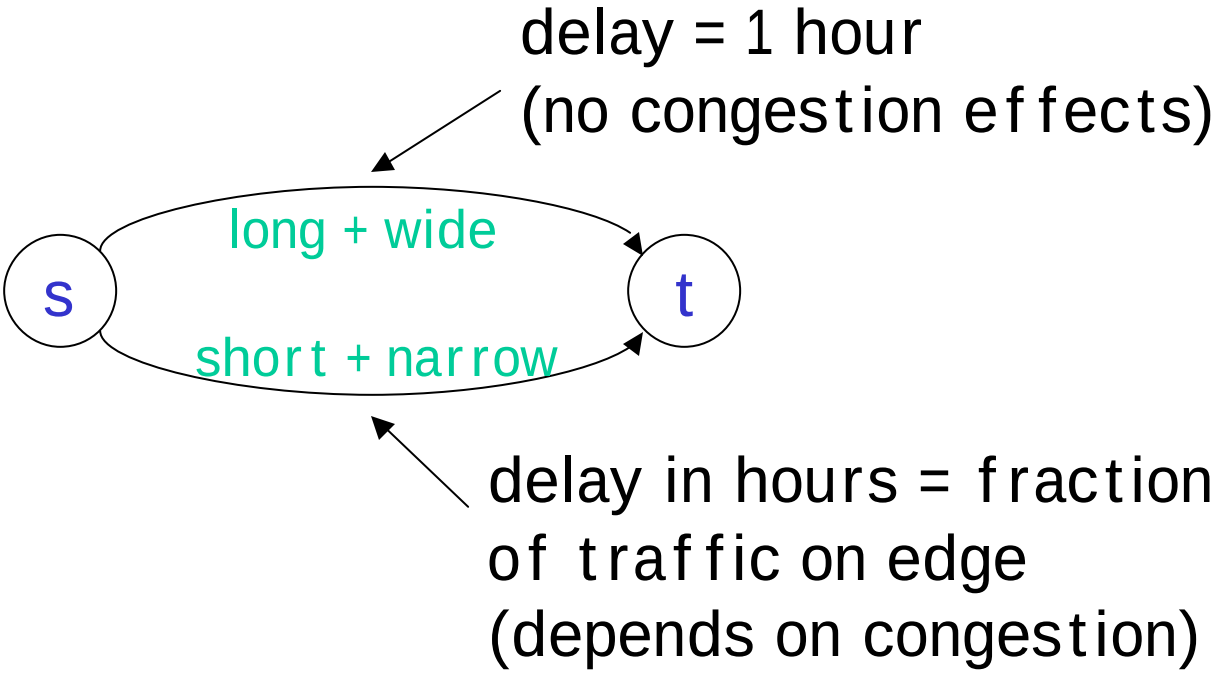
Selfish Routing

Tim Roughgarden
Cornell University

Includes joint work with Éva Tardos

Which route would you choose?

Example: one unit of traffic (e.g., cars) wants to go from s to t

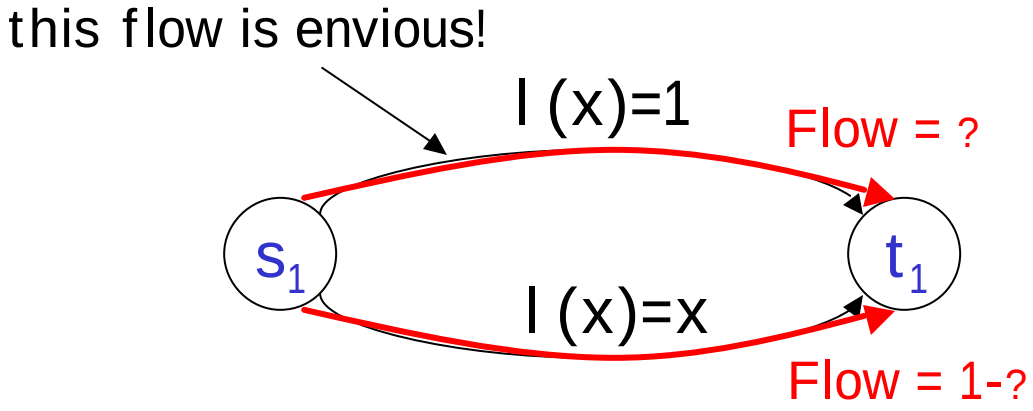


Question: what will selfish network users do?

