

$$f_1 = 0, f_2 = 1 \Rightarrow C(f) = 1$$

Pigou's Example



$$C(f) = 1 \cdot f_1 + f_2^2$$

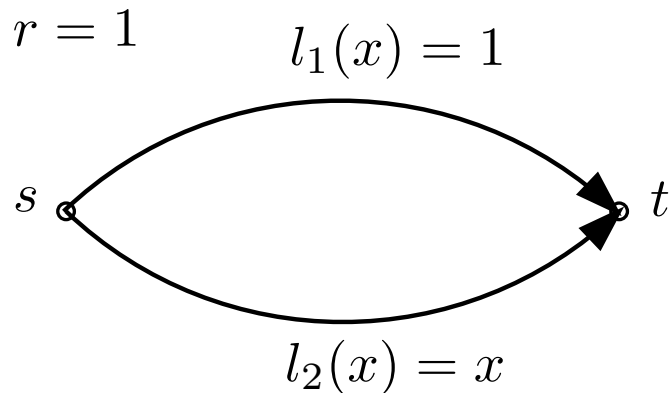
Nash:

$$f_1 = 0, f_2 = 1 \Rightarrow C(f) = 1$$

Optimal Flow:

$$f_1^* = f_2^* = \frac{1}{2} \Rightarrow C(f^*) = \frac{3}{4}$$

Pigou's Example



marginal costs:

$$l_1^*(x) = 1, l_2^*(x) = 2x$$

$$C(f) = 1 \cdot f_1 + f_2^2$$

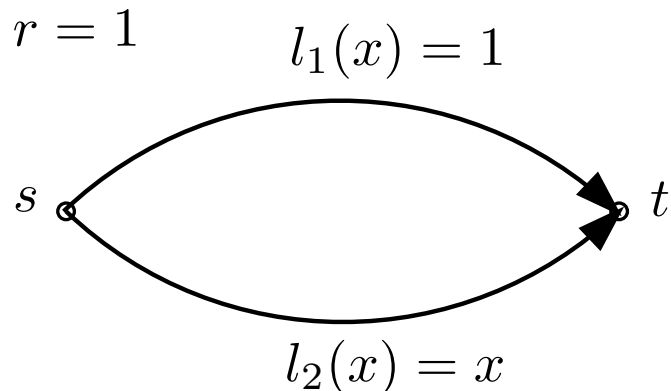
Nash:

$$f_1 = 0, f_2 = 1 \Rightarrow C(f) = 1$$

Optimal Flow:

$$f_1^* = f_2^* = \frac{1}{2} \Rightarrow C(f^*) = \frac{3}{4}$$

Pigou's Example



marginal costs:

$$l_1^*(x) = 1, l_2^*(x) = 2x$$

Nash for (G, r, l^*) :

$$f_1^* = f_2^* = \frac{1}{2}$$

→ **Corollary 1**

$$C(f) = 1 \cdot f_1 + f_2^2$$

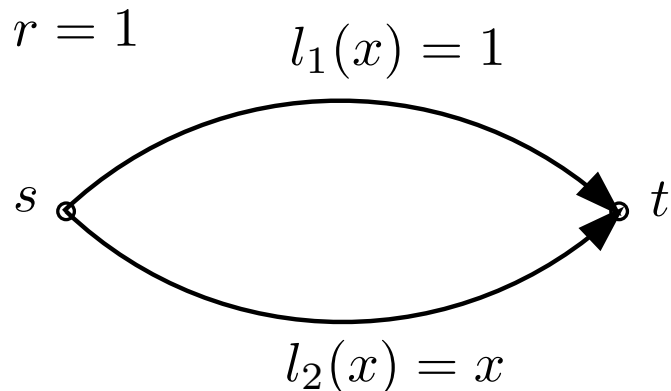
Nash:

$$f_1 = 0, f_2 = 1 \Rightarrow C(f) = 1$$

Optimal Flow:

$$f_1^* = f_2^* = \frac{1}{2} \Rightarrow C(f^*) = \frac{3}{4}$$

Pigou's Example



$$C(f) = 1 \cdot f_1 + f_2^2$$

Nash:

$$f_1 = 0, f_2 = 1 \Rightarrow C(f) = 1$$

Optimal Flow:

$$f_1^* = f_2^* = \frac{1}{2} \Rightarrow C(f^*) = \frac{3}{4}$$

marginal costs:

$$l_1^*(x) = 1, l_2^*(x) = 2x$$

Nash for (G, r, l^*) :

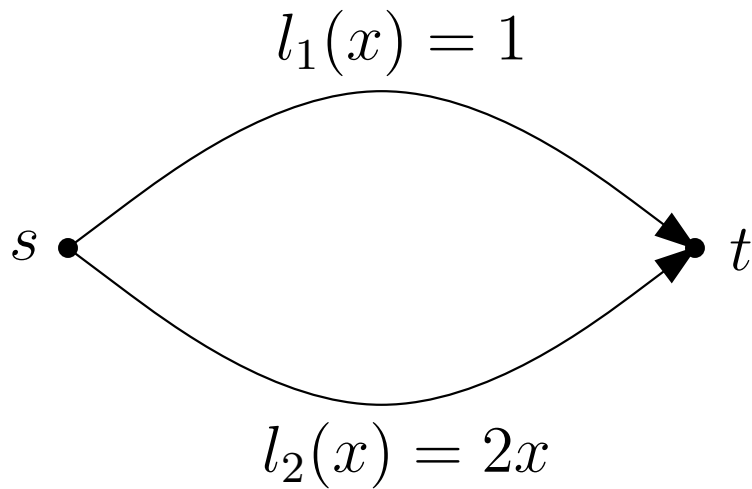
$$f_1^* = f_2^* = \frac{1}{2}$$

→ **Corollary 1**

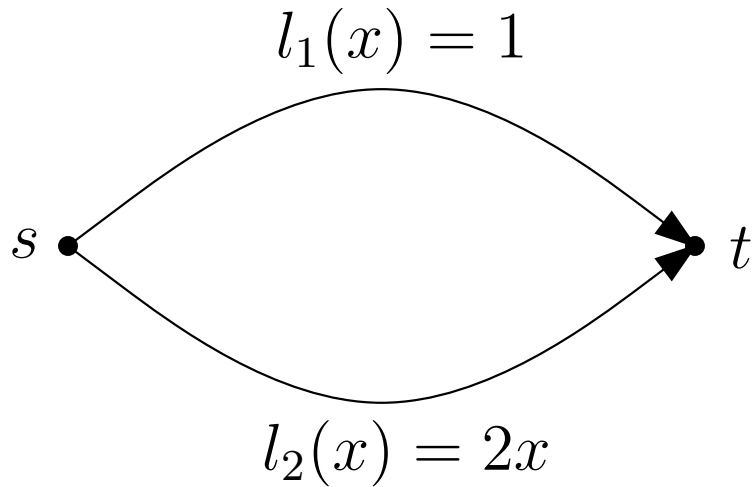
Price of Anarchy:

$$\rho(G, r, l) = \frac{C(f)}{C(f^*)} = \frac{4}{3}$$

Pigou's Example (modified)

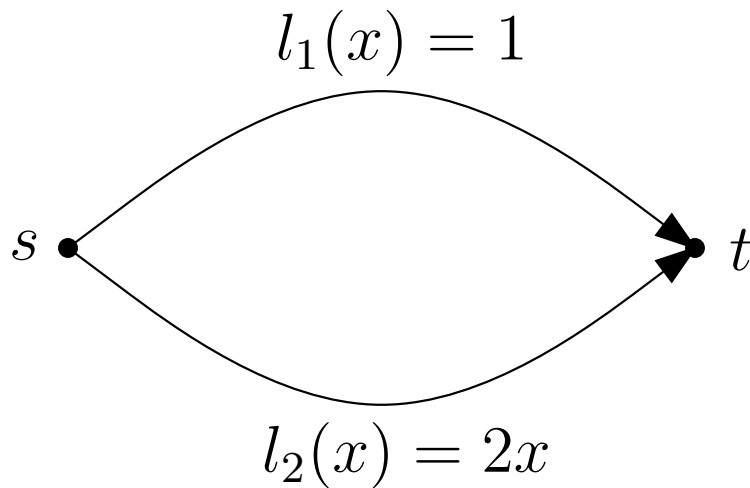


Pigou's Example (modified)



Nash - *unchanged* -

Pigou's Example (modified)



Nash - *unchanged* -

Optimal Flow:

$$C(f^*) \xrightarrow{p \rightarrow \infty} 0$$

Price of Anarchy:

$$\Rightarrow \rho(G, r, l) \xrightarrow{p \rightarrow \infty} \infty$$

Pigou's example

Bad News:

There is no upper bound on the **Price of Anarchy** for instances with general latency functions.

The Price of Anarchy

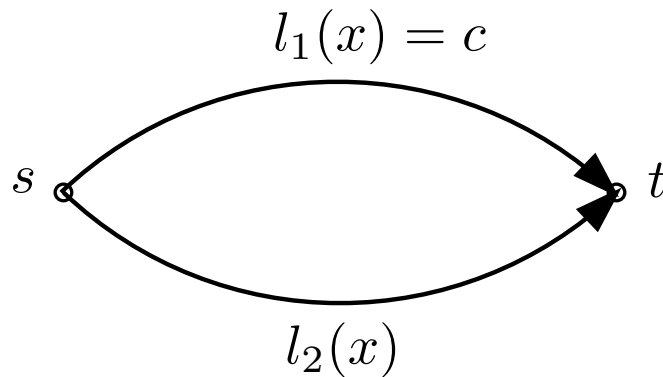
- Introduction
- The Model
- Bounding the Price of Anarchy
- Conclusion

Some first bounds

- A Bound on $C(f)$
- Non-optimal bound for „not too steep” functions
- Tight bound for linear latency functions:
$$\rho(G, r, l) \leq \frac{4}{3}$$

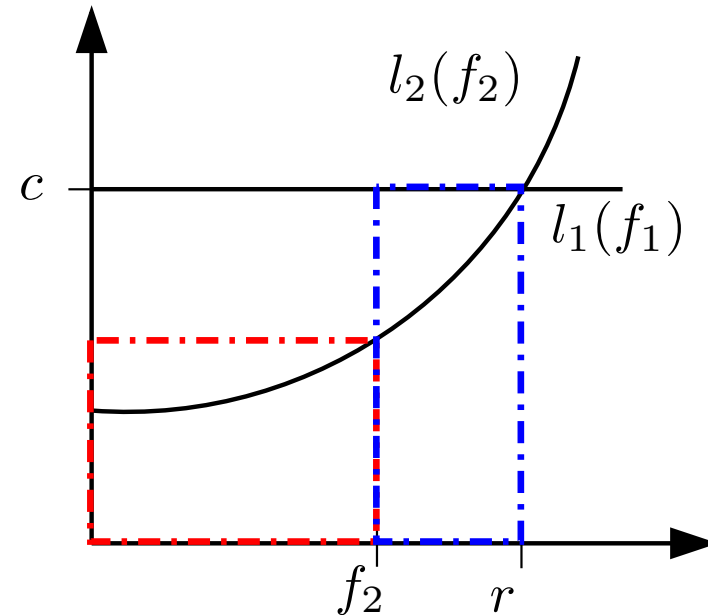
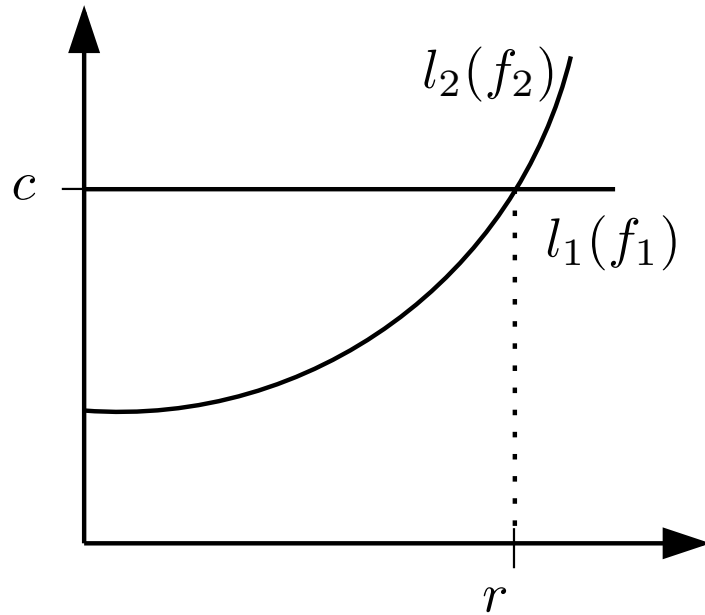
The Anarchy Value

Define a bound on ρ valid for **classes of latency functions** \mathcal{L}
→ Mimic worst-case scenario of Pigou



- $l_1(x) = c$
- $l_2(0) < c, l_2(x) > c$ for some x , $l_2(x)$ as steep as possible

The Anarchy Value



choose $r: l_2(r) = c$

The Anarchy Value

- Nash: routes all flow via $e_2 \Rightarrow C(f) = l_2(r)r = cr$
- Optimal flow: routes share $\lambda \in [0, 1]$ via e_2 :
 $f_2 = \lambda r \Rightarrow C(f^*) = \lambda r l_2(\lambda r) + (1 - \lambda)rc$
- Optimal flow: marginal cost functions have to be equal:
 $l_2^*(\lambda r) = l_1^*((1 - \lambda)r) = c = l_2(r)$
- set $\mu = \frac{l_2(\lambda r)}{l_2(r)} = \frac{l_2(\lambda r)}{c}$
- $\Rightarrow \rho(G, r, l) = \frac{cr}{cr(\lambda\mu + (1 - \lambda))} = [\lambda\mu + (1 - \lambda)]^{-1}$

The Anarchy Value

Definition (3)

Anarchy Value:

$$\alpha(l) = \sup_{r>0:l(r)>0} [\lambda\mu + (1 - \lambda)]^{-1}$$

$$\alpha(\mathcal{L}) = \sup_{0 \neq l \in \mathcal{L}} \alpha(l)$$

The Anarchy Value

Theorem 3

\mathcal{L} standard class, (G, r, l) with $l \in \mathcal{L}$:

$$\rho(G, r, l) \leq \alpha(\mathcal{L})$$

Anarchy Value and Network Topology

Simplest networks provide worst-cases

- Class \mathcal{L} that contains all constant functions
 - G_2 : Graph with two vertices, two edges
 - \mathcal{I}_2 : single-commodity instances on G_2
 - \mathcal{I} : instances with $l \in \mathcal{L}$

$$\sup_{(G_2, r, l) \in \mathcal{I}_2} \rho(G_2, r, l) = \alpha(\mathcal{L}) = \sup_{(G, r, l) \in \mathcal{I}} \rho(G, r, l)$$

