NETWORK DESIGN AND THE BRAESS PARADOX

Algorithmic Game Theory

Corelab
E.C.E - N.T.U.A.

April 14, 2011

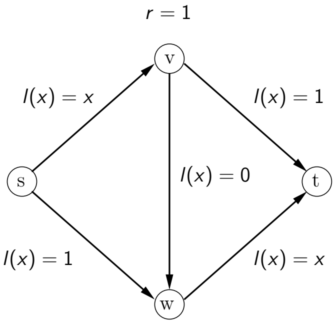
Outline

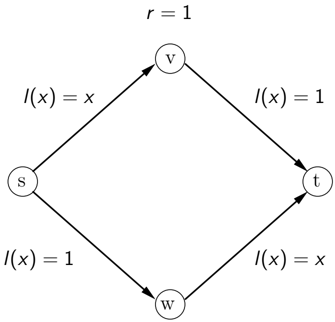
Outline

Selfish Routing

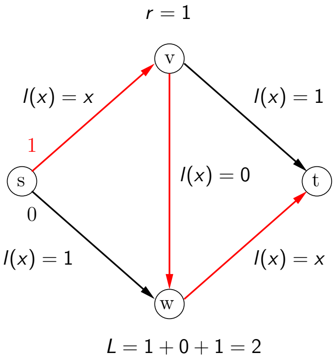
- Problem: route traffic in a network of selfish non-cooperative players.
- Motivation: simple examples show that Nash equilibria can be inefficient (Price of Anarchy).
- Question: which subnetwork will exhibit the best performance when used selfishly?

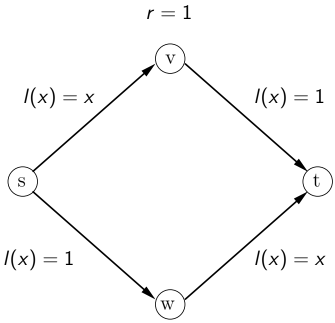
Braess's Paradox



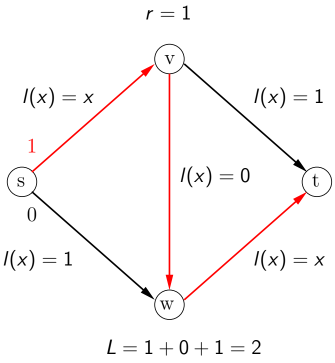


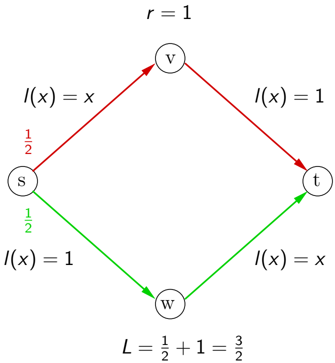
Braess's Paradox





Braess's Paradox





The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P : e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P: e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P : e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P : e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P:e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P: e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P: e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P : e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

The model

- Directed network $G = (V, E)$.
- Source s and destination t.
- Latency function $l_e : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$. We assume that l_e's are continuous and non-decreasing.
- Traffic r, caused by an infinite population of players, each trying to route a negligible amount of traffic through an $s - t$ path.
- Let $\mathcal{P}$ be the set of simple $s - t$ paths. Then, a flow f is non-negative vector indexed by $\mathcal{P}$.
- Feasible flow f: $\sum_{P \in \mathcal{P}} f_P = r$.
- Flow on edges $f_e = \sum_{P: e \in P} f_P$.
- Latency of a path P: $l_P(f) = \sum_{e \in P} l_e(f_e)$.
- Cost of feasible flow f: $C(f) = \sum_{P \in \mathcal{P}} l_P(f) f_P$.

Flows at Nash Equilibrium (1 / 2)

Intuitively, each unit of flow travels along the minimum-latency path in a NE.

Definition

A flow f feasible for (G, r, l) is at Nash equilibrium, or is a Nash (or Wardrop) flow, if for all $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$ and $\delta \in (0, f_{P1}]$, we have

$$l_{P_1}(f) \leq l_{P_2}(\tilde{f}),$$

where

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

Flows at Nash Equilibrium (1 / 2)

Intuitively, each unit of flow travels along the minimum-latency path in a NE.

Definition

A flow f feasible for (G, r, l) is at Nash equilibrium, or is a Nash (or Wardrop) flow, if for all $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$ and $\delta \in (0, f_{P1}]$, we have

$$l_{P_1}(f) \leq l_{P_2}(\tilde{f}),$$

where

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

Flows at Nash Equilibrium (2 / 2)

Theorem

A flow f feasible for (G, r, l) is at Nash equilibrium iff for every $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$,

$$l_{P_1}(f) \leq l_{P_2}(f).$$

Thus, all $s - t$ paths in NE with positive flow have **equal** latency, denoted by $L(G, r, l)$.

Moreover, flows at NE always exist and are **unique** with respect to $L(G, r, l)$.

Finally, there exists a directed acyclic Nash flow.

Flows at Nash Equilibrium (2 / 2)

Theorem

A flow f feasible for (G, r, l) is at Nash equilibrium iff for every $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$,

$$l_{P_1}(f) \leq l_{P_2}(f).$$

Thus, all $s - t$ paths in NE with positive flow have **equal** latency, denoted by $L(G, r, l)$.

Moreover, flows at NE always exist and are **unique** with respect to $L(G, r, l)$.

Finally, there exists a directed acyclic Nash flow.

Flows at Nash Equilibrium (2 / 2)

Theorem

A flow f feasible for (G, r, l) is at Nash equilibrium iff for every $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$,

$$l_{P_1}(f) \leq l_{P_2}(f).$$

Thus, all $s - t$ paths in NE with positive flow have **equal** latency, denoted by $L(G, r, l)$.

Moreover, flows at NE always exist and are **unique** with respect to $L(G, r, l)$.

Finally, there exists a directed acyclic Nash flow.

Flows at Nash Equilibrium (2 / 2)

Theorem

A flow f feasible for (G, r, l) is at Nash equilibrium iff for every $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$,

$$l_{P_1}(f) \leq l_{P_2}(f).$$

Thus, all $s - t$ paths in NE with positive flow have **equal** latency, denoted by $L(G, r, l)$.

Moreover, flows at NE always exist and are **unique** with respect to $L(G, r, l)$.

Finally, there exists a directed acyclic Nash flow.

Formalizing our Problem

Problem

Given an instance (G, r, l), find a subgraph H of G that minimizes $L(H, r, l)$.

Properties of Nash Flows

Lemma

For every instance (G, r, l), $L(G, r, l)$ is a non-decreasing function of r.

Lemma

Let f be a flow feasible for (G, r, l). For a vertex v in G, let $d(v)$ denote the length, with respect to edge lengths $\{l_e(f_e)\}_{e \in E}$ of a shortest $s - v$ path in G. Then f is at Nash equilibrium iff

$$d(w) - d(v) \leq l_e(f_e)$$

for all edges $e = (v, w)$, with equality holding whenever $f_e > 0$.

Lemma

If f is a flow at NE for (G, r, l), then $C(f) = r \cdot L(G, r, l)$.

Properties of Nash Flows

Lemma

For every instance (G, r, l), $L(G, r, l)$ is a non-decreasing function of r.