Outcome of Selfishness

Defining selfishness: all traffic wants to get to t as fast as possible [given what others do]

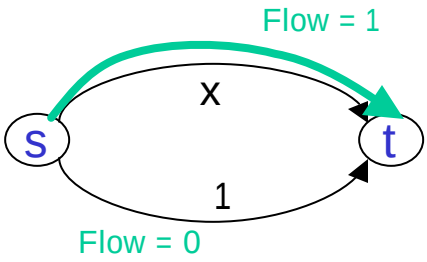
Then: all traffic will use the short narrow road. For if not:



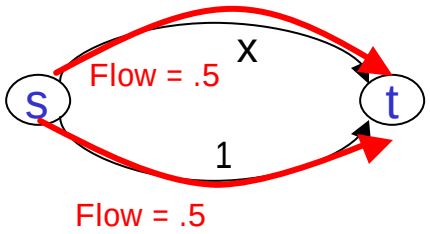
- ? > 0 ⇒ traffic on top is envious
- ? = 0 ⇒ envy-free outcome

Can we improve over the selfish outcome?

Note: all traffic incurs 1 hr of delay



Consider instead:

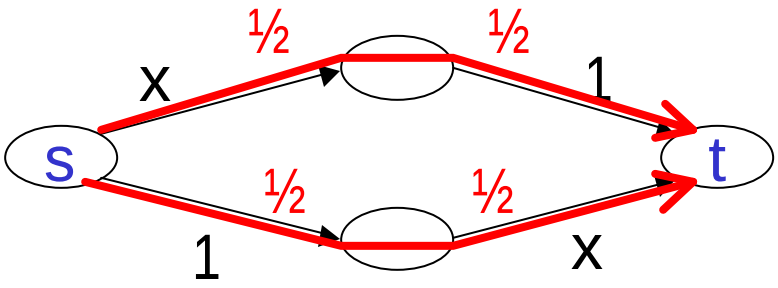


- half of traffic takes 1 hour
  - same as before
- half of traffic takes 30 minutes
  - much improved!

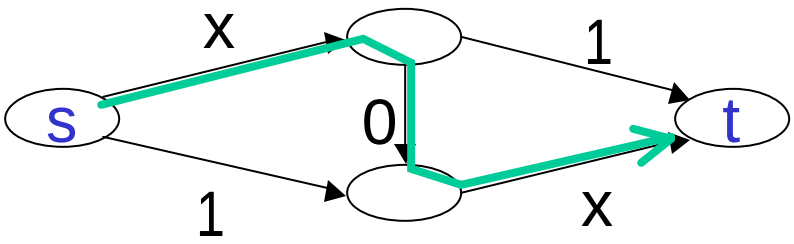
Þ selfish routing need not give the best-possible outcome

Braess's Paradox

Better network, worse outcome:



Delay = 1.5 hours



Delay = 2 hours

All traffic experiences additional delay! [Braess 68]

The Agenda

We know: selfish routing leads to undesirable outcomes.

Our goal: prove worst-case guarantees on

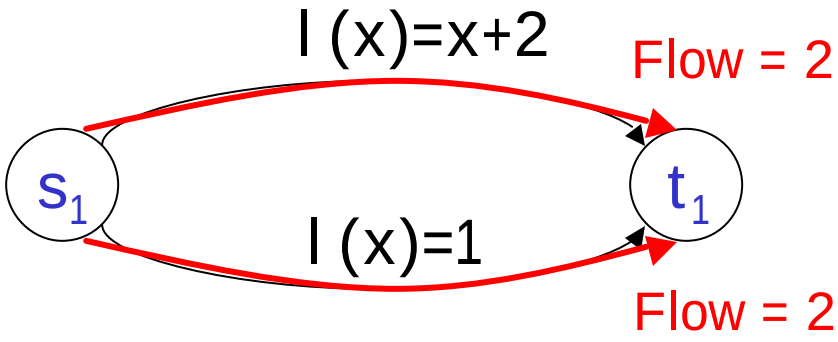
- selfish routing vs. best coordinated outcome
- severity of Braess's Paradox (damage due to extra edges)