⇒ Price of Anarchy is independent of the network topology

Conclusion

- Nash equilibrium f
- Optimal flow f^*
- Compare Nash and optimum by price of anarchy $\rho(G, r, l)$
- no general upper bound on $\rho(G, r, l)$
- anarchy value $\alpha(\mathcal{L})$:
 - upper bound on $\rho(G, r, l)$
 - already tight bound for simple instances



Designing Networks for Selfish Users

Coping with Selfishness I



Overview

1. Introduction to the Network Design Problem



Overview

1. Introduction to the Network Design Problem
2. Possible approaches



Overview

1. Introduction to the Network Design Problem
2. Possible approaches
3. Performance of the approaches

Overview

1. Introduction to the Network Design Problem
2. Possible approaches
3. Performance of the approaches
 - Networks with Linear Latency Functions

Overview

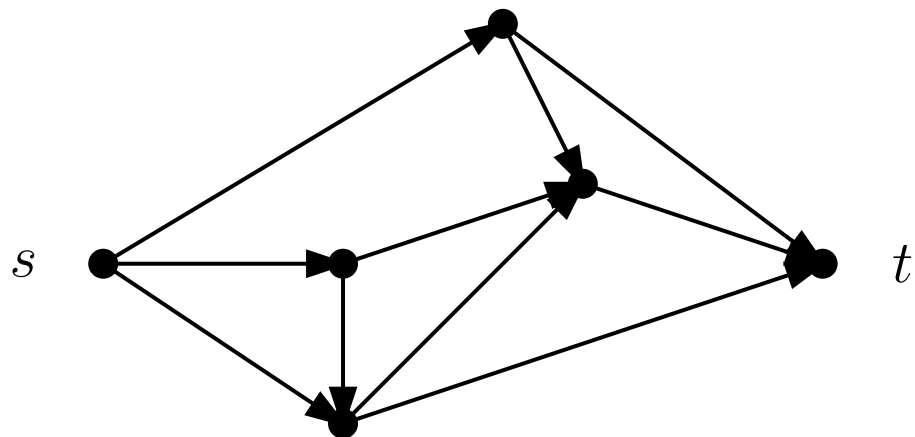
1. Introduction to the Network Design Problem
2. Possible approaches
3. Performance of the approaches
 - Networks with Linear Latency Functions
 - Networks with General Latency Functions

Overview

1. Introduction to the Network Design Problem
2. Possible approaches
3. Performance of the approaches
 - Networks with Linear Latency Functions
 - Networks with General Latency Functions
4. Summary and Extensions

Introduction

Selfish users want to route traffic from a source s to a destination t via a network.

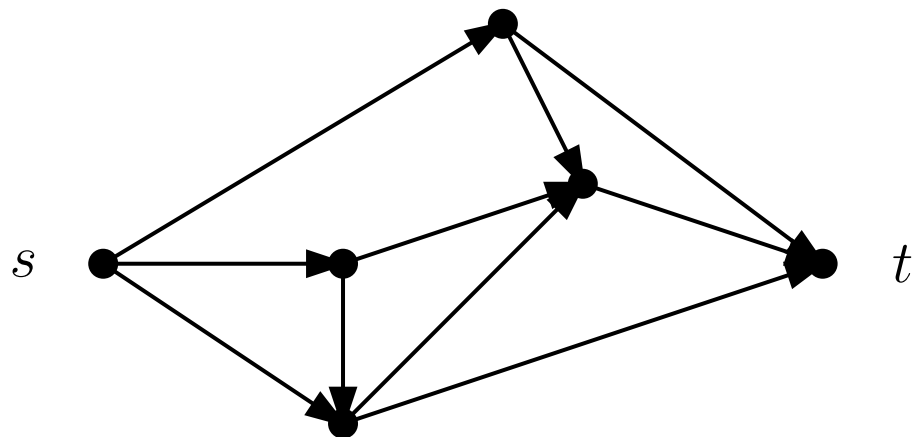


Introduction

Selfish users want to route traffic from a source s to a destination t via a network.

Examples:

- packets in a local area network

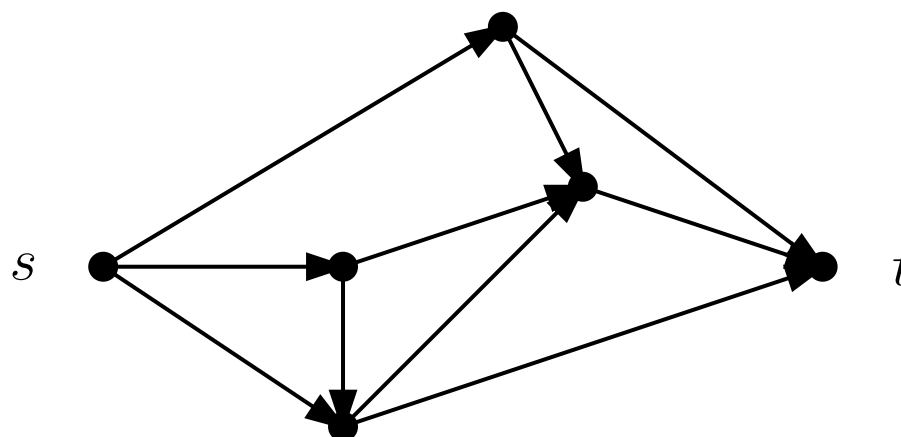


Introduction

Selfish users want to route traffic from a source s to a destination t via a network.

Examples:

- packets in a local area network
- cars in a highway system

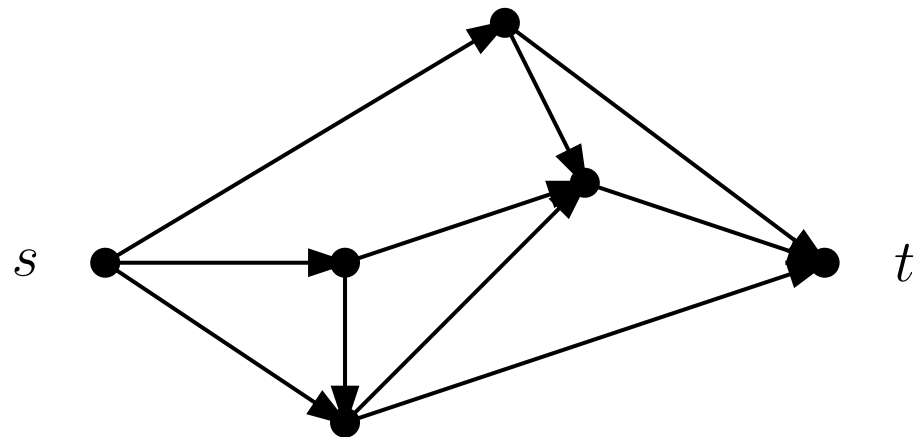


Introduction

Selfish users want to route traffic from a source s to a destination t via a network.

Examples:

- packets in a local area network
- cars in a highway system



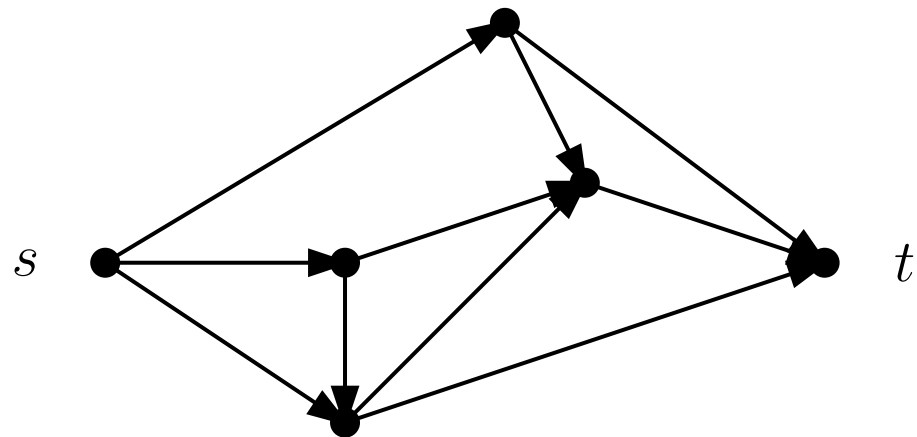
Recall: At Nash equilibrium, the end-to-end latency is the same on all st -paths.

Introduction

Selfish users want to route traffic from a source s to a destination t via a network.

Examples:

- packets in a local area network
- cars in a highway system



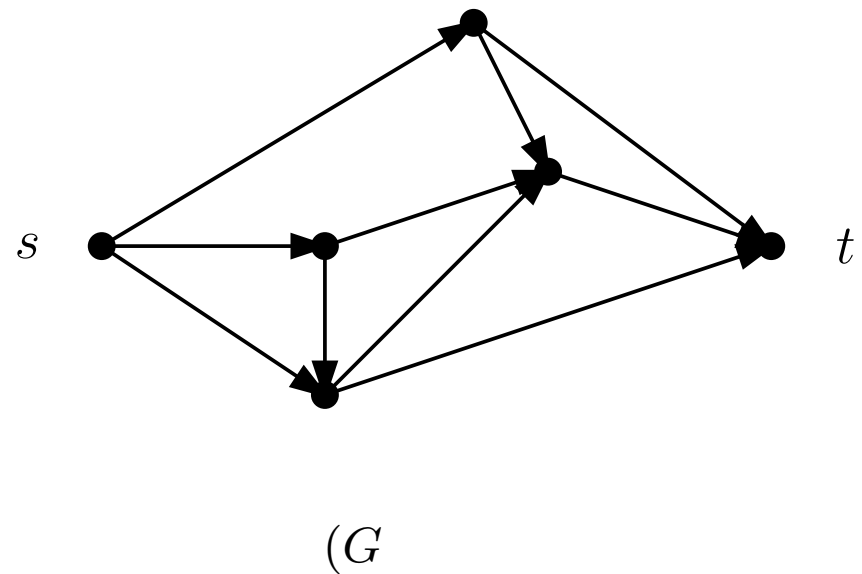
Recall: At Nash equilibrium, the end-to-end latency is the same on all st -paths.

Problem: How to build the network such that the common end-to-end latency is as small as possible?

The Model

We model this setting by

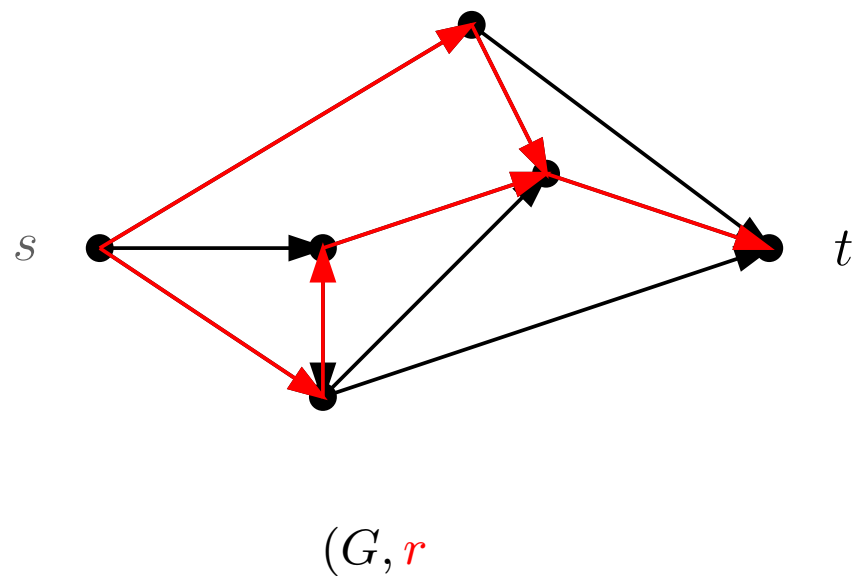
- a graph G consisting of
 - a set V of vertices
 - a set E of candidate edges
- a source-destination pair $\{s, t\}$



The Model

We model this setting by

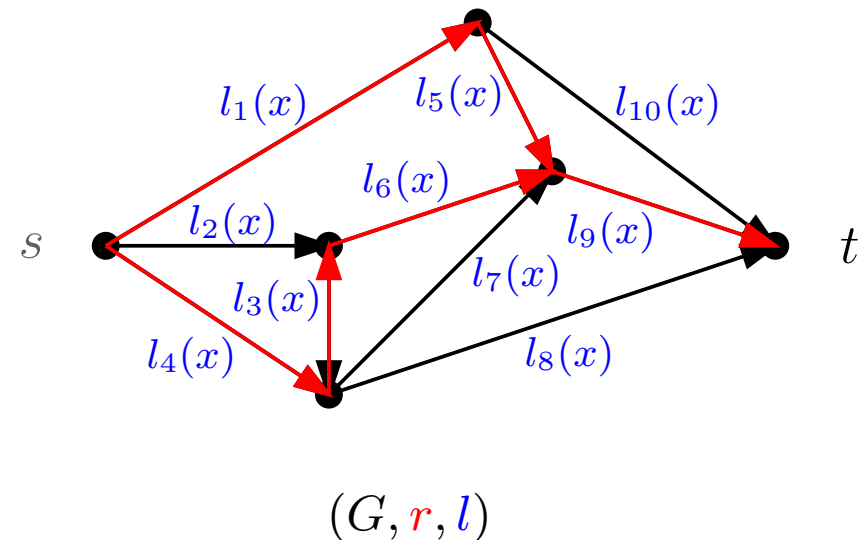
- a graph G consisting of
 - a set V of vertices
 - a set E of candidate edges
- a source-destination pair $\{s, t\}$
- a **flow rate** r to be routed from s to t