Lemma

Let f be a flow feasible for (G, r, l). For a vertex v in G, let $d(v)$ denote the length, with respect to edge lengths $\{l_e(f_e)\}_{e \in E}$ of a shortest $s - v$ path in G. Then f is at Nash equilibrium iff

$$d(w) - d(v) \leq l_e(f_e)$$

for all edges $e = (v, w)$, with equality holding whenever $f_e > 0$.

Lemma

If f is a flow at NE for (G, r, l), then $C(f) = r \cdot L(G, r, l)$.

Properties of Nash Flows

Lemma

For every instance (G, r, l), $L(G, r, l)$ is a non-decreasing function of r.

Lemma

Let f be a flow feasible for (G, r, l). For a vertex v in G, let $d(v)$ denote the length, with respect to edge lengths $\{l_e(f_e)\}_{e \in E}$ of a shortest $s - v$ path in G. Then f is at Nash equilibrium iff

$$d(w) - d(v) \leq l_e(f_e)$$

for all edges $e = (v, w)$, with equality holding whenever $f_e > 0$.

Lemma

If f is a flow at NE for (G, r, l), then $C(f) = r \cdot L(G, r, l)$.

Outline

Linear Latency Functions

We consider latency functions of the form $l_e(x) = a_e x + b_e$, $a_e, b_e \geq 0$. We then call the problem the LINEAR LATENCY NETWORK DESIGN. It is known that the price of anarchy in such networks is at most $\frac{4}{3}$.

Algorithm (Trivial Algorithm)

Given an instance (G, r, l), build the whole network G.

Lemma (Roughgarden - Tardos)

Let f^* and f be feasible and Nash flows, respectively, for an instance (G, r, l) with linear latency functions. Then,

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

Linear Latency Functions

We consider latency functions of the form $l_e(x) = a_e x + b_e$, $a_e, b_e \geq 0$. We then call the problem the LINEAR LATENCY NETWORK DESIGN. It is known that the price of anarchy in such networks is at most $\frac{4}{3}$.

Algorithm (Trivial Algorithm)

Given an instance (G, r, l), build the whole network G.

Lemma (Roughgarden - Tardos)

Let f^* and f be feasible and Nash flows, respectively, for an instance (G, r, l) with linear latency functions. Then,

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

Linear Latency Functions

We consider latency functions of the form $l_e(x) = a_e x + b_e$, $a_e, b_e \geq 0$. We then call the problem the LINEAR LATENCY NETWORK DESIGN. It is known that the price of anarchy in such networks is at most $\frac{4}{3}$.

Algorithm (Trivial Algorithm)

Given an instance (G, r, l), build the whole network G.

Lemma (Roughgarden - Tardos)

Let f^* and f be feasible and Nash flows, respectively, for an instance (G, r, l) with linear latency functions. Then,

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

The trivial algorithm performance for LINEAR LATENCY NETWORK DESIGN

Corollary

The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN.

Proof.

- Let H be the subgraph that minimizes $L(H, r, l)$, and f and f^* be the flows at NE for (G, r, l) and (H, r, l).
- $C(f) = r \cdot L(G, r, l)$ and $C(f^*) = r \cdot L(H, r, l)$.
- f^* feasible for (G, r, l), thus $C(f) \leq \frac{4}{3} C(f^*)$.
- Hence, $L(G, r, l) \leq \frac{4}{3} L(H, r, l)$.

□

The trivial algorithm performance for LINEAR LATENCY NETWORK DESIGN

Corollary

The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN.

Proof.

- Let H be the subgraph that minimizes $L(H, r, l)$, and f and f^* be the flows at NE for (G, r, l) and (H, r, l).
- $C(f) = r \cdot L(G, r, l)$ and $C(f^*) = r \cdot L(H, r, l)$.
- f^* feasible for (G, r, l), thus $C(f) \leq \frac{4}{3} C(f^*)$.
- Hence, $L(G, r, l) \leq \frac{4}{3} L(H, r, l)$.

□

The trivial algorithm performance for LINEAR LATENCY NETWORK DESIGN

Corollary

The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN.

Proof.

- Let H be the subgraph that minimizes $L(H, r, l)$, and f and f^* be the flows at NE for (G, r, l) and (H, r, l).
- $C(f) = r \cdot L(G, r, l)$ and $C(f^*) = r \cdot L(H, r, l)$.
- f^* feasible for (G, r, l), thus $C(f) \leq \frac{4}{3} C(f^*)$.
- Hence, $L(G, r, l) \leq \frac{4}{3} L(H, r, l)$.

□

The trivial algorithm performance for LINEAR LATENCY NETWORK DESIGN

Corollary

The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN.

Proof.

- Let H be the subgraph that minimizes $L(H, r, l)$, and f and f^* be the flows at NE for (G, r, l) and (H, r, l).
- $C(f) = r \cdot L(G, r, l)$ and $C(f^*) = r \cdot L(H, r, l)$.
- f^* feasible for (G, r, l), thus $C(f) \leq \frac{4}{3} C(f^*)$.
- Hence, $L(G, r, l) \leq \frac{4}{3} L(H, r, l)$.

□

The trivial algorithm performance for LINEAR LATENCY NETWORK DESIGN

Corollary

The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN.

Proof.

- Let H be the subgraph that minimizes $L(H, r, l)$, and f and f^* be the flows at NE for (G, r, l) and (H, r, l).
- $C(f) = r \cdot L(G, r, l)$ and $C(f^*) = r \cdot L(H, r, l)$.
- f^* feasible for (G, r, l), thus $C(f) \leq \frac{4}{3} C(f^*)$.
- Hence, $L(G, r, l) \leq \frac{4}{3} L(H, r, l)$.

□

Optimality of the trivial algorithm (1 / 3)

Theorem

For every $\epsilon > 0$, there is no $\left(\frac{4}{3} - \epsilon\right)$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN, assuming $\mathrm{P} \neq \mathrm{NP}$.

We will use a reduction from the 2 DIRECTED DISJOINT PATHS (2DDP) problem: given a directed graph $G = (V, E)$ and distinct vertices $s_1, s_2, t_1, t_2 \in V$, are there $s_i - t_i$ paths P_i for $i = 1, 2$, such that P_1 and P_2 are vertex-disjoint?

2DDP is NP-complete.

Optimality of the trivial algorithm (1 / 3)

Theorem

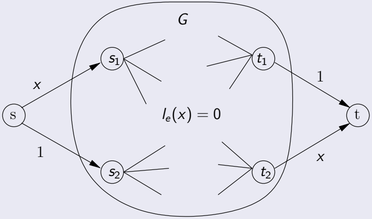
For every $\epsilon > 0$, there is no $\left(\frac{4}{3} - \epsilon\right)$-approximation algorithm for LINEAR LATENCY NETWORK DESIGN, assuming $\mathrm{P} \neq \mathrm{NP}$.

We will use a reduction from the 2 DIRECTED DISJOINT PATHS (2DDP) problem: given a directed graph $G = (V, E)$ and distinct vertices $s_1, s_2, t_1, t_2 \in V$, are there $s_i - t_i$ paths P_i for $i = 1, 2$, such that P_1 and P_2 are vertex-disjoint?

2DDP is NP-complete.

Optimality of the trivial algorithm (2 / 3)

Proof.