Traffic in Congested Networks

The Model:

- A directed graph $G = (V,E)$
- k source-destination pairs $(s_1, t_1), \ldots, (s_k, t_k)$
- A rate r_i of traffic from s_i to t_i
- For each edge e, a latency fn $l_e(\bullet)$ [continuous, nondecreasing]

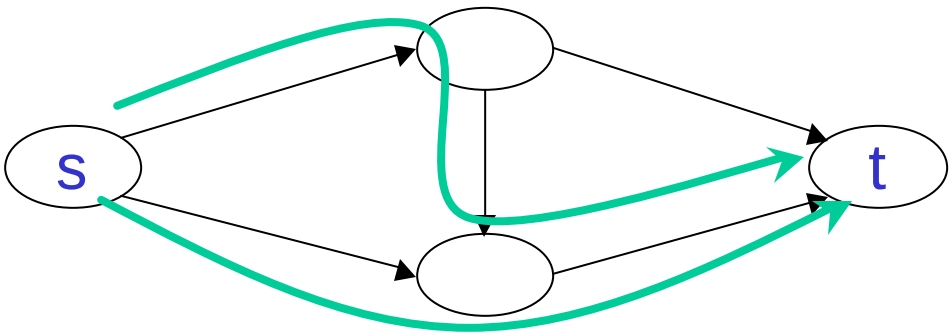
Example: $(k=1, r=4)$



Routings of Traffic

Traffic and Flows:

- f_P = amount of traffic routed on s_i-t_i path P
- flow vector f $\Leftrightarrow$ routing of traffic



Selfish routing: what flows arise as stable equilibria among selfish network users?

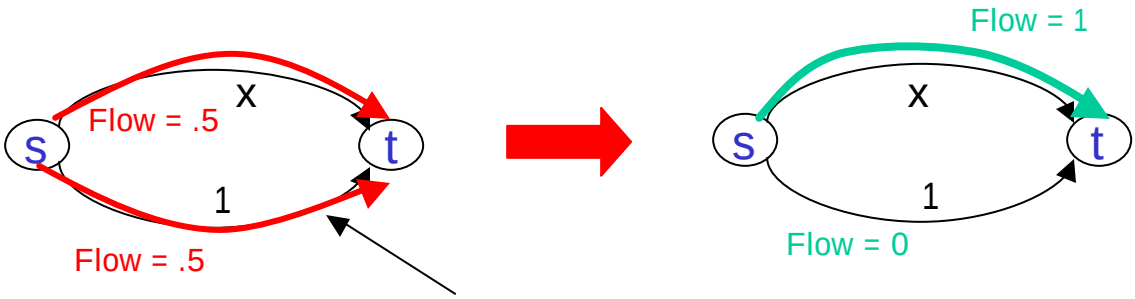
Nash Flows

Some assumptions:

- agents small relative to network
- want to minimize personal latency

Def: A flow is at Nash equilibrium (or is a Nash flow) if all flow is routed on min-latency paths [given current edge congestion]

Example:



this flow is envious!

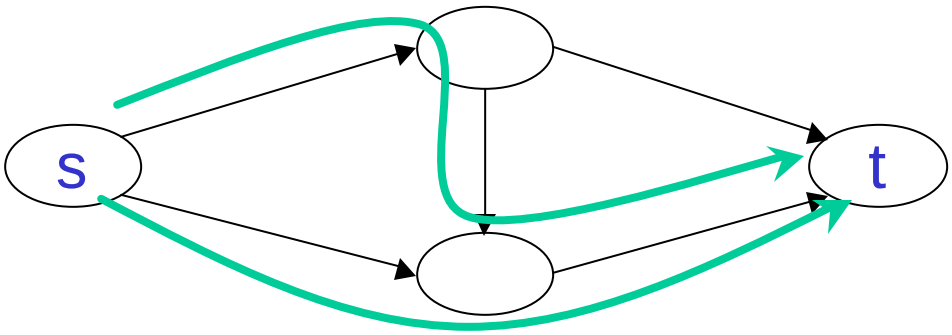
Some History

- traffic model, def of Nash flows due to [Wardrop 52]
  - historically called a user equilibrium
- Nash flows always exist, are (essentially) unique
  - due to [Beckmann et al. 56]