The Model

We model this setting by

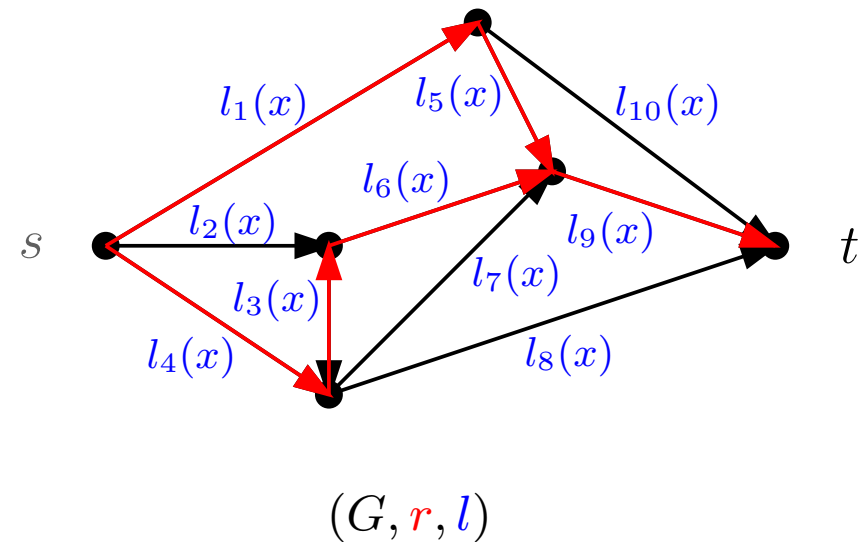
- a graph G consisting of
 - a set V of vertices
 - a set E of candidate edges
- a source-destination pair $\{s, t\}$
- a **flow rate** r to be routed from s to t
- continuous, non-decreasing **latency functions** l_e for all edges $e \in E$



The Model

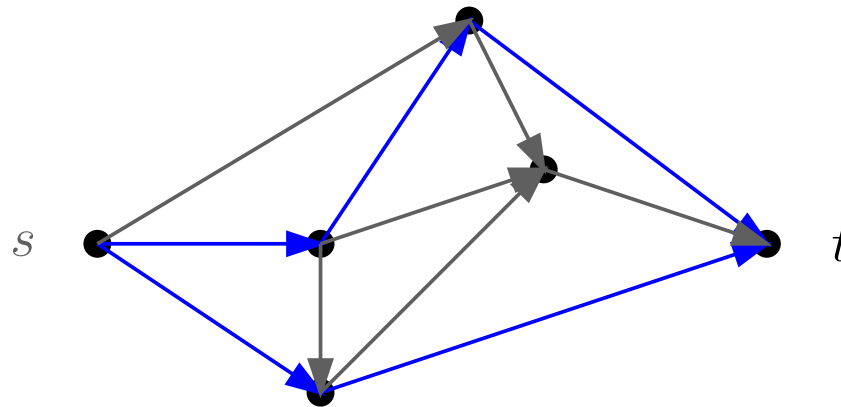
We model this setting by

- a graph G consisting of
 - a set V of vertices
 - a set E of candidate edges
- a source-destination pair $\{s, t\}$
- a **flow rate** r to be routed from s to t
- continuous, non-decreasing **latency functions** l_e for all edges $e \in E$



Denote the **common end-to-end latency** at equilibrium by $L(G, r, l)$.

The Network Design Problem



NETWORK DESIGN PROBLEM:

*Given an instance (G, r, l) , find a set of edges $E' \subseteq E$ such that for the **subgraph** $H = (V, E')$, the common end-to-end latency $L(H, r, l)$ is **minimum among all subgraphs.***

The Network Design Problem

Possible Approaches:

The Network Design Problem

Possible Approaches:

- Compute the optimal solution → time consuming

The Network Design Problem

Possible Approaches:

- Compute the optimal solution → time consuming
- Approximate the optimal solution in polynomial time

The Network Design Problem

Possible Approaches:

- Compute the optimal solution \rightarrow time consuming
- Approximate the optimal solution in polynomial time

Definition. A γ -**approximation algorithm** is a polynomial-time algorithm that, given an instance (G, r, l) , computes a subgraph H of G such that

$$L(H, r, l) \leq \gamma \cdot L(H^*, r, l)$$

where H^* denotes the optimal subgraph.

The Network Design Problem

Possible Approaches:

- Compute the optimal solution → time consuming
- Approximate the optimal solution in polynomial time

Definition. A γ -**approximation algorithm** is a polynomial-time algorithm that, given an instance (G, r, l) , computes a subgraph H of G such that

$$L(H, r, l) \leq \gamma \cdot L(H^*, r, l)$$

where H^* denotes the optimal subgraph.

→ Look for an approximation algorithm with minimal γ .

The Network Design Problem

Trivial algorithm:

Build all candidate edges, i.e. choose $H := G$.

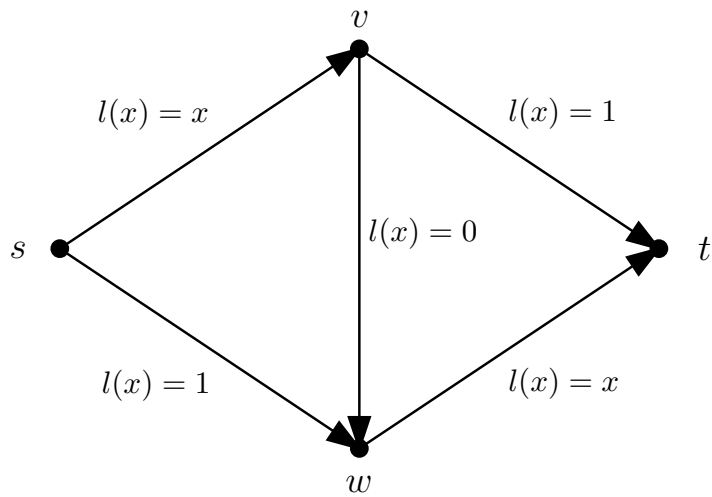
The Network Design Problem

Trivial algorithm:

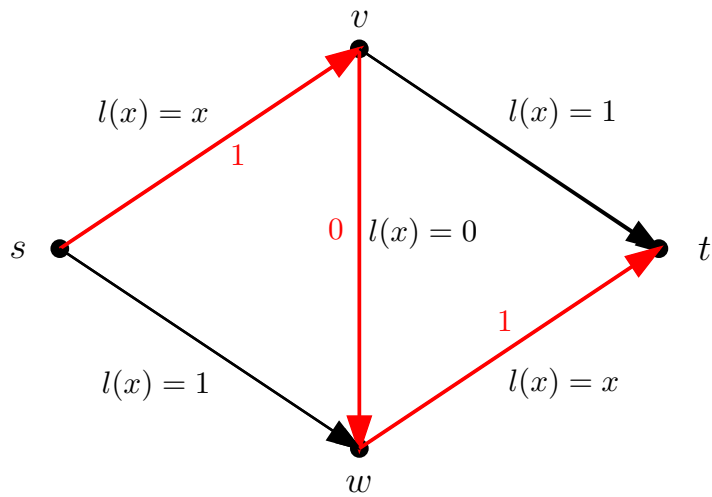
Build all candidate edges, i.e. choose $H := G$.

How good is this strategy?