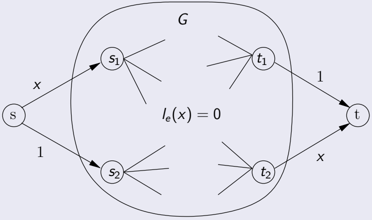


$(G', 1, l)$

- If algorithm returns a subgraph H with $L(H, 1, l) < 2$, then "yes" instance of 2DDP, else "no".
- If "yes" instance, let P_1 and P_2 be vertext disjoint $s_1 - t_1$ and $s_2 - t_2$ paths. Obtain H by deleting all other edges. Observe now that $L(H, 1, l) = \frac{3}{2}$ (1/2 routed in $s_1 \to t_1 \to t$ and 1/2 in $s_2 \to t_2 \to t$). So, $ALG \leq \left(\frac{4}{3} - \epsilon\right) \cdot \frac{3}{2} < 2$.

Optimality of the trivial algorithm (2 / 3)

Proof.



$(G', 1, l)$

- If algorithm returns a subgraph H with $L(H, 1, l) < 2$, then "yes" instance of 2DDP, else "no".
- If "yes" instance, let P_1 and P_2 be vertext disjoint $s_1 - t_1$ and $s_2 - t_2$ paths. Obtain H by deleting all other edges. Observe now that $L(H, 1, l) = \frac{3}{2}$ (1/2 routed in $s_1 \to t_1 \to t$ and 1/2 in $s_2 \to t_2 \to t$). So, $ALG \leq \left(\frac{4}{3} - \epsilon\right) \cdot \frac{3}{2} < 2$.

Optimality of the trivial algorithm (3 / 3)

Proof (continued).

- We will prove that if "no" instance, then $L(H, 1, l) \geq 2$ for all subgraphs of G', and so $ALG \geq 2$.
- Split subgraphs of G' in 3 groups: (i) those with an $s_2 - t_1$ path, (ii) those with an $s_1 - t_2$ path and (iii) those with an $s_i - t_i$ path for exactly one $i \in \{1, 2\}$.
- In all cases, routing flow in such a path gives NE and $L(H) = 2$.
- Thus, $ALG \geq OPT \geq 2$, and so we solve 2DDP.

□

Interpretation of results

- Efficiently detecting Braess's Paradox in networks with linear latency functions is impossible (i.e. NP-hard). This holds even in the most severe cases, where $PoA = \frac{4}{3}$.
- However, by restricting our linear latency functions only to strictly increasing ones, we can get positive results!

Interpretation of results

- Efficiently detecting Braess's Paradox in networks with linear latency functions is impossible (i.e. NP-hard). This holds even in the most severe cases, where $PoA = \frac{4}{3}$.
- However, by restricting our linear latency functions only to strictly increasing ones, we can get positive results!

Towards some positive results

For instances with strictly increasing linear latencies, the optimal flow is **unique** and can be efficiently computed.

Definition

An instance (G, r, l) is called *paradox-free* if for every subgraph H of G, $L(H, r, l) \geq L(G, r, l)$. An instance (G, r, l) is called *paradox-ridden* if there is a subgraph H of G, such that $L(H, r, l) = L^*(G, r, l) = L(G, r, l)/PoA(G, r, l) \leq L(G, r, l)$.

Note: In a paradox-free instance PoA cannot be improved by edge removal.

Lemma

An instance (G, r, l) with $G = (V, E)$ is paradox-ridden iff there is an optimal flow f^* that is a Nash flow on the subgraph $G^*(V, E^*)$, where $E^* = \{e \in E : f^*_e > 0\}$.

Towards some positive results

For instances with strictly increasing linear latencies, the optimal flow is **unique** and can be efficiently computed.

Definition

An instance (G, r, l) is called *paradox-free* if for every subgraph H of G, $L(H, r, l) \geq L(G, r, l)$. An instance (G, r, l) is called *paradox-ridden* if there is a subgraph H of G, such that $L(H, r, l) = L^*(G, r, l) = L(G, r, l)/PoA(G, r, l) \leq L(G, r, l)$.

Note: In a paradox-free instance PoA cannot be improved by edge removal.

Lemma

An instance (G, r, l) with $G = (V, E)$ is paradox-ridden iff there is an optimal flow f^* that is a Nash flow on the subgraph $G^*(V, E^*)$, where $E^* = \{e \in E : f^*_e > 0\}$.

Towards some positive results

For instances with strictly increasing linear latencies, the optimal flow is **unique** and can be efficiently computed.

Definition

An instance (G, r, l) is called *paradox-free* if for every subgraph H of G, $L(H, r, l) \geq L(G, r, l)$. An instance (G, r, l) is called *paradox-ridden* if there is a subgraph H of G, such that $L(H, r, l) = L^*(G, r, l) = L(G, r, l)/PoA(G, r, l) \leq L(G, r, l)$.

Note: In a paradox-free instance PoA cannot be improved by edge removal.

Lemma

An instance (G, r, l) with $G = (V, E)$ is paradox-ridden iff there is an optimal flow f^* that is a Nash flow on the subgraph $G^*(V, E^*)$, where $E^* = \{e \in E : f^*_e > 0\}$.

Towards some positive results

For instances with strictly increasing linear latencies, the optimal flow is **unique** and can be efficiently computed.

Definition

An instance (G, r, l) is called *paradox-free* if for every subgraph H of G, $L(H, r, l) \geq L(G, r, l)$. An instance (G, r, l) is called *paradox-ridden* if there is a subgraph H of G, such that $L(H, r, l) = L^*(G, r, l) = L(G, r, l)/PoA(G, r, l) \leq L(G, r, l)$.

Note: In a paradox-free instance PoA cannot be improved by edge removal.

Lemma

An instance (G, r, l) with $G = (V, E)$ is paradox-ridden iff there is an optimal flow f^* that is a Nash flow on the subgraph $G^*(V, E^*)$, where $E^* = \{e \in E : f_e^* > 0\}$.

Detecting paradox-ridden networks

Theorem (Fotakis, Kaporis, Spirakis)

Given an instance (G, r, l) with strictly increasing linear latencies, one can decide in polynomial time whether the instance is paradox-ridden or not.

Proof.

- We can efficiently compute the *unique* optimal flow f^*.
- We then compute the length $d(v)$ of a shortest $s - v$ path wrt edge lengths $\{l_e(f_e^*)\}_{e \in E^*}$ for all $v \in V$.
- f^* Nash flow $\Leftrightarrow \forall (u, v) \in E^*, d(v) = d(u) + l_{(u,v)}(f^*_{(u,v)})$.

□

Detecting paradox-ridden networks

Theorem (Fotakis, Kaporis, Spirakis)

Given an instance (G, r, l) with strictly increasing linear latencies, one can decide in polynomial time whether the instance is paradox-ridden or not.

Proof.

- We can efficiently compute the *unique* optimal flow f^*.
- We then compute the length $d(v)$ of a shortest $s - v$ path wrt edge lengths $\{l_e(f_e^*)\}_{e \in E^*}$ for all $v \in V$.
- f^* Nash flow $\Leftrightarrow \forall (u, v) \in E^*, d(v) = d(u) + l_{(u,v)}(f^*_{(u,v)})$.

□

Detecting paradox-ridden networks

Theorem (Fotakis, Kaporis, Spirakis)

Given an instance (G, r, l) with strictly increasing linear latencies, one can decide in polynomial time whether the instance is paradox-ridden or not.