The Cost of a Flow

Our objective function:

- $l_P(f)$ = sum of latencies of edges of P (w.r.t. the flow f)
- $C(f)$ = cost or total latency of flow f: $\sum_P f_P \cdot l_P(f)$

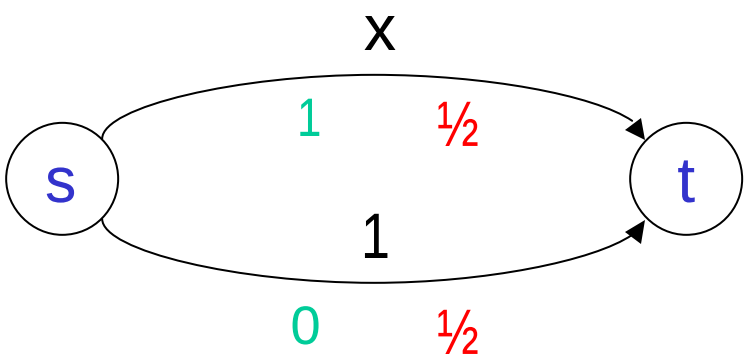


Central question #1: how good (or bad) are Nash flows?

The Inefficiency of Nash Flows

Fact: Nash flows do not optimize total latency [Pigou 1920]

- ⇒ lack of coordination leads to inefficiency

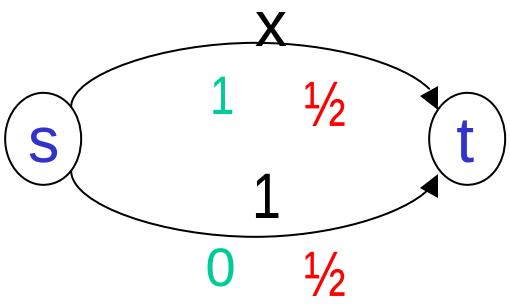


Cost of Nash flow = 1•1 + 0•1 = 1

Cost of optimal (min-cost) flow
= ½•½ +½•1 = ¾

How Bad is Selfish Routing?

Pigou's example is simple…



How inefficient are Nash flows:

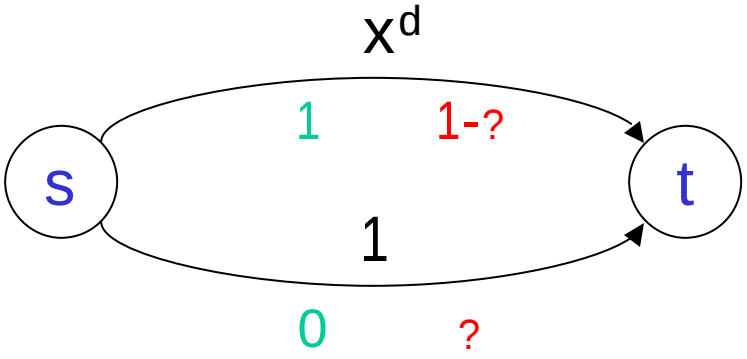
- with more realistic latency fns?
- in more realistic networks?

Goal: prove that Nash flows are near-optimal

- want a laissez-faire approach to managing networks
  - also [Koutsoupias/Papadimitriou 99]

The Bad News

Bad Example: (r = 1, d large)



Nash flow has cost 1, min cost ≈ 0

Þ Nash flow can cost arbitrarily more than the optimal (min-cost) flow

- even if latency functions are polynomials

A Bicriteria Bound

Approach #1: settle for weaker type of guarantee

Theorem: [Roughgarden/Tardos 00] network w/cts, nondecreasing latency functions Þ

cost of Nash at rate r = cost of opt at rate 2r

Corollary: M/M/1 delay fns ($l(x)=1/(u-x)$, u = capacity) Þ

Nash cost w/ capacities 2u = opt cost w/ capacities u

Key Difficulty

Sps f a Nash flow, f^* an opt flow at twice the rate.