Braess's Paradox - Basic Example

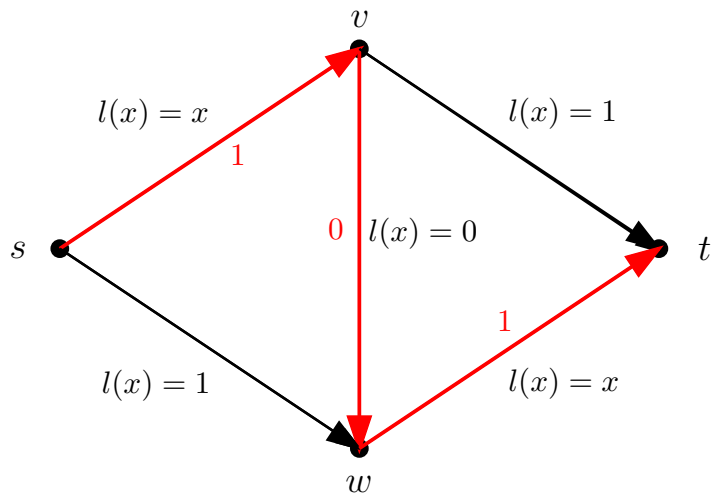


Braess's Paradox - Basic Example

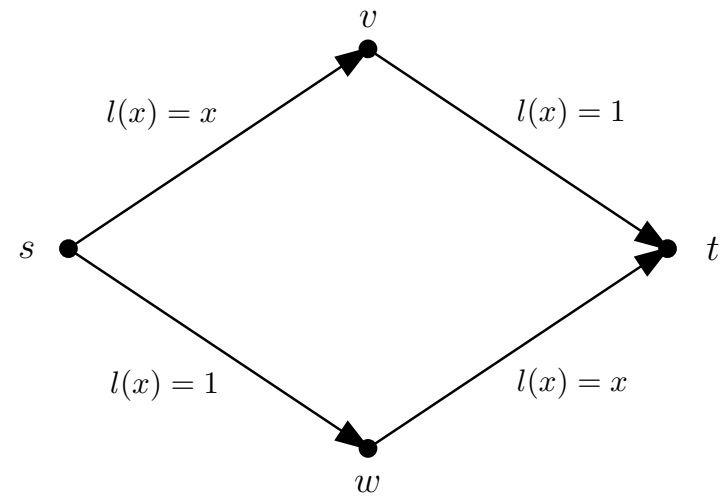


$$L(G, 1, l) = 2$$

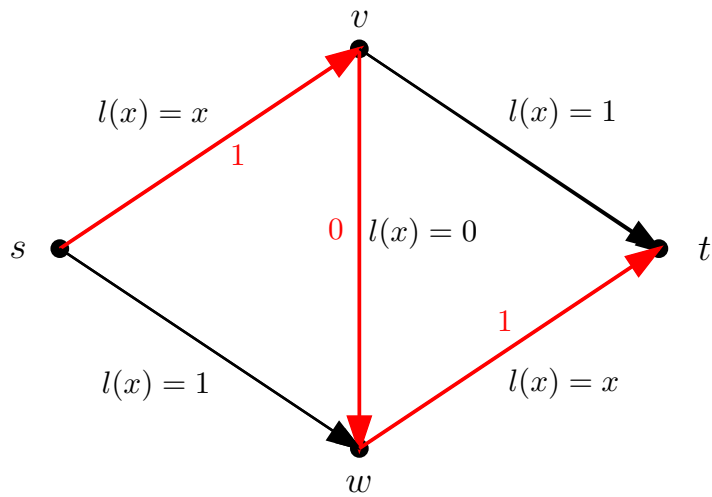
Braess's Paradox - Basic Example



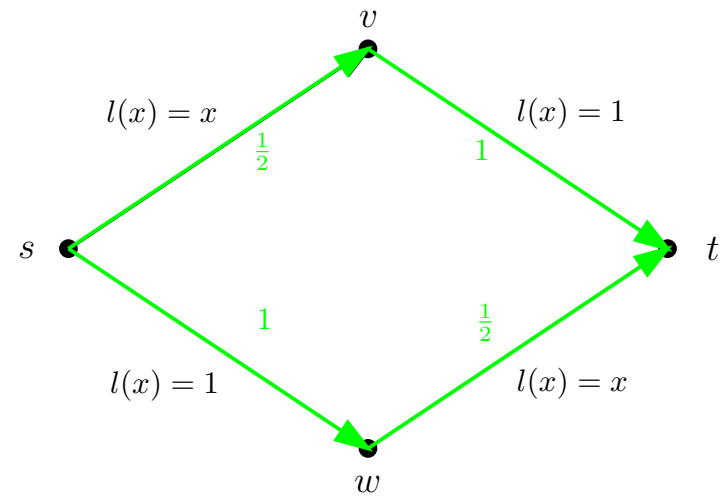
$$L(G, 1, l) = 2$$



Braess's Paradox - Basic Example

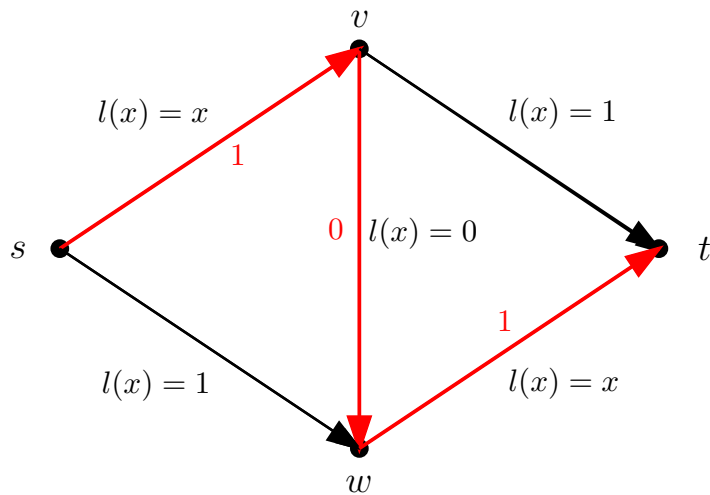


$$L(G, 1, l) = 2$$

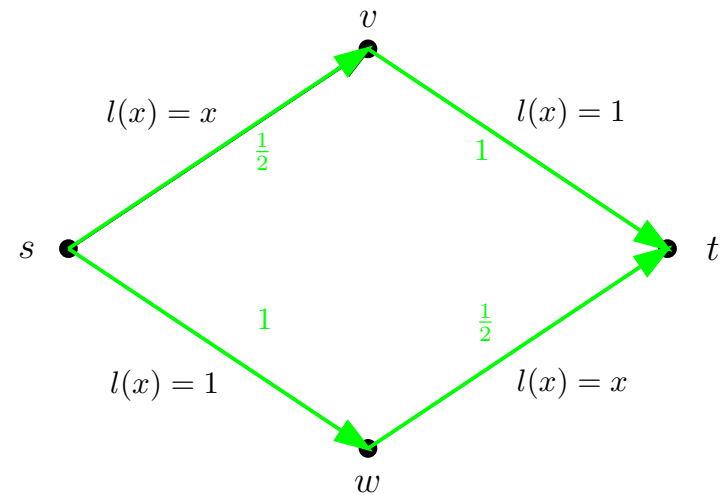


$$L(H^*, 1, l) = \frac{3}{2}$$

Braess's Paradox - Basic Example



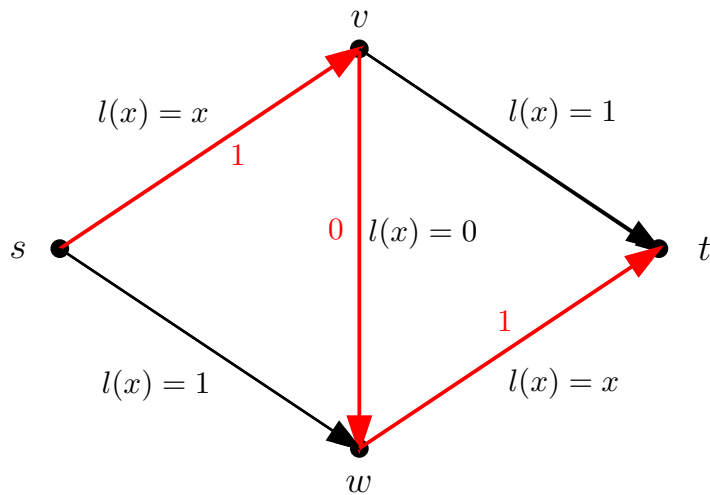
$$L(G, 1, l) = 2$$



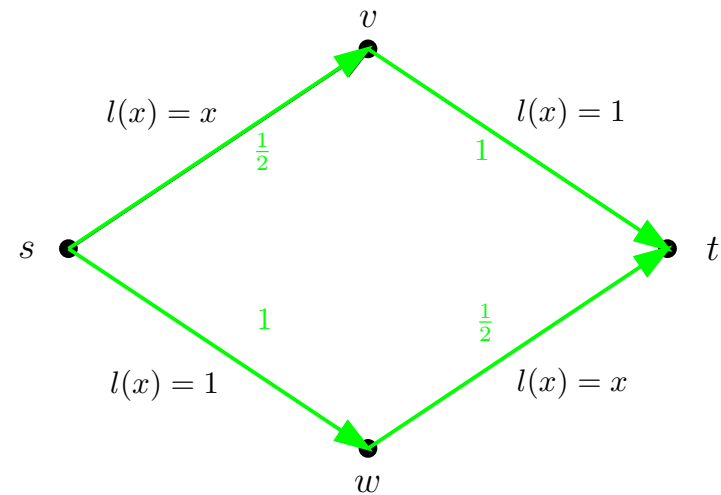
$$L(H^*, 1, l) = \frac{3}{2}$$

Braess's paradox: *Removing* (seemingly harmless) *edges* can decrease the common latency.

Braess's Paradox - Basic Example



$$L(G, 1, l) = 2$$



$$L(H^*, 1, l) = \frac{3}{2}$$

Braess's paradox: *Removing* (seemingly harmless) *edges* can decrease the common latency.

→ The trivial algorithm is not optimal.

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Let H^* be the subgraph of G that minimizes $L(H, r, l)$, and let f^* denote a Nash flow for H^* .

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Let H^* be the subgraph of G that minimizes $L(H, r, l)$, and let f^* denote a Nash flow for H^* . Since f^* is a feasible flow for G , we get

$$C(f) \leq \frac{4}{3} \cdot C(f^*)$$

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Let H^* be the subgraph of G that minimizes $L(H, r, l)$, and let f^* denote a Nash flow for H^* . Since f^* is a feasible flow for G , we get

$$C(f) \leq \frac{4}{3} \cdot C(f^*)$$

$$\Leftrightarrow r \cdot L(G, r, l) \leq \frac{4}{3} \cdot r \cdot L(H^*, r, l)$$

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Let H^* be the subgraph of G that minimizes $L(H, r, l)$, and let f^* denote a Nash flow for H^* . Since f^* is a feasible flow for G , we get

$$C(f) \leq \frac{4}{3} \cdot C(f^*)$$

$$\Leftrightarrow r \cdot L(G, r, l) \leq \frac{4}{3} \cdot r \cdot L(H^*, r, l)$$

$$\Leftrightarrow L(G, r, l) \leq \frac{4}{3} \cdot L(H^*, r, l)$$

Linear Latency Functions

Recall: For a Nash flow f and a feasible flow f^* for an instance (G, r, l) with linear latency functions, we have $C(f) \leq \frac{4}{3} \cdot C(f^*)$.

Let H^* be the subgraph of G that minimizes $L(H, r, l)$, and let f^* denote a Nash flow for H^* . Since f^* is a feasible flow for G , we get

$$C(f) \leq \frac{4}{3} \cdot C(f^*)$$

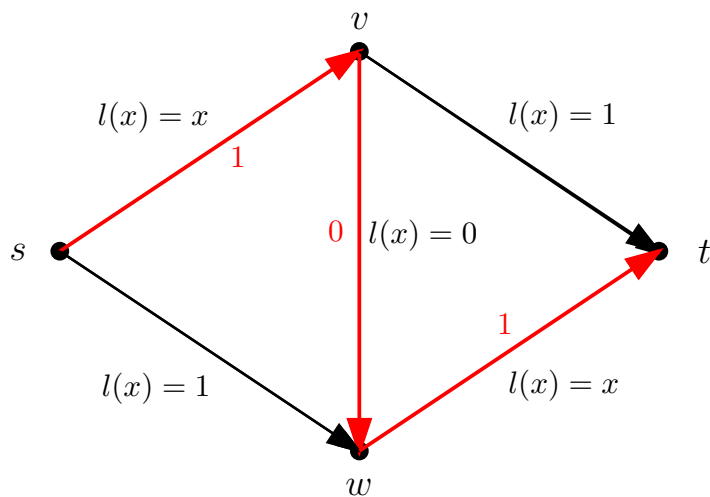
$$\Leftrightarrow r \cdot L(G, r, l) \leq \frac{4}{3} \cdot r \cdot L(H^*, r, l)$$

$$\Leftrightarrow L(G, r, l) \leq \frac{4}{3} \cdot L(H^*, r, l)$$

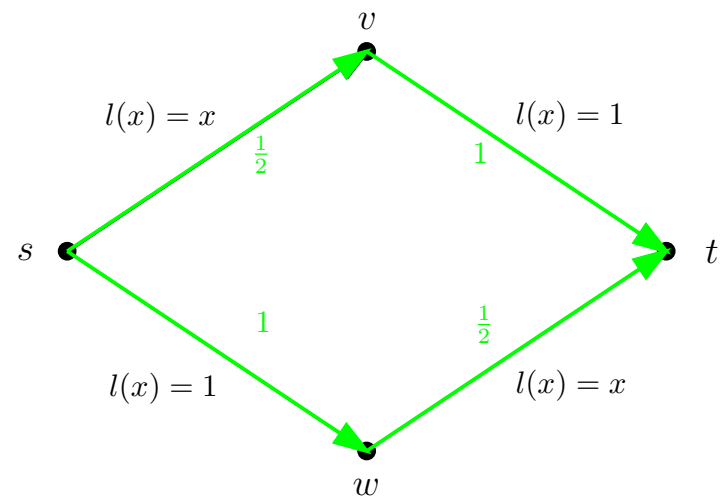
→ **Theorem.** *The trivial algorithm is a $\frac{4}{3}$ -approximation algorithm for instances with linear latency functions.*

Linear Latency Functions

Worst-case for the trivial algorithm: Braess's paradox basic example



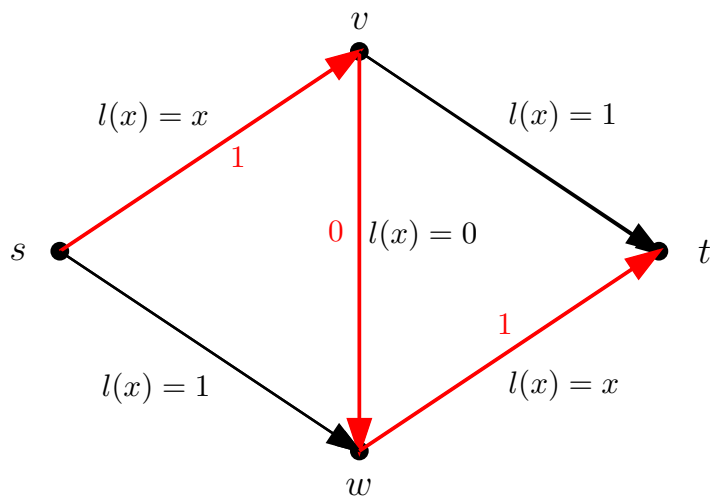
$$L(G, 1, l) = 2$$



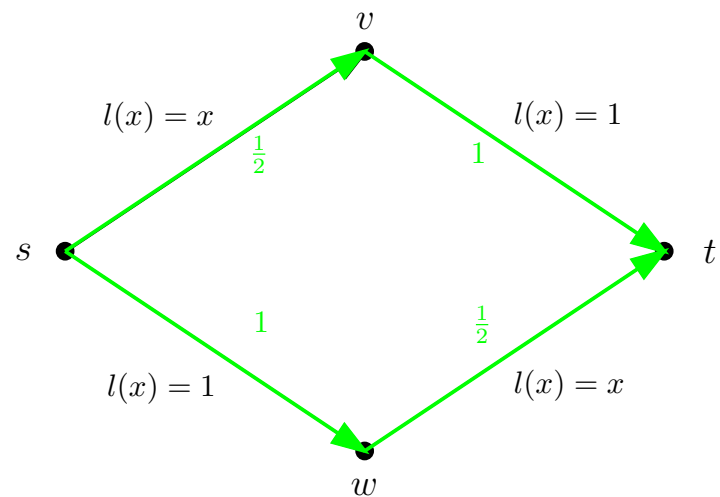
$$L(H^*, 1, l) = \frac{3}{2}$$

Linear Latency Functions

Worst-case for the trivial algorithm: Braess's paradox basic example



$$L(G, 1, l) = 2$$



$$L(H^*, 1, l) = \frac{3}{2}$$

$$L(G, 1, l) = 2 = \frac{4}{3} \cdot \frac{3}{2} = \frac{4}{3} \cdot L(H^*, 1, l)$$

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$?

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$? **No!**

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$? **No!**

Theorem. *For every $\epsilon > 0$, there is **no** $(\frac{4}{3} - \epsilon)$ -approximation algorithm for instances with linear latency functions, unless $P = NP$.*

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\frac{4}{3} - \epsilon)$ -**approximation algorithm** for instances with linear latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with linear latency functions is

• **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\frac{4}{3} - \epsilon)$ -**approximation algorithm** for instances with linear latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with linear latency functions is

- **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G
- **paradox-ridden** if $L(G, r, l) = \frac{4}{3} \cdot L(H, r, l)$ for some subgraph H of G .

Linear Latency Functions

Can we do better than $\gamma = \frac{4}{3}$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\frac{4}{3} - \epsilon)$ -**approximation algorithm** for instances with linear latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with linear latency functions is

- **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G
- **paradox-ridden** if $L(G, r, l) = \frac{4}{3} \cdot L(H, r, l)$ for some subgraph H of G .

Corollary. It is **NP-hard to distinguish between paradox-free and paradox-ridden instances.**

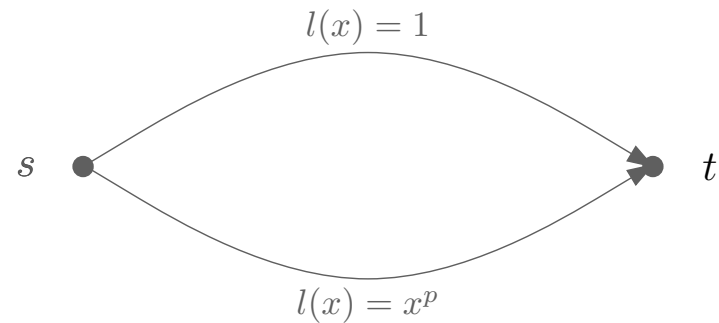
-
-
-

General Latency Functions

General Latency Functions

Recall: Pigou's example

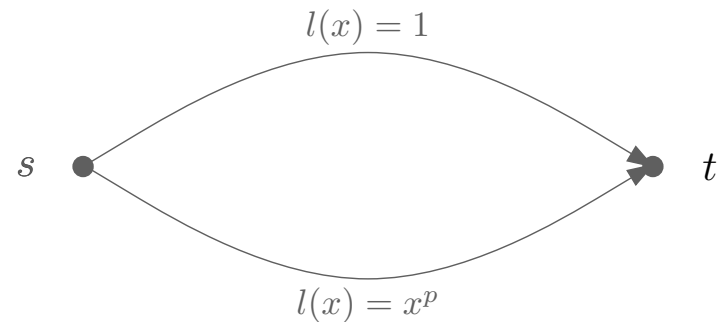
No upper bound on $\rho(G, r, l)$ for general latency functions.



General Latency Functions

Recall: Pigou's example

No upper bound on $\rho(G, r, l)$ for general latency functions.

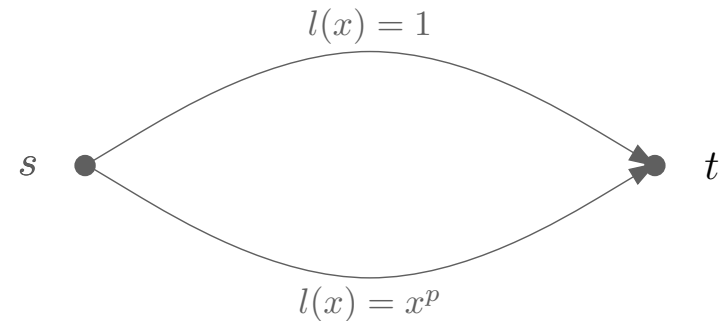


Bad news: Comparing the cost of a Nash flow and the cost of other feasible flows in G like in the linear case does not work.

General Latency Functions

Recall: Pigou's example

No upper bound on $\rho(G, r, l)$ for general latency functions.



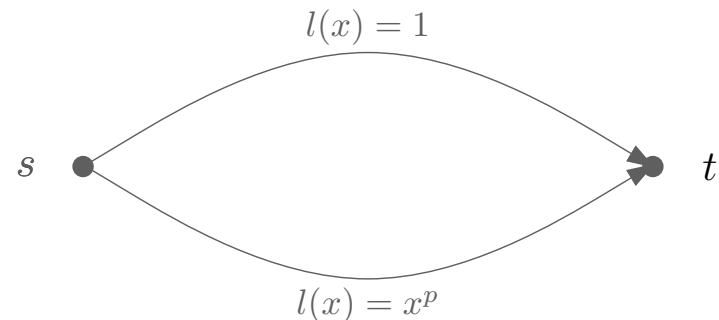
Bad news: Comparing the cost of a Nash flow and the cost of other feasible flows in G like in the linear case does not work.