Proof.

- We can efficiently compute the *unique* optimal flow f^*.
- We then compute the length $d(v)$ of a shortest $s - v$ path wrt edge lengths $\{l_e(f_e^*)\}_{e \in E^*}$ for all $v \in V$.
- f^* Nash flow $\Leftrightarrow \forall (u, v) \in E^*, d(v) = d(u) + l_{(u,v)}(f^*_{(u,v)})$.

□

Detecting paradox-ridden networks

Theorem (Fotakis, Kaporis, Spirakis)

Given an instance (G, r, l) with strictly increasing linear latencies, one can decide in polynomial time whether the instance is paradox-ridden or not.

Proof.

- We can efficiently compute the *unique* optimal flow f^*.
- We then compute the length $d(v)$ of a shortest $s - v$ path wrt edge lengths $\{l_e(f_e^*)\}_{e \in E^*}$ for all $v \in V$.
- f^* Nash flow $\Leftrightarrow \forall (u, v) \in E^*, d(v) = d(u) + l_{(u,v)}(f^*_{(u,v)})$.

□

Towards a positive result for arbitrary linear latencies

- As already stated, we cannot decide whether an instance with arbitrary linear latencies is paradox-ridden or not.
- However, we can reach sufficient conditions under which we can answer the above question.
- Let (G, r, l) be an instance with $l_e(x) = a_e(x) + b_e$ and $E^c = \{e \in E : a_e = 0\}$. Let $E^i = E \setminus E^c$ and let $\mathcal{O}$ be the set of optimal flows.

Note: All optimal flows assign the same traffic to the edges with strictly increasing latencies, and can differ only on edges with constant latencies.

This motivates the following LP formulation, given a **fixed** optimal flow o.

Towards a positive result for arbitrary linear latencies

- As already stated, we cannot decide whether an instance with arbitrary linear latencies is paradox-ridden or not.
- However, we can reach sufficient conditions under which we can answer the above question.
- Let (G, r, l) be an instance with $l_e(x) = a_e(x) + b_e$ and $E^c = \{e \in E : a_e = 0\}$. Let $E^i = E \setminus E^c$ and let $\mathcal{O}$ be the set of optimal flows.

Note: All optimal flows assign the same traffic to the edges with strictly increasing latencies, and can differ only on edges with constant latencies.

This motivates the following LP formulation, given a **fixed** optimal flow o.

Towards a positive result for arbitrary linear latencies

- As already stated, we cannot decide whether an instance with arbitrary linear latencies is paradox-ridden or not.
- However, we can reach sufficient conditions under which we can answer the above question.
- Let (G, r, l) be an instance with $l_e(x) = a_e(x) + b_e$ and $E^c = \{e \in E : a_e = 0\}$. Let $E^i = E \setminus E^c$ and let $\mathcal{O}$ be the set of optimal flows.

Note: All optimal flows assign the same traffic to the edges with strictly increasing latencies, and can differ only on edges with constant latencies.

This motivates the following LP formulation, given a **fixed** optimal flow o.

Towards a positive result for arbitrary linear latencies

- As already stated, we cannot decide whether an instance with arbitrary linear latencies is paradox-ridden or not.
- However, we can reach sufficient conditions under which we can answer the above question.
- Let (G, r, l) be an instance with $l_e(x) = a_e(x) + b_e$ and $E^c = \{e \in E : a_e = 0\}$. Let $E^i = E \setminus E^c$ and let $\mathcal{O}$ be the set of optimal flows.

Note: All optimal flows assign the same traffic to the edges with strictly increasing latencies, and can differ only on edges with constant latencies.

This motivates the following LP formulation, given a **fixed** optimal flow o.

An LP formulation

(LP):

$$\min \quad \sum_{e \in E^c} f_e b_e, \quad s.t.:$$

$$\sum_{u:(v,u)\in E^i} o_{(v,u)} + \sum_{u:(v,u)\in E^c} f_{(v,u)} = \sum_{u:(u,v)\in E^i} o_{(u,v)} + \sum_{u:(u,v)\in E^c} f_{(u,v)} \qquad \forall v \in V \setminus \{s,t\},$$

$$\sum_{u:(s,u)\in E^i} o_{(s,u)} + \sum_{u:(s,u)\in E^c} f_{(s,u)} = r,$$

$$\sum_{u:(u,t)\in E^i} o_{(u,t)} + \sum_{u:(u,t)\in E^c} f_{(u,t)} = r,$$

$$f_e \geq 0 \qquad \forall e \in E^c.$$

Some notes on the (LP)

- An optimal solution to (LP) corresponds to a feasible flow for (G, r, l) that agrees with o on all edges in E^i and allocates traffic to the edges in E^c so that the total latency is minimized.
- Optimal solutions to (LP) $\xleftrightarrow{1-1}$ Optimal flows in $\mathcal{O}$.
- Given an optimal flow o, the problem of checking if there is a $o \in \mathcal{O}$ that is a Nash flow on G_o reduces to the problem of generating all optimal solutions of (LP) and checking whether some of them can be translated into a Nash flow on the corresponding subnetwork.
- This can be performed in polynomial time if (LP)'s optimal solution is unique.

Some notes on the (LP)

- An optimal solution to (LP) corresponds to a feasible flow for (G, r, l) that agrees with o on all edges in E^i and allocates traffic to the edges in E^c so that the total latency is minimized.
- Optimal solutions to (LP) $\xleftrightarrow{1-1}$ Optimal flows in $\mathcal{O}$.
- Given an optimal flow o, the problem of checking if there is a $o \in \mathcal{O}$ that is a Nash flow on G_o reduces to the problem of generating all optimal solutions of (LP) and checking whether some of them can be translated into a Nash flow on the corresponding subnetwork.
- This can be performed in polynomial time if (LP)'s optimal solution is unique.

Some notes on the (LP)

- An optimal solution to (LP) corresponds to a feasible flow for (G, r, l) that agrees with o on all edges in E^i and allocates traffic to the edges in E^c so that the total latency is minimized.
- Optimal solutions to (LP) $\stackrel{1-1}{\longleftrightarrow}$ Optimal flows in $\mathcal{O}$.
- Given an optimal flow o, the problem of checking if there is a $o \in \mathcal{O}$ that is a Nash flow on G_o reduces to the problem of generating all optimal solutions of (LP) and checking whether some of them can be translated into a Nash flow on the corresponding subnetwork.
- This can be performed in polynomial time if (LP)'s optimal solution is unique.

Some notes on the (LP)

- An optimal solution to (LP) corresponds to a feasible flow for (G, r, l) that agrees with o on all edges in E^i and allocates traffic to the edges in E^c so that the total latency is minimized.
- Optimal solutions to (LP) $\stackrel{1-1}{\longleftrightarrow}$ Optimal flows in $\mathcal{O}$.
- Given an optimal flow o, the problem of checking if there is a $o \in \mathcal{O}$ that is a Nash flow on G_o reduces to the problem of generating all optimal solutions of (LP) and checking whether some of them can be translated into a Nash flow on the corresponding subnetwork.
- This can be performed in polynomial time if (LP)'s optimal solution is unique.

A positive result for arbitrary linear latencies (1 / 2)

Theorem

Given an instance (G, r, l) with arbitrary linear latencies where the corresponding (LP) has a unique optimal solution, one can decide in polynomial time whether the instance is paradox-ridden or not.