Note: we can write

$$C(f) = \sum_e f_e \cdot l_e(f_e)$$

- sum over edges instead of paths
- f_e = amount of flow on edge e

Similarly: $C(f^*) = \sum_e f_e^* \cdot l_e(f_e^*)$

Problem: what is the relation between $l_e(f_e)$ and $l_e(f_e^*)$?

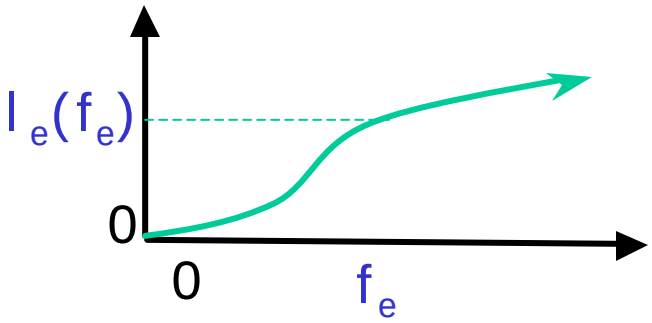
Key Trick

Idea: lower bound cost of f^* using a different set of latency fns c with the properties:

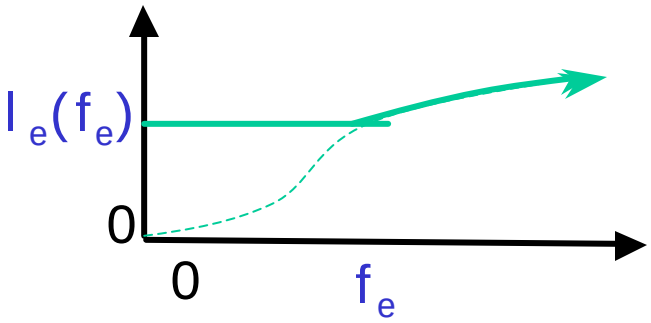
- easy to lower bound cost of f^* w.r.t. latency fns c
- cost of f^* w.r.t. latency fns c $\approx$ cost of f^* w.r.t. latency fns l

The construction:

graph of l



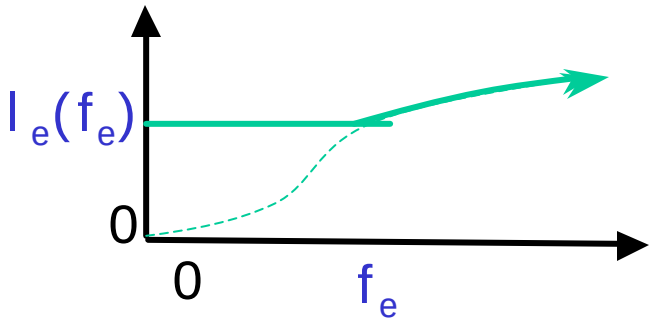
graph of c



Lower Bounding OPT

Assume: only one commodity (multicommodity no harder).

Key observation: latency of path P w.r.t. latency fns c with no congestion is $l_P(f)$ [latency in Nash]



Corollary: Sps in Nash, everyone has latency L, so $C(f) = rL$. Then cost of f^* w.r.t. latency fns c is $\geq 2rL$.

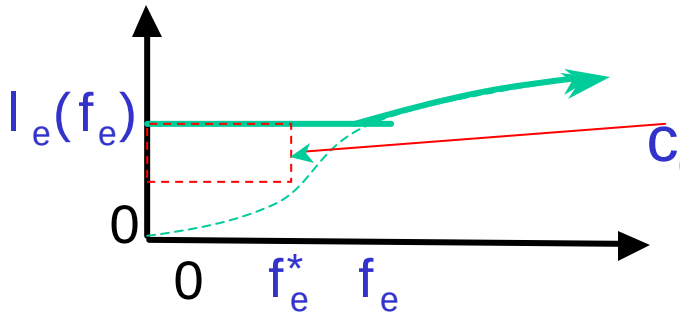
Upper Bounding the Overestimate

Thus: cost of f^* w.r.t. c is $\geq 2C(f)$.