New approach: Directly compare the common latency in G and the common latency in the subgraphs at equilibrium.

General Latency Functions

Recall: Pigou's example

No upper bound on $\rho(G, r, l)$ for general latency functions.



Bad news: Comparing the cost of a Nash flow and the cost of other feasible flows in G like in the linear case does not work.

New approach: Directly compare the common latency in G and the common latency in the subgraphs at equilibrium.

→ **Theorem.** *The trivial algorithm is an $\lfloor \frac{n}{2} \rfloor$ -approximation algorithm for instances (G, r, l) with general latency functions where G has n vertices.*

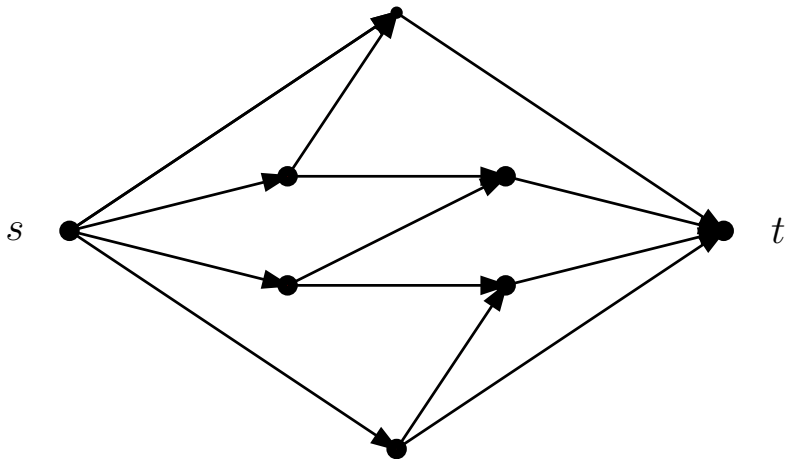
General Latency Functions

Worst-case for the trivial algorithm: Braess graphs B^k with $n = 2k + 2$ vertices

General Latency Functions

Worst-case for the trivial algorithm: Braess graphs B^k with $n = 2k + 2$ vertices

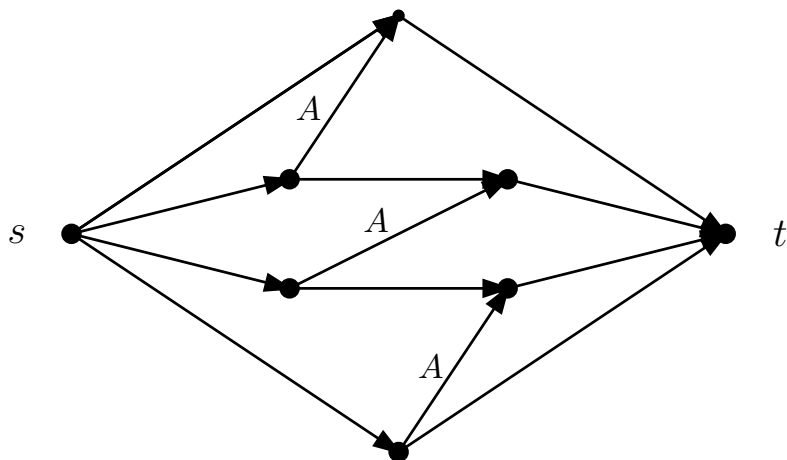
Example: B^3 with 8 vertices



General Latency Functions

Worst-case for the trivial algorithm: Braess graphs B^k with $n = 2k + 2$ vertices

Example: B^3 with 8 vertices



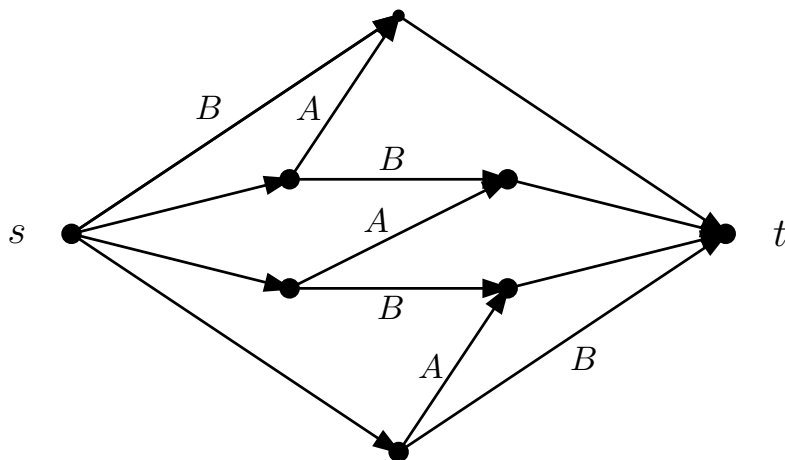
Latency Functions:

Type A : $l(x) = 0$

General Latency Functions

Worst-case for the trivial algorithm: Braess graphs B^k with $n = 2k + 2$ vertices

Example: B^3 with 8 vertices



Latency Functions:

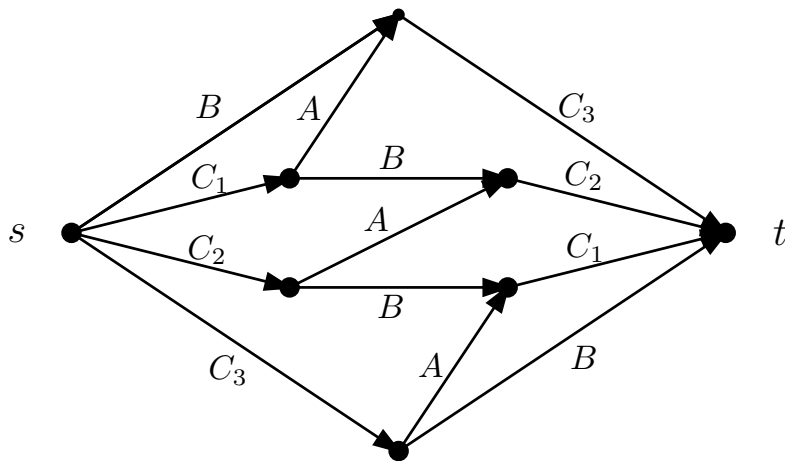
Type A : $l(x) = 0$

Type B : $l(x) = 1$

General Latency Functions

Worst-case for the trivial algorithm: Braess graphs B^k with $n = 2k + 2$ vertices

Example: B^3 with 8 vertices



Latency Functions:

Type A : $l(x) = 0$

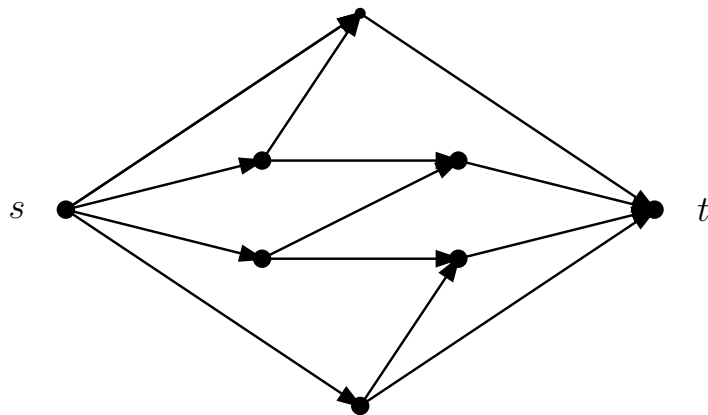
Type B : $l(x) = 1$

Type C_i : $l(x) = \begin{cases} 0 & x = \frac{k}{k+1} \\ i & x = 1 \end{cases}$

continuous and non-decreasing elsewhere

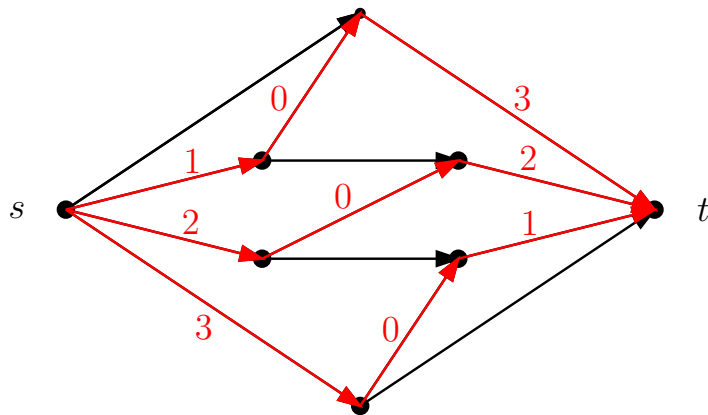
General Latency Functions

$$G = B^3$$



General Latency Functions

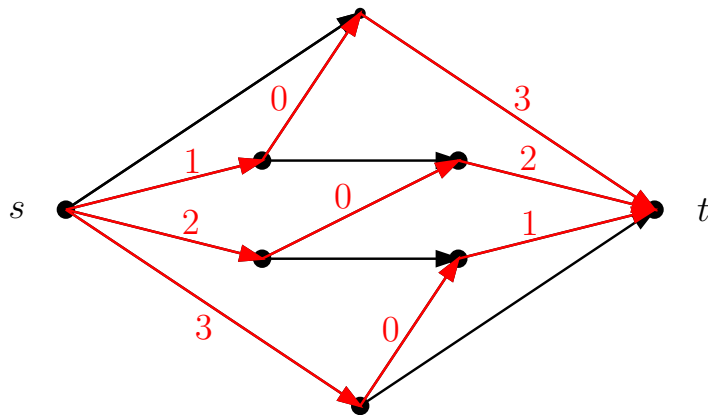
$$G = B^3$$



$$L(G, 3, l) = 4$$

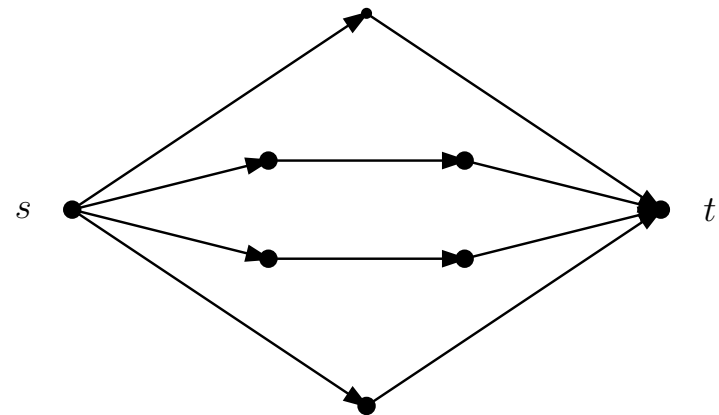
General Latency Functions

$G = B^3$



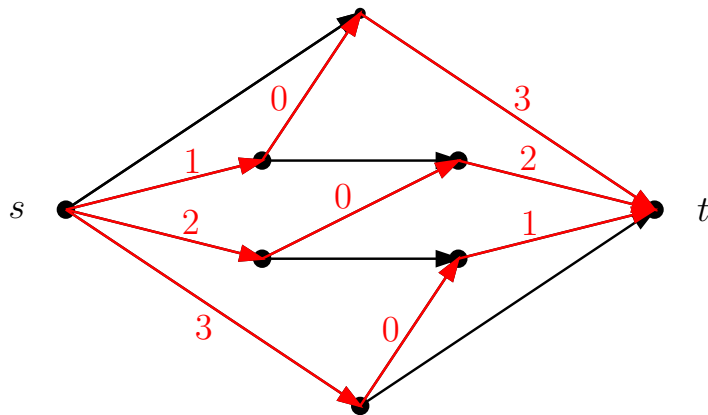
$$L(G, 3, l) = 4$$

$H^* = B^3$ without A -edges



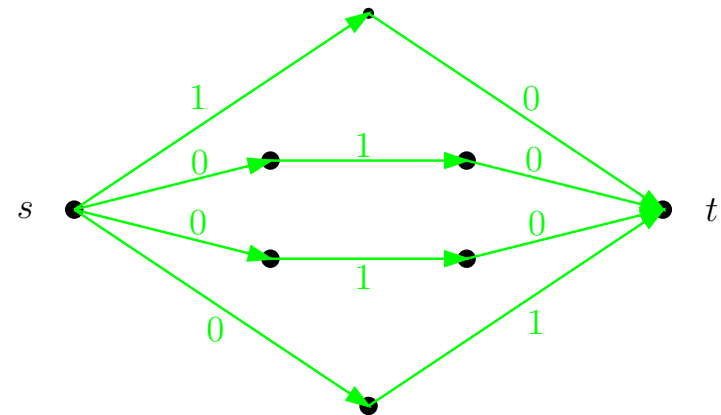
General Latency Functions

$G = B^3$



$$L(G, 3, l) = 4$$

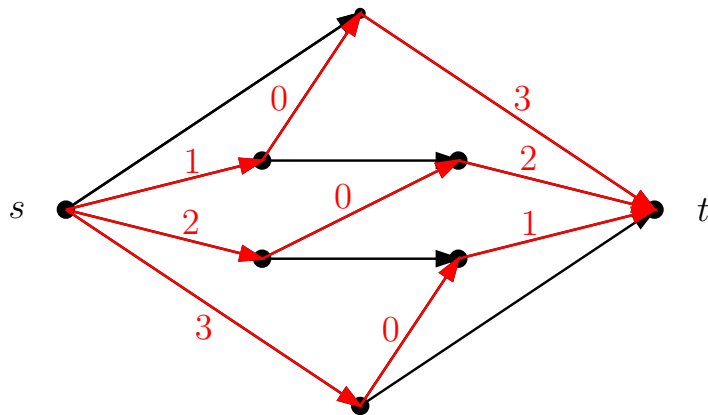
$H^* = B^3$ without A -edges



$$L(H^*, 3, l) = 1$$

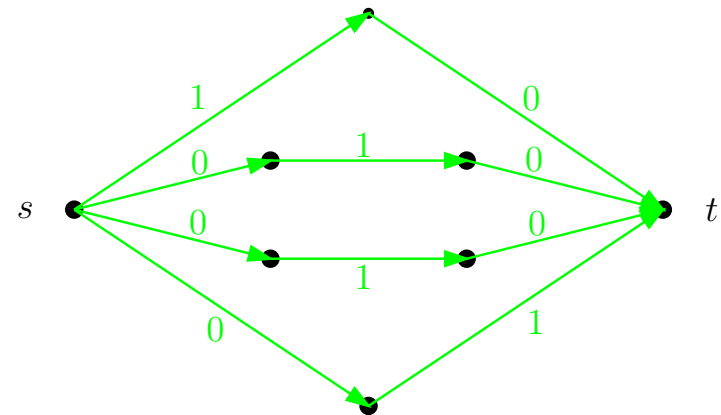
General Latency Functions

$G = B^3$



$$L(G, 3, l) = 4$$

$H^* = B^3$ without A -edges



$$L(H^*, 3, l) = 1$$

$$L(G, 3, l) = 4 = \lfloor \frac{8}{2} \rfloor \cdot 1 = \lfloor \frac{n}{2} \rfloor \cdot L(H^*, 3, l)$$

General Latency Functions

Observation. We removed 3 A -edges from G in order to improve the common latency by a factor of 4.