Note: In fact, it suffices to generate all optimal basic feasible solutions, as the (LP) allocates traffic to constant latency edges. Observe that if a feasible flow f is a Nash flow, then any solution f' with $\{e : f'_e > 0\} \subseteq \{e : f_e > 0\}$ is a Nash flow, too.

A positive result for arbitrary linear latencies (1 / 2)

Theorem

Given an instance (G, r, l) with arbitrary linear latencies where the corresponding (LP) has a unique optimal solution, one can decide in polynomial time whether the instance is paradox-ridden or not.

Note: In fact, it suffices to generate all optimal basic feasible solutions, as the (LP) allocates traffic to constant latency edges. Observe that if a feasible flow f is a Nash flow, then any solution f' with $\{e : f'_e > 0\} \subseteq \{e : f_e > 0\}$ is a Nash flow, too.

A positive result for arbitrary linear latencies (2 / 2)

Theorem

Given an instance (G, r, l) with arbitrary linear latencies where the corresponding (LP) has a polynomial number of basic feasible solutions, one can decide in polynomial time whether the instance is paradox-ridden or not.

Note: The above class includes instances with a constant number of constant latency edges.

A positive result for arbitrary linear latencies (2 / 2)

Theorem

Given an instance (G, r, l) with arbitrary linear latencies where the corresponding (LP) has a polynomial number of basic feasible solutions, one can decide in polynomial time whether the instance is paradox-ridden or not.

Note: The above class includes instances with a constant number of constant latency edges.

Finding near-optimal subnetworks

- In general, finding optimal subnetworks in paradox-ridden instances is NP-hard.
- However, we can reach a subexponential-time approximation scheme on networks with polynomially many paths, each of polylogarithmic length.
- For this purpose, we need to turn our attention to "sparse" flows and ε-Nash flows.

Definition (ε-Nash flow)

For some $\varepsilon > 0$, a flow f is an ε-Nash flow if for every path P and P' with $f_P > 0$, $l_P(f) \leq l_{P'}(f) + \varepsilon$.

Finding near-optimal subnetworks

- In general, finding optimal subnetworks in paradox-ridden instances is NP-hard.
- However, we can reach a subexponential-time approximation scheme on networks with polynomially many paths, each of polylogarithmic length.
- For this purpose, we need to turn our attention to "sparse" flows and ε-Nash flows.

Definition (ε-Nash flow)

For some $\varepsilon > 0$, a flow f is an ε-Nash flow if for every path P and P' with $f_P > 0$, $l_P(f) \leq l_{P'}(f) + \varepsilon$.

Finding near-optimal subnetworks

- In general, finding optimal subnetworks in paradox-ridden instances is NP-hard.
- However, we can reach a subexponential-time approximation scheme on networks with polynomially many paths, each of polylogarithmic length.
- For this purpose, we need to turn our attention to "sparse" flows and ε-Nash flows.

Definition (ε-Nash flow)

For some $\varepsilon > 0$, a flow f is an ε-Nash flow if for every path P and P' with $f_P > 0$, $l_P(f) \leq l_{P'}(f) + \varepsilon$.

Finding near-optimal subnetworks

- In general, finding optimal subnetworks in paradox-ridden instances is NP-hard.
- However, we can reach a subexponential-time approximation scheme on networks with polynomially many paths, each of polylogarithmic length.
- For this purpose, we need to turn our attention to "sparse" flows and ε-Nash flows.

Definition (ε-Nash flow)

For some $\varepsilon > 0$, a flow f is an ε-Nash flow if for every path P and P' with $f_P > 0$, $l_P(f) \leq l_{P'}(f) + \varepsilon$.

Making a flow "sparse" (1 / 3)

Lemma (Fotakis, Kaporis, Spirakis)

Let $(G, 1, l)$ be an instance on a graph $G = (V, E)$, and let f be any feasible flow. For any $\varepsilon > 0$, there exists a feasible flow $\tilde{f}$ that assigns positive traffic to at most $\lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$ paths, such that $|\tilde{f}_e - f_e| \leq \varepsilon$, $\forall e \in E$.

Proof.

- Let $\mu = |\mathcal{P}|$, and we index the $s - t$ paths by integers in $[\mu]$.
- Flow f can be seen as a probability distribution on $\mathcal{P}$.
- We prove that if we select $k > \log(2m)/(2\varepsilon^2)$ paths uniformly at random with replacement according to f, and assign to each path j a flow equal to the number of times j is selected divided by k, we obtain a flow that is an ε-approximation to f with positive probability. (Probabilistic Method)

Making a flow "sparse" (1 / 3)

Lemma (Fotakis, Kaporis, Spirakis)

Let $(G, 1, l)$ be an instance on a graph $G = (V, E)$, and let f be any feasible flow. For any $\varepsilon > 0$, there exists a feasible flow $\tilde{f}$ that assigns positive traffic to at most $\lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$ paths, such that $|\tilde{f}_e - f_e| \leq \varepsilon$, $\forall e \in E$.

Proof.

- Let $\mu = |\mathcal{P}|$, and we index the $s - t$ paths by integers in $[\mu]$.
- Flow f can be seen as a probability distribution on $\mathcal{P}$.
- We prove that if we select $k > \log(2m)/(2\varepsilon^2)$ paths uniformly at random with replacement according to f, and assign to each path j a flow equal to the number of times j is selected divided by k, we obtain a flow that is an ε-approximation to f with positive probability. (Probabilistic Method)

Making a flow "sparse" (2 / 3)

Proof (continued).

- Fix ε and let $k = \lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$.
- Define random variables $P_1, ...P_k \in [\mu]$, i.i.d., such that $\mathrm{P}[P_i = j] = f_j$.
- For each path $j \in [\mu]$, $F_j = |\{i \in [k] : P_i = j\}| / k$. Note that $\mathbf{E}[F_j] = f_j$.
- For each edge e and random variable P_i, define the independent indicator variables $F_{e,i} = 1$ if e in path P_i, otherwise 0.
- Let $F_e = \frac{1}{k} \sum_{i=1}^{k} F_{e,i}$. Observe that $F_e = \sum_{j:e \in j} F_j$ and $\mathbf{E}[F_e] = f_e$.

Making a flow "sparse" (2 / 3)

Proof (continued).

- Fix ε and let $k = \lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$.
- Define random variables $P_1, \ldots P_k \in [\mu]$, i.i.d., such that $\mathrm{P}[P_i = j] = f_j$.
- For each path $j \in [\mu]$, $F_j = |\{i \in [k] : P_i = j\}| / k$. Note that $\mathbf{E}[F_j] = f_j$.
- For each edge e and random variable P_i, define the independent indicator variables $F_{e,i} = 1$ if e in path P_i, otherwise 0.
- Let $F_e = \frac{1}{k} \sum_{i=1}^{k} F_{e,i}$. Observe that $F_e = \sum_{j:e \in j} F_j$ and $\mathbf{E}[F_e] = f_e$.

Making a flow "sparse" (2 / 3)

Proof (continued).