Claim: (will finish proof of Thm)

[cost of f^* w.r.t. c] - $C(f^*) = C(f)$.

Reason: difference in costs on e is



typical value of $c_e(f_e)f_e^* - l_e(f_e)f_e^*$

$\Rightarrow$ $c_e(f_e^*)f_e^* - l_e(f_e^*)f_e^* = l_e(f_e)f_e$

sum over edges to get Claim

Linear Latency Functions

Approach #2: restrict class of allowable latency functions

Def: a linear latency function is of the form $l_e(x)=a_e x+b_e$

Theorem: [Roughgarden/Tardos 00]

network w/linear latency fns Þ

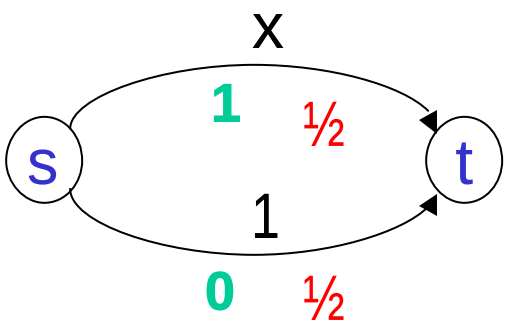
cost of Nash flow = 4/3 × cost of opt flow

aka price of anarchy [Papadimitriou 01]

Sources of Inefficiency

Corollary of main Theorem:

- For linear latency fns, worst Nash/OPT ratio is realized in a two-link network!



- Cost of Nash = 1
- Cost of OPT = ¾

- one source of inefficiency:
  - confronted w/ two routes, selfish users overcongest one of them
- Corollary ⇒ that's all, folks!
  - network topology plays no role

No Dependence on Network Topology

Thm: [Roughgarden 02] for any class of convex latency fns including the constant fns, worst Nash/OPT ratio occurs in a two-node, two-link network.

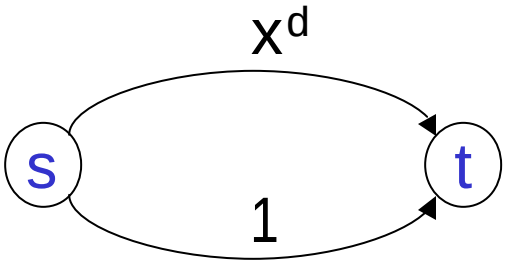
- inefficiency of Nash flows always has simple explanation
- network topology plays no role

Recall: worst ratio may be (much) larger than 4/3 (modify Pigou's ex)

Computing the Price of Anarchy

Application: worst-case examples simple ⇒ worst-case ratio is easy to calculate

Example: polynomials with degree = d, nonnegative coeffs ⇒ price of anarchy $\Theta(d/\log d)$



Also: M/M/1, M/G/1 queue delay fns, etc.

A Convex Program for the Optimal Flow

Key Lemma: [BMW 56] the opt flow is just a Nash flow w.r.t. a different set of latency fns.

Idea: the optimal flow is minimizer for a convex program:

$$\text{Min} \quad C(f) = \sum_e f_e \cdot l_e(f_e)$$

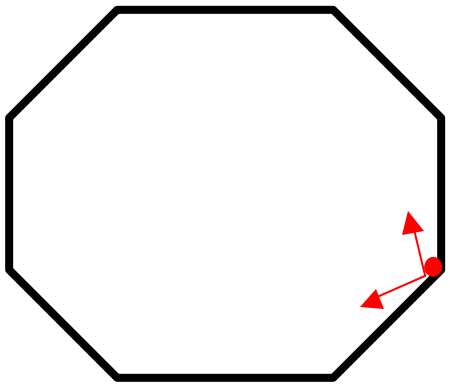
$$\text{s.t.} \quad f \text{ is a flow}$$

Question: What does this buy us?