General Latency Functions

Observation. We removed 3 A -edges from G in order to improve the common latency by a factor of 4.

In general we have:

General Latency Functions

Observation. We removed 3 A -edges from G in order to improve the common latency by a factor of 4.

In general we have:

Theorem. *If H is obtained from G by removing k edges, then*

$$L(H, r, l) \geq \frac{1}{k+1} \cdot L(G, r, l)$$

General Latency Functions

Observation. We removed 3 A -edges from G in order to improve the common latency by a factor of 4.

In general we have:

Theorem. *If H is obtained from G by removing k edges, then*

$$L(H, r, l) \geq \frac{1}{k+1} \cdot L(G, r, l)$$

→ In order to improve the common latency by a factor greater than k , we need to remove at least k edges.

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$?

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$? **No!**

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$? **No!**

Theorem. For every $\epsilon > 0$, there is *no* $(\lfloor \frac{n}{2} \rfloor - \epsilon)$ -*approximation algorithm* for instances with general latency functions, unless $P = NP$.

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\lfloor \frac{n}{2} \rfloor - \epsilon)$ -**approximation algorithm** for instances with general latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with general latency functions is

• **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\lfloor \frac{n}{2} \rfloor - \epsilon)$ -**approximation algorithm** for instances with general latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with general latency functions is

- **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G
- **paradox-ridden** if $L(G, r, l) = \lfloor \frac{n}{2} \rfloor \cdot L(H, r, l)$ for some subgraph H of G .

General Latency Functions

Can we do better than $\gamma = \lfloor \frac{n}{2} \rfloor$? **No!**

Theorem. For every $\epsilon > 0$, there is **no** $(\lfloor \frac{n}{2} \rfloor - \epsilon)$ -**approximation algorithm** for instances with general latency functions, unless $P = NP$.

Definition. An instance (G, r, l) with general latency functions is

- **paradox-free** if $L(G, r, l) \leq L(H, r, l)$ for all subgraphs H of G
- **paradox-ridden** if $L(G, r, l) = \lfloor \frac{n}{2} \rfloor \cdot L(H, r, l)$ for some subgraph H of G .

Corollary. It is **NP-hard to distinguish between paradox-free and paradox-ridden instances.**

Summary

Latency functions	Best γ (for $P \neq NP$)	Trivial Algorithm	
		γ	Worst case example
linear	$\frac{4}{3}$	$\frac{4}{3}$	Braess's paradox basic example
general (continuous and non-decreasing)	$\lfloor \frac{n}{2} \rfloor$	$\lfloor \frac{n}{2} \rfloor$	Braess graphs B^k

Summary

Latency functions	Best γ (for $P \neq NP$)	Trivial Algorithm	
		γ	Worst case example
linear	$\frac{4}{3}$	$\frac{4}{3}$	Braess's paradox basic example
general (continuous and non-decreasing)	$\lfloor \frac{n}{2} \rfloor$	$\lfloor \frac{n}{2} \rfloor$	Braess graphs B^k

For polynomial latency functions of degree $p \geq 2$, a sharper bound can be proved:

Summary

Latency functions	Best γ (for $P \neq NP$)	Trivial Algorithm	
		γ	Worst case example
linear	$\frac{4}{3}$	$\frac{4}{3}$	Braess's paradox basic example
general (continuous and non-decreasing)	$\lfloor \frac{n}{2} \rfloor$	$\lfloor \frac{n}{2} \rfloor$	Braess graphs B^k

For polynomial latency functions of degree $p \geq 2$, a sharper bound can be proved:

polynomials of degree p	$c_1 \frac{p}{\ln p}$ for a constant $c_1 > 0$	$c_2 \frac{p}{\ln p}$ for a constant $c_2 > 0$	modified Braess graphs \tilde{B}^k
------------------------------	--	--	---

Summary

Latency functions	Best γ (for $P \neq NP$)	Trivial Algorithm	
		γ	Worst case example
linear	$\frac{4}{3}$	$\frac{4}{3}$	Braess's paradox basic example
general (continuous and non-decreasing)	$\lfloor \frac{n}{2} \rfloor$	$\lfloor \frac{n}{2} \rfloor$	Braess graphs B^k

For polynomial latency functions of degree $p \geq 2$, a sharper bound can be proved:

polynomials of degree p	$c_1 \frac{p}{\ln p}$ for a constant $c_1 > 0$	$c_2 \frac{p}{\ln p}$ for a constant $c_2 > 0$	modified Braess graphs \tilde{B}^k
------------------------------	--	--	---

→ *Building all candidate edges yields a (nearly) optimal approximation algorithm.*



Stackelberg Strategies

Coping with Selfishness II

In many systems: mix of

- "selfishly controlled" traffic
- "centrally controlled" traffic

~> *How should centrally controlled traffic be routed to induce "good" behaviour from the noncooperative users?*

Stackelberg game: the model

- **one player = *leader***
 - strategy for routing the centrally controlled flow (remains fixed)
 - goal: minimizing total latency
- **other players = *followers***
 - choose personal strategy w.r.t. the leader's strategy
 - goal: minimizing personal latency

Stackelberg game: the model

Definition: **Stackelberg instance**

Instance (G, r, ℓ, β) where

- $G = (V, E)$ network of parallel links (one s - t -pair)
- r rate of flow
- $\ell = \{\ell_e | e \in E\}$ set of latency functions
- $\beta \in [0, 1]$ fraction of centrally controlled flow

Stackelberg game: the model

Definitions

- **Stackelberg strategy**

Centrally controlled flow f , feasible for (G, r, ℓ) .

- **Induced equilibrium**

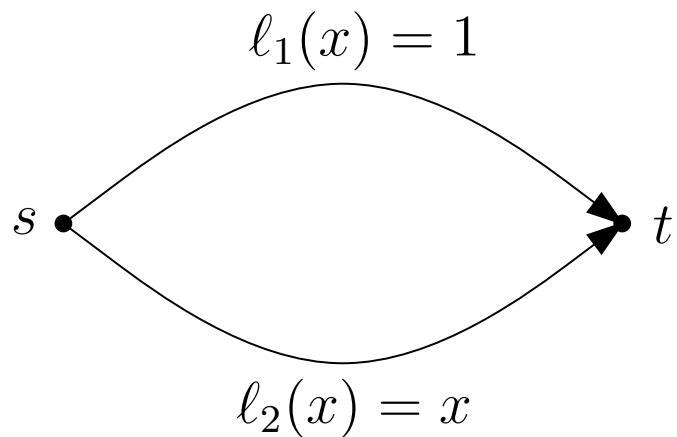
Flow g of selfish users.

(is a Nash flow w.r.t. $\tilde{\ell}_e(x) = \ell_e(f_e + x) \rightsquigarrow$ all selfish users experience the same latency)