- Fix ε and let $k = \lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$.
- Define random variables $P_1, \ldots P_k \in [\mu]$, i.i.d., such that $\mathrm{P}[P_i = j] = f_j$.
- For each path $j \in [\mu]$, $F_j = |\{i \in [k] : P_i = j\}| \,/\, k$. Note that $\mathbf{E}[F_j] = f_j$.
- For each edge e and random variable P_i, define the independent indicator variables $F_{e,i} = 1$ if e in path P_i, otherwise 0.
- Let $F_e = \frac{1}{k} \sum_{i=1}^{k} F_{e,i}$. Observe that $F_e = \sum_{j: e \in j} F_j$ and $\mathbf{E}[F_e] = f_e$.

Making a flow "sparse" (2 / 3)

Proof (continued).

- Fix ε and let $k = \lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$.
- Define random variables $P_1, \ldots P_k \in [\mu]$, i.i.d., such that $\mathrm{P}[P_i = j] = f_j$.
- For each path $j \in [\mu]$, $F_j = |\{i \in [k] : P_i = j\}| \,/\, k$. Note that $\mathbf{E}[F_j] = f_j$.
- For each edge e and random variable P_i, define the independent indicator variables $F_{e,i} = 1$ if e in path P_i, otherwise 0.
- Let $F_e = \frac{1}{k} \sum_{i=1}^{k} F_{e,i}$. Observe that $F_e = \sum_{j:e \in j} F_j$ and $\mathbf{E}[F_e] = f_e$.

Making a flow "sparse" (2 / 3)

Proof (continued).

- Fix ε and let $k = \lfloor \log(2m)/(2\varepsilon^2) \rfloor + 1$.
- Define random variables $P_1, ...P_k \in [\mu]$, i.i.d., such that $\mathrm{P}[P_i = j] = f_j$.
- For each path $j \in [\mu]$, $F_j = |\{i \in [k] : P_i = j\}| \,/\, k$. Note that $\mathbf{E}[F_j] = f_j$.
- For each edge e and random variable P_i, define the independent indicator variables $F_{e,i} = 1$ if e in path P_i, otherwise 0.
- Let $F_e = \frac{1}{k} \sum_{i=1}^{k} F_{e,i}$. Observe that $F_e = \sum_{j:e \in j} F_j$ and $\mathbf{E}[F_e] = f_e$.

Making a flow "sparse" (3 / 3)

Proof (continued).

- Note that $\sum_{j=1}^{\mu} F_j = 1$. Thus, $F_1, \ldots, F_\mu$ define a feasible flow that assignes positive traffic to at most k paths and "agrees" with f on expectation.
- By the Chernoff-Hoeffding bound we get that for every edge e

$$\mathrm{P}[|F_e - f_e| > \varepsilon] \leq 2e^{-2\varepsilon^2 k} < 1/m$$

- Thus, by union bound, $\mathrm{P}[\exists e : |F_e - f_e| > \varepsilon] < m(1/m) = 1$.
- So, there is positive probability that the flow $(F_1, \ldots, F_\mu)$ satisfies $|F_e - f_e| \leq \varepsilon$, $\forall e \in E$. Thus, there exists a flow $\tilde{f}$ with the properties of $(F_1, \ldots, F_\mu)$.

□

Making a flow "sparse" (3 / 3)

Proof (continued).

- Note that $\sum_{j=1}^{\mu} F_j = 1$. Thus, $F_1, ..., F_\mu$ define a feasible flow that assignes positive traffic to at most k paths and "agrees" with f on expectation.
- By the Chernoff-Hoeffding bound we get that for every edge e

$$\mathrm{P}[|F_e - f_e| > \varepsilon] \leq 2e^{-2\varepsilon^2 k} < 1/m$$

- Thus, by union bound, $\mathrm{P}[\exists e : |F_e - f_e| > \varepsilon] < m(1/m) = 1$.
- So, there is positive probability that the flow $(F_1, ..., F_\mu)$ satisfies $|F_e - f_e| \leq \varepsilon, \forall e \in E$. Thus, there exists a flow $\tilde{f}$ with the properties of $(F_1, ..., F_\mu)$.

Making a flow "sparse" (3 / 3)

Proof (continued).

- Note that $\sum_{j=1}^{\mu} F_j = 1$. Thus, $F_1, \ldots, F_\mu$ define a feasible flow that assignes positive traffic to at most k paths and "agrees" with f on expectation.
- By the Chernoff-Hoeffding bound we get that for every edge e

$$\mathrm{P}[|F_e - f_e| > \varepsilon] \leq 2e^{-2\varepsilon^2 k} < 1/m$$

- Thus, by union bound, $\mathrm{P}[\exists e : |F_e - f_e| > \varepsilon] < m(1/m) = 1$.
- So, there is positive probability that the flow $(F_1, \ldots, F_\mu)$ satisfies $|F_e - f_e| \leq \varepsilon$, $\forall e \in E$. Thus, there exists a flow $\tilde{f}$ with the properties of $(F_1, \ldots, F_\mu)$.

□

Making a flow "sparse" (3 / 3)

Proof (continued).

- Note that $\sum_{j=1}^{\mu} F_j = 1$. Thus, $F_1, \ldots, F_\mu$ define a feasible flow that assignes positive traffic to at most k paths and "agrees" with f on expectation.
- By the Chernoff-Hoeffding bound we get that for every edge e

$$\mathrm{P}[|F_e - f_e| > \varepsilon] \leq 2e^{-2\varepsilon^2 k} < 1/m$$

- Thus, by union bound, $\mathrm{P}[\exists e : |F_e - f_e| > \varepsilon] < m(1/m) = 1$.
- So, there is positive probability that the flow $(F_1, \ldots, F_\mu)$ satisfies $|F_e - f_e| \leq \varepsilon, \forall e \in E$. Thus, there exists a flow $\tilde{f}$ with the properties of $(F_1, \ldots, F_\mu)$.

□

Finding a near-optimal subnetwork

Theorem

Let $(G(V,E), 1, \{a_e x + b_e\}_{e \in E})$ *be an instance,* $\alpha = \max_{e \in E}\{a_e\}$*, and let* H^B *be the best subnetwork of* G*. For some constants* $d_1, d_2 > 0$*, let* $|\mathcal{P}| \leq m^{d_1}$ *and* $|P| \leq \log^{d_2} m$*, for all* $P \in \mathcal{P}$*. Then, for any* $\varepsilon > 0$*, we can compute in time* $m^{O(d_1 \alpha^2 \log^{2d_2+1}(2m)/\varepsilon^2)}$ *a flow* $\tilde{f}$ *that is an* ε*-Nash flow on* $G_{\tilde{f}}$ *and satisfies* $l_P(\tilde{f}) \leq L(H^B, 1, \{a_e x + b_e\}_{e \in E(H^B)}) + \varepsilon/2$*, for all paths* $P \in G_{\tilde{f}}$*.*

Moving on to general latency functions

- We will now consider general (continuous, non-decreasing) latency functions (we call this problem the GENERAL LATENCY NETWORK DESIGN).
- We will see that the trivial algorithm is still the best thing we can do. However, the approximation factor gets worse.
- In order to prove the above, we will need new techniques, as in networks with general latency functions, a Nash flow can be arbitrarily more costly than other feasible flows.

A useful tool in our approach

Definition

Let f be a Nash flow for the instance (G, r, l). Let $d(v)$ denote the length of a shortest $s - v$ path with respect to the edge lengths $\{l_e(f_e)\}_{e \in E}$. An ordering of the vertices of G is f-monotone if it satisfies the following two properties:

(P1) All f-flow travels forward in the ordering.