Optima of Convex Programs

Recall: A solution is optimal for a convex program if and only if:

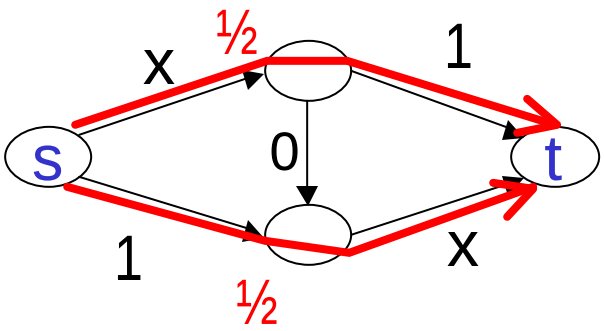
- a small change in a locally feasible direction cannot decrease the cost



feasible directions

Characterizing the Optimal Flow

Direction of change: moving a small amount of flow from one path to another



a flow f is optimal if and only if its cost cannot be improved by moving a small amount of flow from one path to another

Characterizing the Optimal Flow

Cost $f_e \cdot l_e(f_e)$ Þ marginal cost of increasing flow on edge e is

$$l_e(f_e) + f_e \cdot l_e'(f_e)$$

latency of new flow

Added latency for flow already using edge

Key Lemma: a flow f is optimal if and only if all flow travels along paths with minimum marginal cost (w.r.t. f).

Optimal Flow as a Socially Aware Nash Flow

A flow f is optimal if and only if all flow travels along paths with minimum marginal cost

Marginal cost: $l_e(f_e) + f_e \bullet l_e'(f_e)$

A flow f is at Nash equilibrium if and only if all flow travels along minimum latency paths

Latency: $l_e(f_e)$

Summary of Nash vs. OPT

Goal: prove that loss in network performance due to selfish routing is not too large.

Problem: a Nash flow can cost arbitrarily more than an optimal flow.

Solutions:

- prove a bicriteria bound instead
- restrict class of allowable edge latency functions

Results on Nash vs. OPT

Thm 1: cost of Nash = cost of OPT at twice the traffic rates.

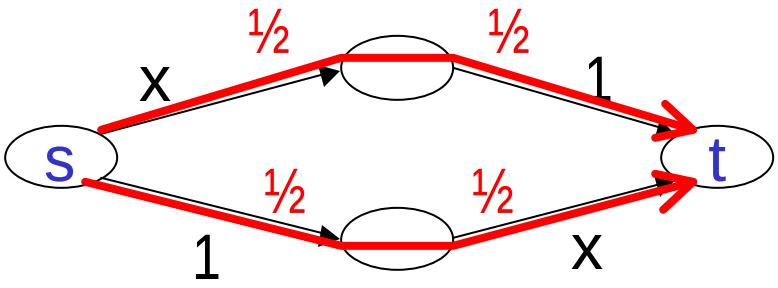
- Key to analysis: lower-bound OPT using modified latency functions (easy to obtain lower bound, but also close to original latency fns)

Thm 2: worst-case examples for selfish routing are simple.

- Worst-case Nash/OPT ratio small unless latency fns highly nonlinear
- Key to analysis: OPT is just a Nash flow w.r.t. different latency fns

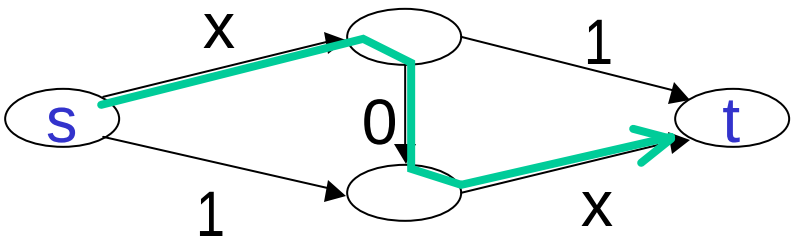
Braess's Paradox

Recall:



rate = 1

Cost of Nash flow = 1.5



Cost of Nash flow = 2

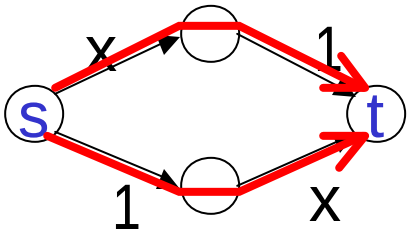
Þ Nash flow suffers due to extra edge

Generalizing Braess's Paradox

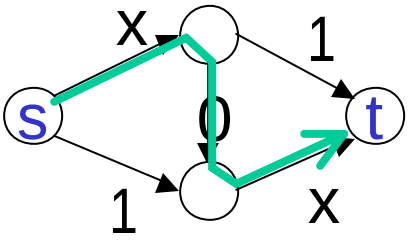
Question: is Braess's Paradox more severe in bigger networks?