- **Induced flow** $= f + g$

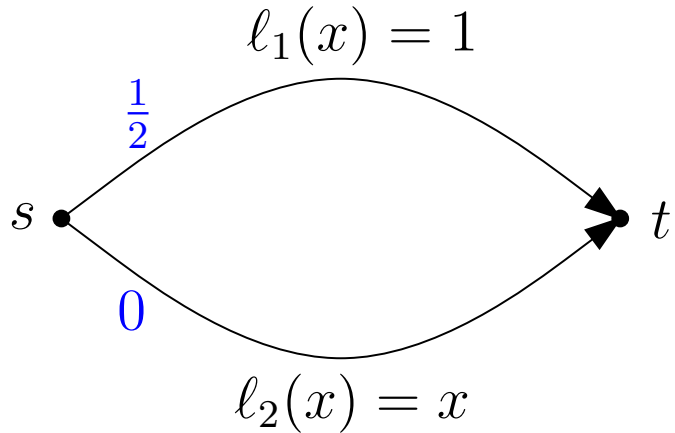
Two examples for $\beta = \frac{1}{2}$

Pigou's example



Two examples for $\beta = \frac{1}{2}$

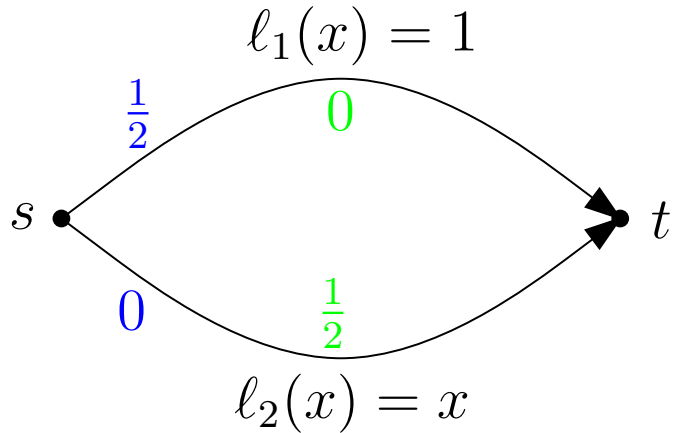
Pigou's example



strategy f

Two examples for $\beta = \frac{1}{2}$

Pigou's example

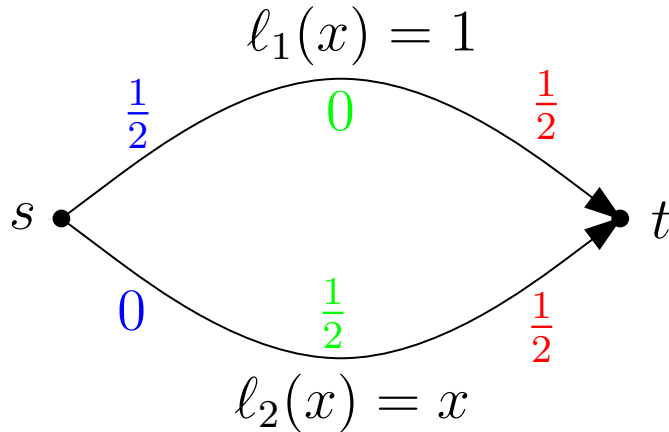


strategy f

induced equilibrium g

Two examples for $\beta = \frac{1}{2}$

Pigou's example



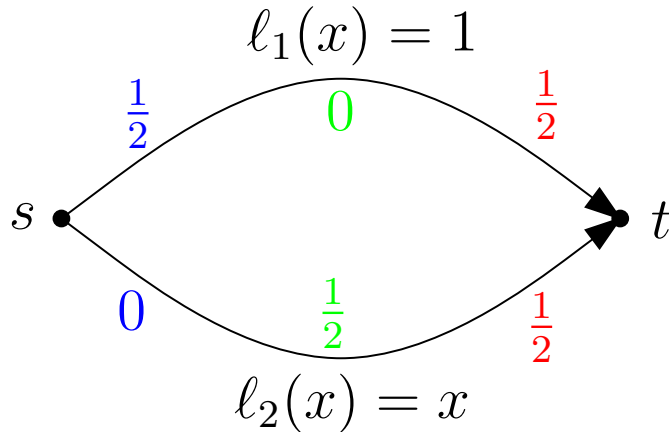
strategy f

induced equilibrium g

induced flow $f + g$

Two examples for $\beta = \frac{1}{2}$

Pigou's example

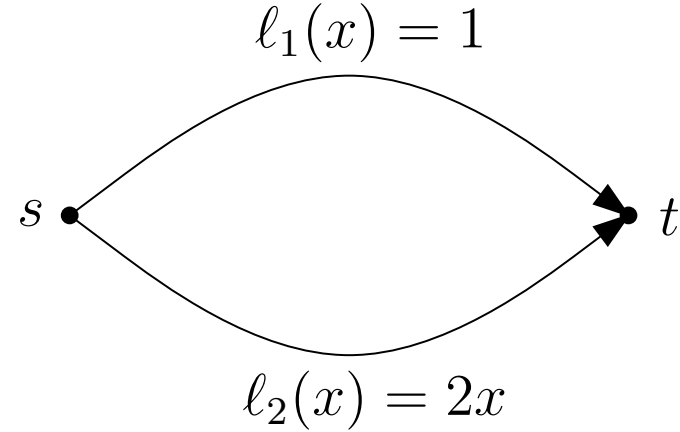


strategy f

induced equilibrium g

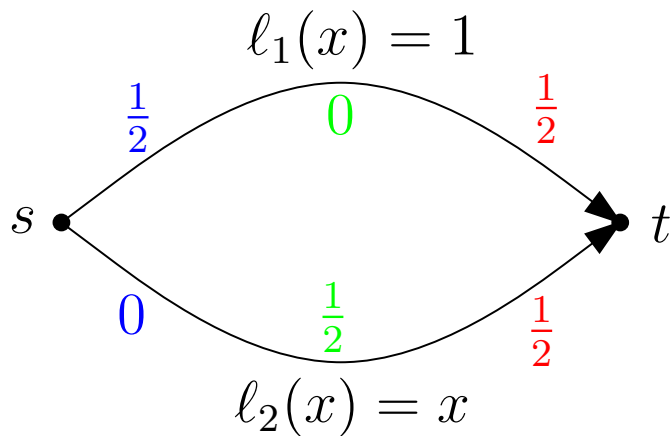
induced flow $f + g$

variation of Pigou



Two examples for $\beta = \frac{1}{2}$

Pigou's example

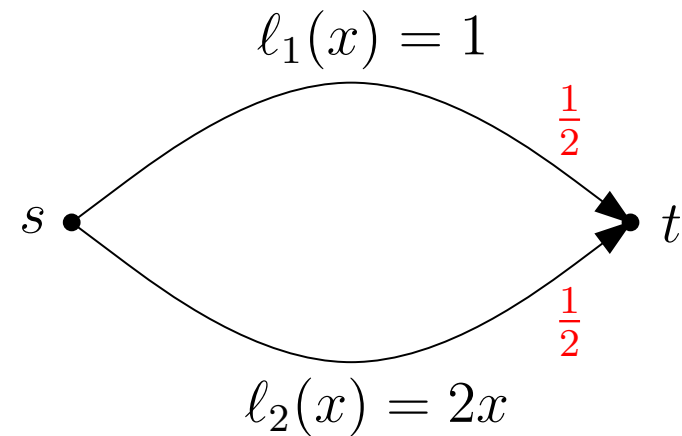


strategy f

induced equilibrium g

induced flow $f + g$

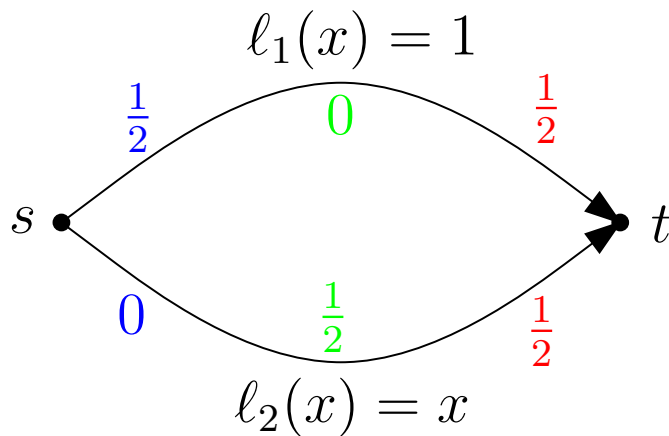
variation of Pigou



flow induced by any strategy

Two examples for $\beta = \frac{1}{2}$

Pigou's example

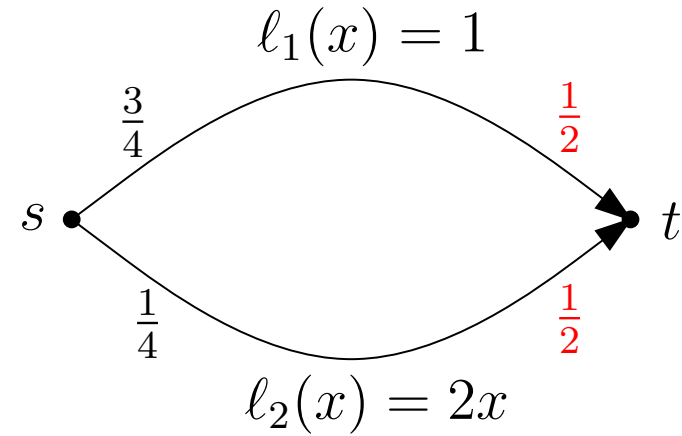


strategy f

induced equilibrium g

induced flow $f + g$

variation of Pigou

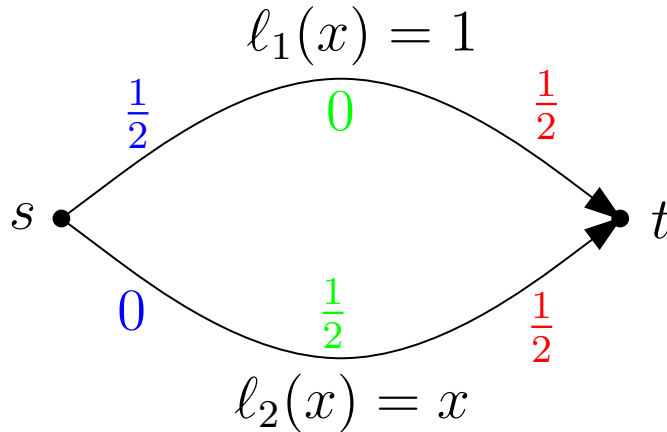


optimal flow

flow induced by any strategy

Two examples for $\beta = \frac{1}{2}$

Pigou's example

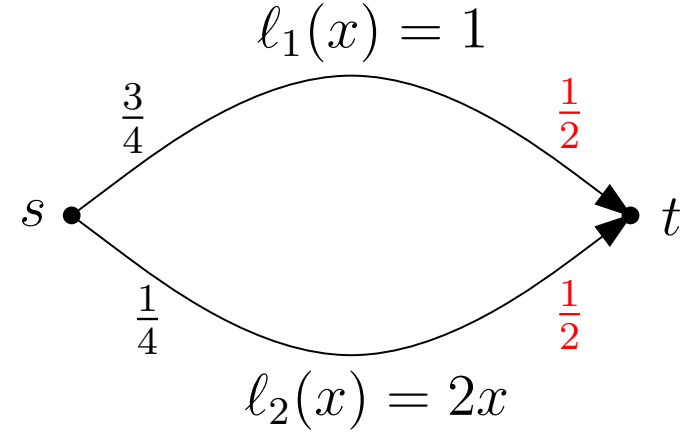


strategy f

induced equilibrium g

induced flow $f + g$

variation of Pigou



optimal flow

flow induced by any strategy

\Rightarrow not optimal

Basic questions

- What's a good / the best strategy?
- Can we compute it?
- How (in)efficient is the induced equilibrium?

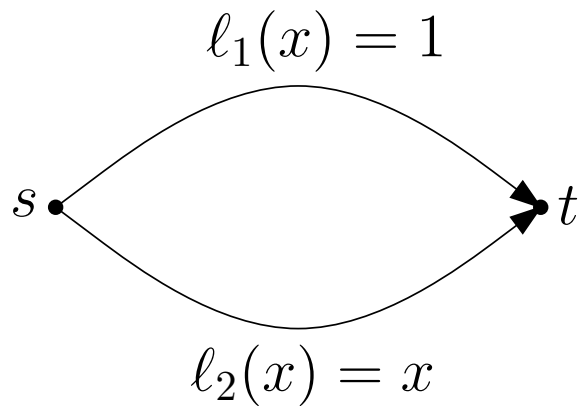
The Aloof Strategy

To get a strategy f for instance (G, r, ℓ, β)

- compute the minimum latency flow f^* for $(G, \beta \cdot r, \ell)$
- put $f = f^*$

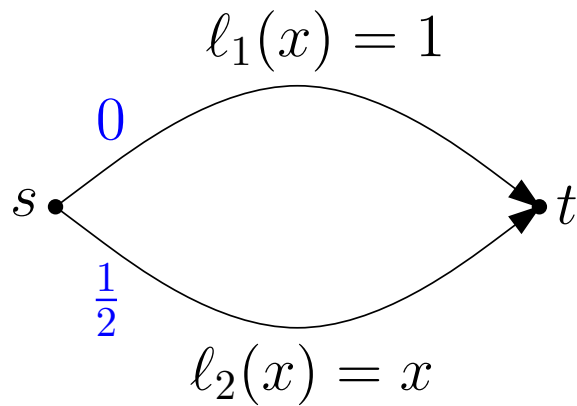
The Aloof Strategy

performs poorly: example for $\beta = \frac{1}{2}$



The Aloof Strategy

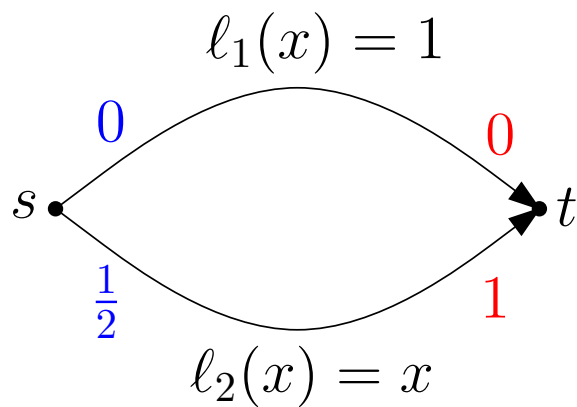
performs poorly: example for $\beta = \frac{1}{2}$



optimal flow for $(G, \frac{1}{2}, \ell) =$ strategy f

The Aloof Strategy

performs poorly: example for $\beta = \frac{1}{2}$

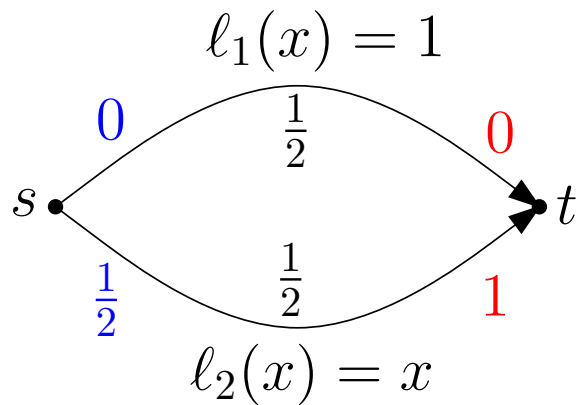


optimal flow for $(G, \frac{1}{2}, \ell) =$ strategy f

induced flow for $(G, 1, \ell) \rightsquigarrow$ cost: 1

The Aloof Strategy

performs poorly: example for $\beta = \frac{1}{2}$



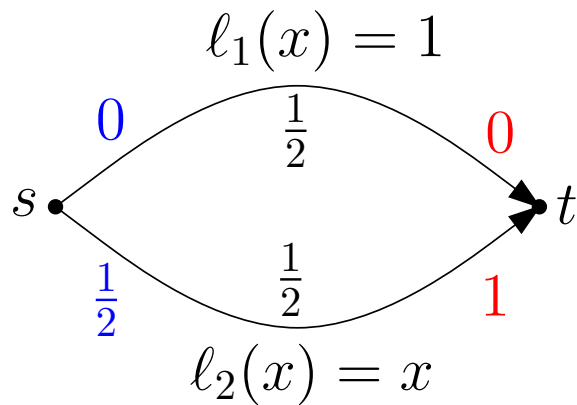
optimal flow for $(G, \frac{1}{2}, \ell) =$ strategy f

induced flow for $(G, 1, \ell) \rightsquigarrow$ cost: 1

optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{3}{4}$

The Aloof Strategy

performs poorly: example for $\beta = \frac{1}{2}$



optimal flow for $(G, \frac{1}{2}, \ell) =$ strategy f

induced flow for $(G, 1, \ell) \rightsquigarrow$ cost: 1

optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{3}{4}$

\rightsquigarrow arbitrarily bad for nonlinear latency functions (e.g. for $\ell_2(x) = x^p, p \in \mathbb{N}^+$)

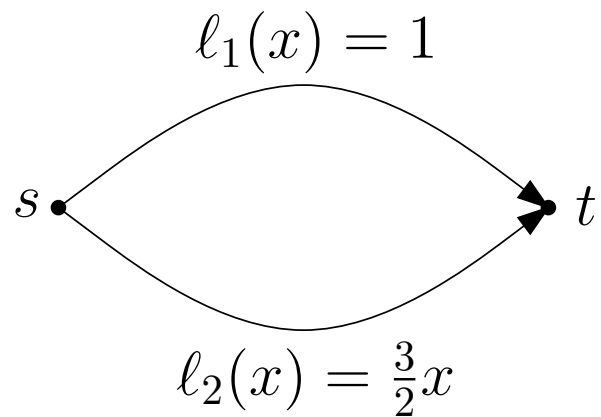
The Scale Strategy

To get a strategy f for instance (G, r, ℓ, β)

- compute a minimum latency flow f^* for (G, r, ℓ)
- put $f = \beta \cdot f^*$

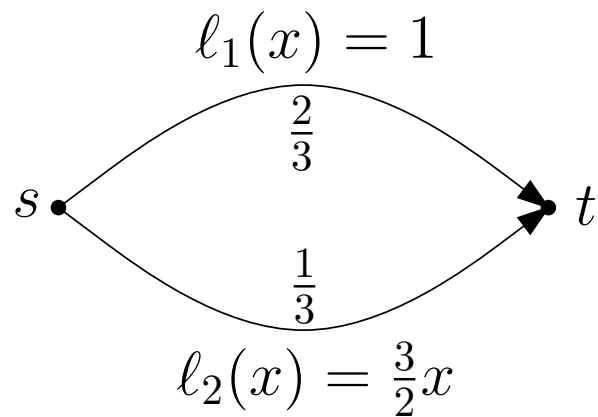
The Scale Strategy

performs poorly: example for $\beta = \frac{1}{2}$



The Scale Strategy

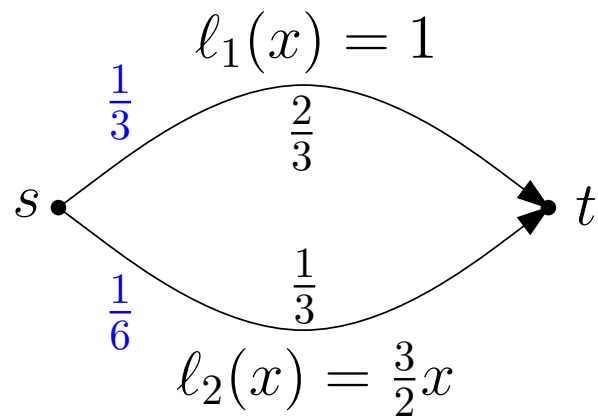
performs poorly: example for $\beta = \frac{1}{2}$



optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{5}{6}$

The Scale Strategy

performs poorly: example for $\beta = \frac{1}{2}$

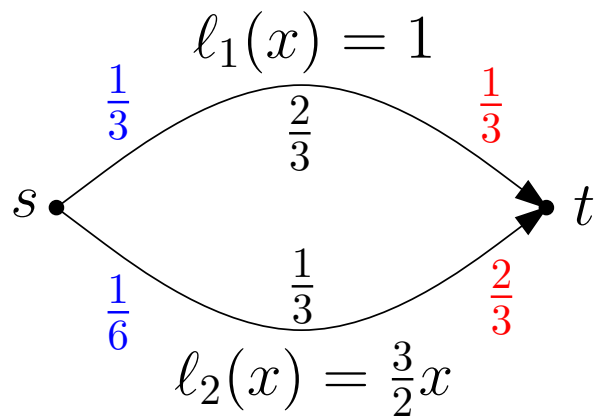


optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{5}{6}$

strategy f

The Scale Strategy

performs poorly: example for $\beta = \frac{1}{2}$



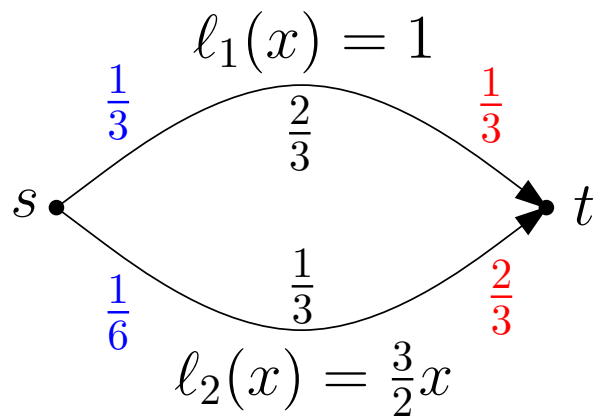
optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{5}{6}$

strategy f

induced flow for $(G, 1, \ell) \rightsquigarrow$ cost: 1

The Scale Strategy

performs poorly: example for $\beta = \frac{1}{2}$



optimal flow for $(G, 1, \ell) \rightsquigarrow$ cost: $\frac{5}{6}$

strategy f

induced flow for $(G, 1, \ell) \rightsquigarrow$ cost: 1

better strategy: route all centrally controlled flow on the upper edge

\rightsquigarrow induced flow $(\frac{1}{2}, \frac{1}{2})$ has cost $\frac{7}{8}$



Insights

What have we learned?