(P2) The d-values of vertices are non-decreasing in the ordering.

Note: It can be proved that an f-monotone ordering exists relative to a directed acyclic Nash flow.

A useful tool in our approach

Definition

Let f be a Nash flow for the instance (G, r, l). Let $d(v)$ denote the length of a shortest $s - v$ path with respect to the edge lengths $\{l_e(f_e)\}_{e \in E}$. An ordering of the vertices of G is f-monotone if it satisfies the following two properties:

(P1) All f-flow travels forward in the ordering.

(P2) The d-values of vertices are non-decreasing in the ordering.

Note: It can be proved that an f-monotone ordering exists relative to a directed acyclic Nash flow.

The trivial algorithm performance for GENERAL LATENCY NETWORK DESIGN

Theorem

The trivial algorithm is a $\lfloor n/2 \rfloor$-approximation algorithm for GENERAL LATENCY NETWORK DESIGN.

Proof.

On board

The trivial algorithm performance for GENERAL LATENCY NETWORK DESIGN

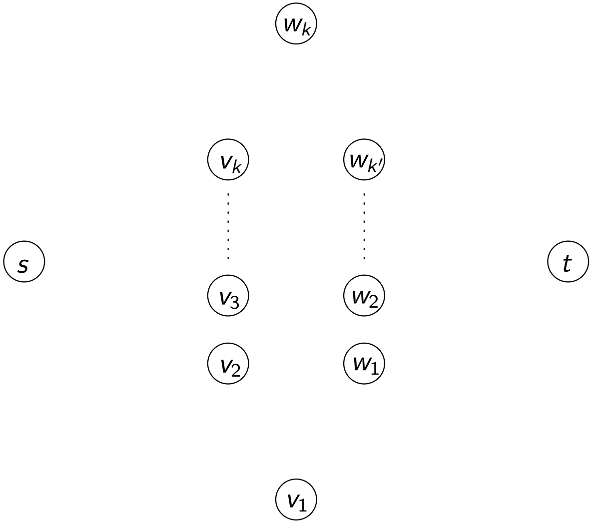
Theorem

The trivial algorithm is a $\lfloor n/2 \rfloor$-approximation algorithm for GENERAL LATENCY NETWORK DESIGN.

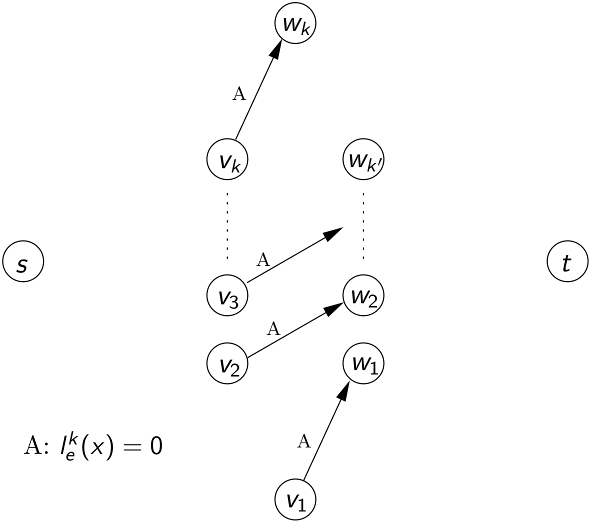
Proof.

On board

Tightness of the $\lfloor n/2 \rfloor$ bound: the B^k Braess Graph

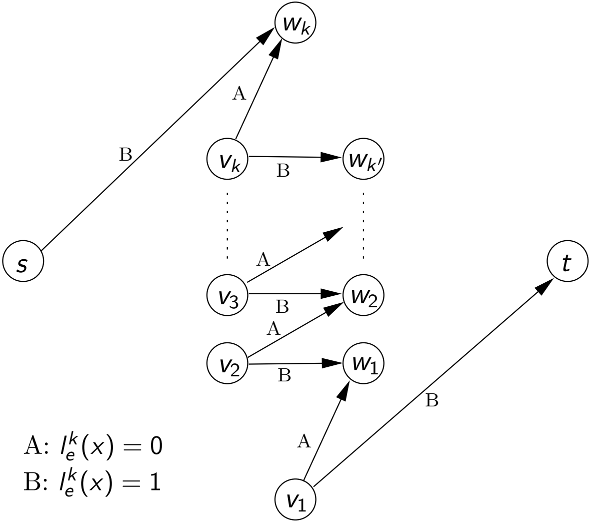


Tightness of the $\lfloor n/2 \rfloor$ bound: the B^k Braess Graph



A: $l_e^k(x) = 0$

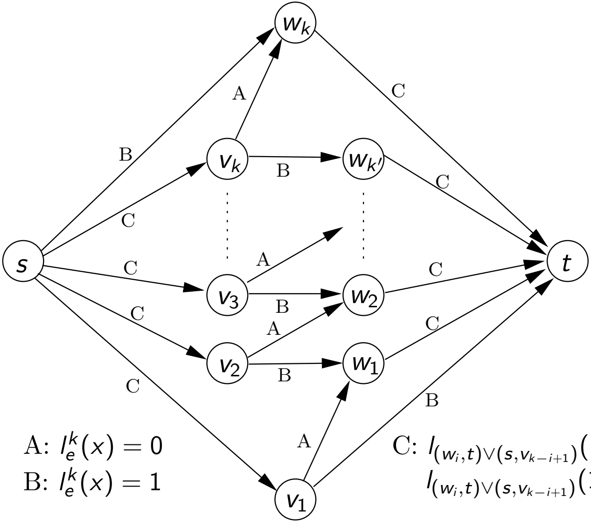
Tightness of the $\lfloor n/2 \rfloor$ bound: the B^k Braess Graph



A: $l_e^k(x) = 0$

B: $l_e^k(x) = 1$

Tightness of the $\lfloor n/2 \rfloor$ bound: the B^k Braess Graph



A: $l_e^k(x) = 0$
B: $l_e^k(x) = 1$

C: $l_{(w_i,t)\vee(s,v_{k-i+1})}(k/(k+1)) = 0$
$l_{(w_i,t)\vee(s,v_{k-i+1})}(1) = i$

Tightness of the $\lfloor n/2 \rfloor$ bound (1 / 2)

Theorem

For every integer $n \geq 2$, there is an instance (G, r, l) in which G has n vertices and a subgraph H with

$$L(G, r, l) = \left\lfloor \frac{n}{2} \right\rfloor \cdot L(H, r, l).$$

Proof.

- Assume that $n \geq 4$ is even (otherwise, add an isolated vertex).
- So, $n = 2k + 2$ and we consider the instance (B^k, k, l^k).
- NE for (B^k, k, l^k): 1 unit on each path $s \to v_i \to w_i \to t$, and $L(B^k, k, l^k) = k + 1$.

Tightness of the $\lfloor n/2 \rfloor$ bound (2 / 2)

Proof (continued).

- We now remove all A-type edges and obtain H.
- Routing $k/(k+1)$ units on paths $s \rightarrow v_1 \rightarrow t$, $s \rightarrow w_k \rightarrow t$ and $\{s \rightarrow v_i \rightarrow w_{i-1} \rightarrow t\}_{(i=2,\ldots,k)}$, we get a NE with $L(H, k, l^k) = 1$.
- Thus, $L(G)/L(H) = k + 1 = n/2$.

□

Hardness of approximation for GENERAL LATENCY NETWORK DESIGN

Theorem (Roughgarden)

For every $\epsilon > 0$, there is no $(\lfloor n/2 \rfloor - \epsilon)$-approximation algorithm for GENERAL LATENCY NETWORK DESIGN, assuming $\mathrm{P} \neq \mathrm{NP}$.

Proof is based on a reduction from the NP-complete problem PARTITION.

Price of anarchy in networks with general latency functions

Theorem (Lin, Roughgarden, Tardos)

For every $n \geq 2$ and every single-commodity instance (G, r, l) with n vertices, $PoA(G, r, l) \leq n - 1$.

Lemma

For all $k \geq 1$, the only way to decrease the latency in a Nash flow by a factor strictly larger than k is to remove at least k edges from the network.

Theorem

The worst-case price of anarchy in multicommodity instances with at most n vertices is $2^{\Omega(n)}$ as $n \to \infty$. Moreover, there are instances in which PoA can be reduced to 1 by edge removal.

Outline

1. Introduction
2. Approximation Algorithms - Inapproximability results
3. Frequency of Braess's Paradox

How often does Braess's paradox occur?

Question: Is Braess's paradox often in practical networks or is it just a theoretical curiosity?

Valiant and Roughgarden answer that it occurs in many networks by utilizing random graph models.

Definition (Braess ratio)

The Braess ratio of a network is the largest factor by which the removal of one or more edges can improve the latency of traffic in an equilibrium flow.

How often does Braess's paradox occur?

Question: Is Braess's paradox often in practical networks or is it just a theoretical curiosity?

Valiant and Roughgarden answer that it occurs in many networks by utilizing random graph models.

Definition (Braess ratio)

The Braess ratio of a network is the largest factor by which the removal of one or more edges can improve the latency of traffic in an equilibrium flow.

How often does Braess's paradox occur?

Question: Is Braess's paradox often in practical networks or is it just a theoretical curiosity?

Valiant and Roughgarden answer that it occurs in many networks by utilizing random graph models.

Definition (Braess ratio)

The Braess ratio of a network is the largest factor by which the removal of one or more edges can improve the latency of traffic in an equilibrium flow.

The model

- Probability distribution oven graphs and edge latency functions.
- Graph G distributed according to the standard Erdös-Renyi $\mathcal{G}(n, p)$ model. For a fixed $n \geq 2$, each edge is present independently with probability p. We assume that $p = \Omega(n^{-1/2+\epsilon})$ for some $\epsilon > 0$.
- Source s and destination t are chosen randomly or arbitrarily. (we assume that there is no edge (s, t)).
- Linear latency functions $l(x) = ax + b$, $a, b \geq 0$:
  1. *Independent coefficients* model: two fixed distributions $\mathcal{A}$ and $\mathcal{B}$, and each edge is independently given a latency function $l(x) = ax + b$, where a and b are drawn independently from $\mathcal{A}$ and $\mathcal{B}$, respectively.
  2. 1/x model: each edge present in the graph (independently) has the latency function $l(x) = x$ with probability q and $l(x) = 1$ with probability $1 - q$.

The model

- Probability distribution oven graphs and edge latency functions.
- Graph G distributed according to the standard Erdös-Renyi $\mathcal{G}(n, p)$ model. For a fixed $n \geq 2$, each edge is present independently with probability p. We assume that $p = \Omega(n^{-1/2+\epsilon})$ for some $\epsilon > 0$.
- Source s and destination t are chosen randomly or arbitrarily. (we assume that there is no edge (s, t)).
- Linear latency functions $l(x) = ax + b$, $a, b \geq 0$:
  1. *Independent coefficients* model: two fixed distributions $\mathcal{A}$ and $\mathcal{B}$, and each edge is independently given a latency function $l(x) = ax + b$, where a and b are drawn independently from $\mathcal{A}$ and $\mathcal{B}$, respectively.
  2. $1/x$ model: each edge present in the graph (independently) has the latency function $l(x) = x$ with probability q and $l(x) = 1$ with probability $1 - q$.

The model

- Probability distribution oven graphs and edge latency functions.
- Graph G distributed according to the standard Erdös-Renyi $\mathcal{G}(n, p)$ model. For a fixed $n \geq 2$, each edge is present independently with probability p. We assume that $p = \Omega(n^{-1/2+\epsilon})$ for some $\epsilon > 0$.
- Source s and destination t are chosen randomly or arbitrarily. (we assume that there is no edge (s, t)).
- Linear latency functions $l(x) = ax + b$, $a, b \geq 0$:
  1. *Independent coefficients* model: two fixed distributions $\mathcal{A}$ and $\mathcal{B}$, and each edge is independently given a latency function $l(x) = ax + b$, where a and b are drawn independently from $\mathcal{A}$ and $\mathcal{B}$, respectively.
  2. 1/x model: each edge present in the graph (independently) has the latency function $l(x) = x$ with probability q and $l(x) = 1$ with probability $1 - q$.

The model

- Probability distribution oven graphs and edge latency functions.
- Graph G distributed according to the standard Erdös-Renyi $\mathcal{G}(n, p)$ model. For a fixed $n \geq 2$, each edge is present independently with probability p. We assume that $p = \Omega(n^{-1/2+\epsilon})$ for some $\epsilon > 0$.
- Source s and destination t are chosen randomly or arbitrarily. (we assume that there is no edge (s, t)).
- Linear latency functions $l(x) = ax + b$, $a, b \geq 0$:
  1. *Independent coefficients* model: two fixed distributions $\mathcal{A}$ and $\mathcal{B}$, and each edge is independently given a latency function $l(x) = ax + b$, where a and b are drawn independently from $\mathcal{A}$ and $\mathcal{B}$, respectively.
  2. 1/x model: each edge present in the graph (independently) has the latency function $l(x) = x$ with probability q and $l(x) = 1$ with probability $1 - q$.

Main results

Theorem (Independent coefficients model)

Let $\mathcal{A}$ and $\mathcal{B}$ be reasonable distributions. There is a constant $p = p(\mathcal{A}, \mathcal{B}) > 1$ such that, with high probability, a random network (G, l) admits a choice of traffic rate r such that the Braess ration of the instance (G, r, l) is at least p.

Theorem (The 1/x model)

There is a traffic rate $R = R(n, p, q)$ such that, with high probability as $n \to \infty$, the Braess ratio of a random n-node network from $\mathcal{G}(n, p, q)$ with traffic rate R is at least

$$\frac{4 - 3pq}{3 - 2pq}.$$

Main results

Theorem (Independent coefficients model)

Let $\mathcal{A}$ and $\mathcal{B}$ be reasonable distributions. There is a constant $p = p(\mathcal{A}, \mathcal{B}) > 1$ such that, with high probability, a random network (G, l) admits a choice of traffic rate r such that the Braess ration of the instance (G, r, l) is at least p.

Theorem (The 1/x model)

There is a traffic rate $R = R(n, p, q)$ such that, with high probability as $n \to \infty$, the Braess ratio of a random n-node network from $\mathcal{G}(n, p, q)$ with traffic rate R is at least

$$\frac{4 - 3pq}{3 - 2pq}.$$

THANK YOU!