Insights

What have we learned?

- avoid edges that selfish users will use anyway

Insights

What have we learned?

- avoid edges that selfish users will use anyway
- use edges that are not very attractive to selfish users, i.e. edges with high latency

Largest Latency First (LLF) Strategy

For instance (G, r, ℓ, β)

1. compute a min latency flow f^* for (G, r, ℓ)
2. label the edges of G from 1 to $m = |E|$ so that $\ell_1(f_1^*) \leq \dots \leq \ell_m(f_m^*)$
3. let $k \leq m$ be minimal with $\sum_{i=k+1}^m f_i^* \leq \beta r$
4. put

$$f_i = \begin{cases} f_i^* & i > k \\ \beta r - \sum_{i=k+1}^m f_i^* & i = k \\ 0 & i < k \end{cases}$$

Largest Latency First (LLF) Strategy

The LLF strategy

- can be computed in polynomial time
- always induces a flow with near-optimal total latency

Performance guarantee I

Theorem

In a network of parallel links and arbitrary (nondecreasing, differentiable) latency functions the LLF strategy always induces a flow with cost no more than $\frac{1}{\beta}$ times the cost of a minimum latency flow.

Performance guarantee I

Theorem

In a network of parallel links and arbitrary (nondecreasing, differentiable) latency functions the LLF strategy always induces a flow with cost no more than $\frac{1}{\beta}$ times the cost of a minimum latency flow.

~> price of anarchy can now be bounded for general latency functions!

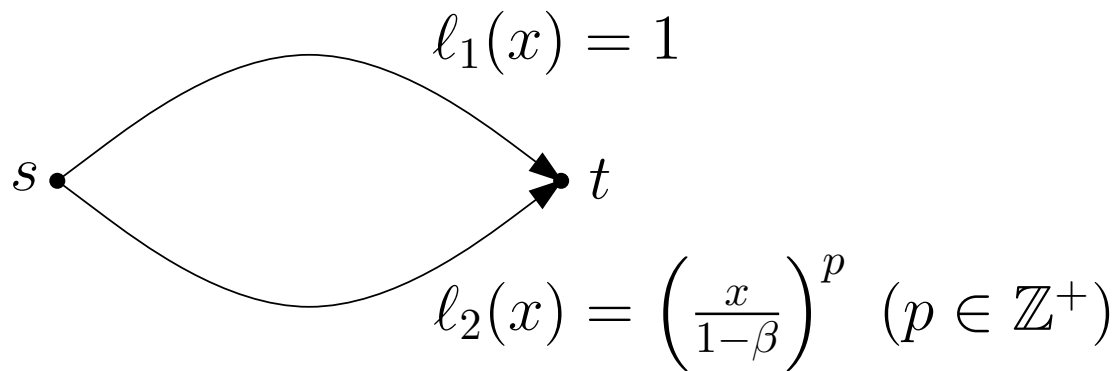
Performance guarantee II

Theorem

In a network of parallel links and linear (nondecreasing, differentiable) latency functions the LLF strategy always induces a flow with cost no more than $\frac{4}{3+\beta}$ times the cost of a minimum latency flow.

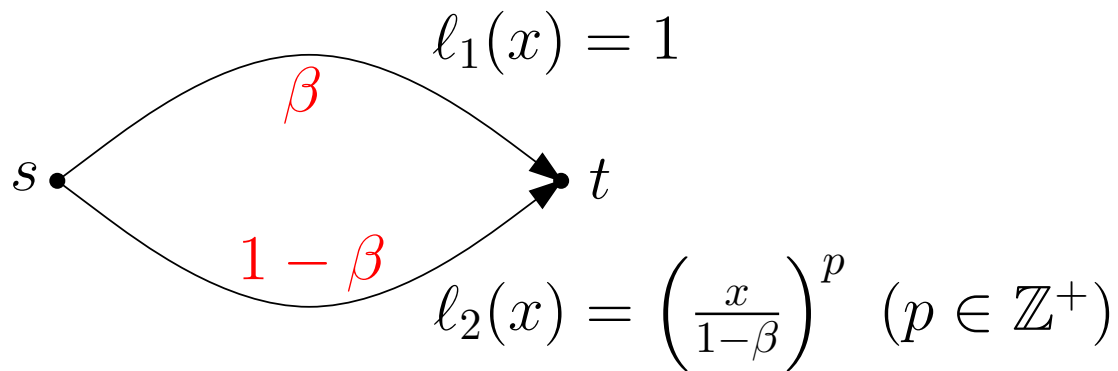
Performance guarantee

best possible worst case guarantee:



Performance guarantee

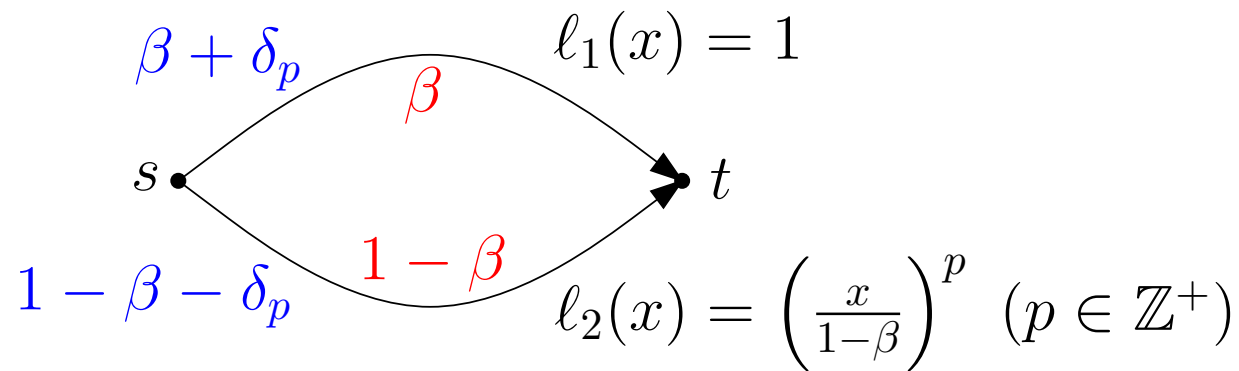
best possible worst case guarantee:



cost of induced flow (any strategy): 1

Performance guarantee

best possible worst case guarantee:



cost of induced flow (any strategy): 1

cost of optimal flow: $(\beta + \delta_p) + (1 - \beta - \delta_p) \left(\frac{1 - \beta - \delta_p}{1 - \beta}\right)^p \xrightarrow[\delta_p \rightarrow 0]{p \rightarrow \infty} \beta$

Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Optimal strategy

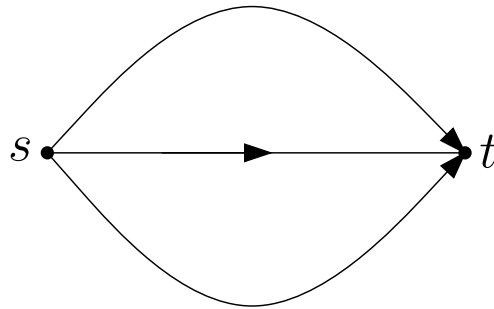
Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:

$$\ell_1(x) = x$$

$$\ell_2(x) = 1 + x$$

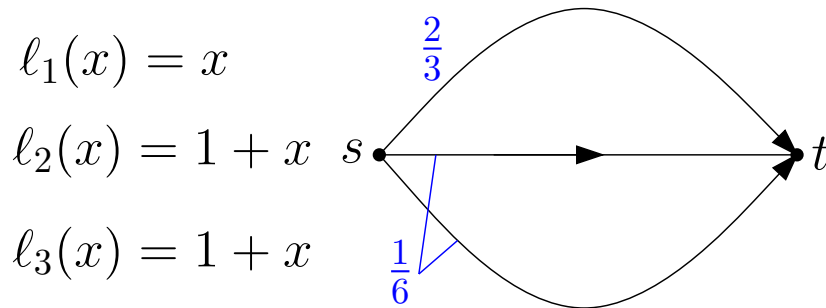
$$\ell_3(x) = 1 + x$$



Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:

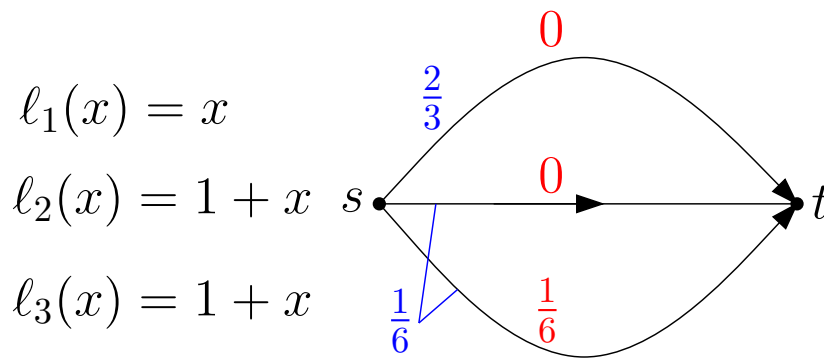


optimal flow

Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:

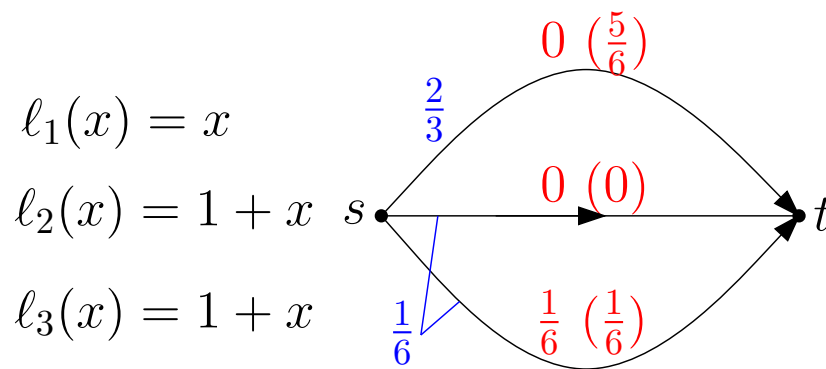


optimal flow
LLF strategy

Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:



cost: $\frac{8}{9}$

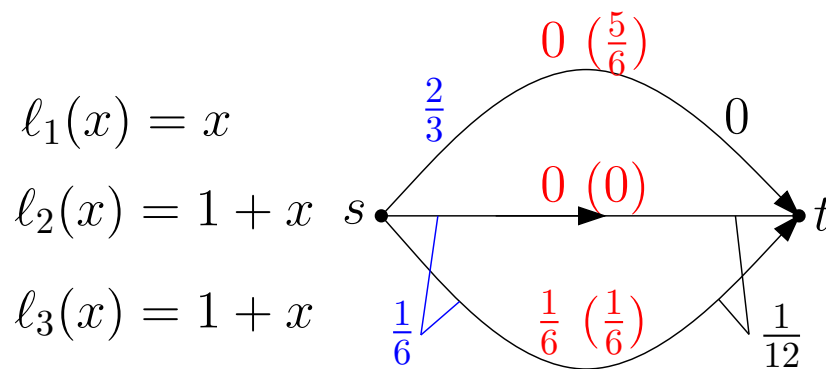
optimal flow

LLF strategy (induced flow)

Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:



cost: $\frac{8}{9}$

optimal flow

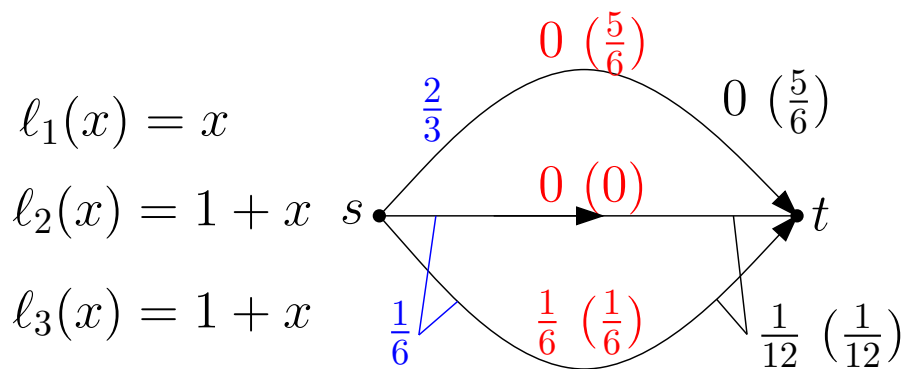
LLF strategy (induced flow)

better strategy

Optimal strategy

Is the LLF strategy optimal (i.e. always inducing a flow with minimal cost within all flows induced by any strategy)?

Unfortunately not:



cost: $\frac{8}{9}$

optimal flow

LLF strategy (induced flow)

better strategy (induced flow)

cost: $\frac{7}{8}$

Optimal strategy

Can we compute the optimal Stackelberg strategy in polynomial time?

Optimal strategy

Can we compute the optimal Stackelberg strategy in polynomial time?

NO!

Theorem

Computing the optimal Stackelberg strategy is \mathcal{NP} -hard, even for networks of parallel links and with linear latency functions.

•
•
•



Thank you!

schwarze@mathematik.uni-kl.de

dirk.stegemann@uni-mannheim.de

swagner@ira.uka.de