- focus on single-commodity networks

Fact: w/linear fns, worst case is



vs.



cost = 3/2

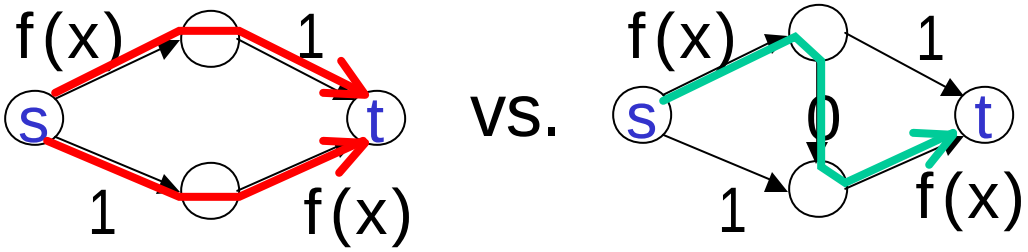
cost = 2

Reason: with linear latency fns,

cost of Nash flow = 4/3 × cost of any other flow

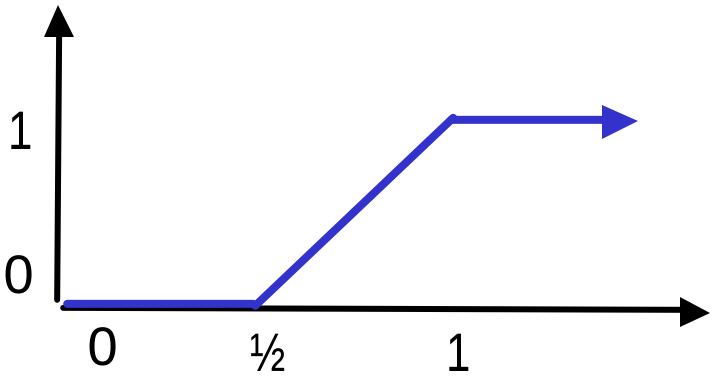
General Latency Fns

A more severe version:



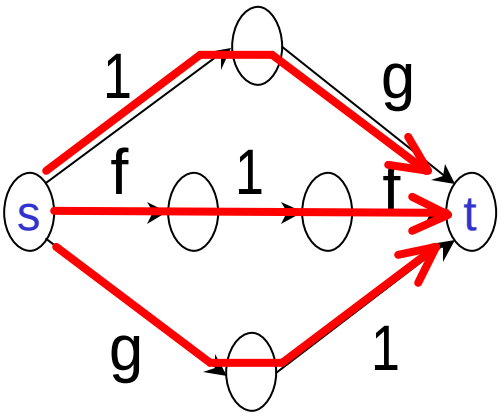
cost = 1 cost = 2

where f(x) is:



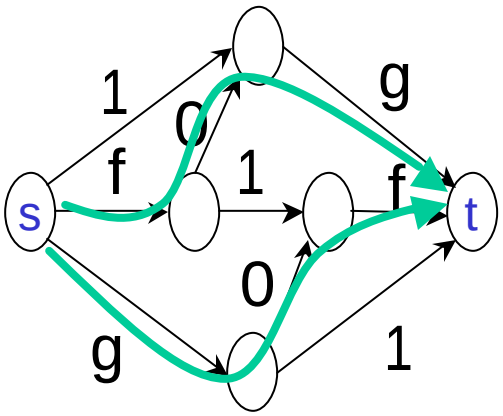
Þ extra edge causes factor 2 increase in delay

A Bigger Braess Paradox



vs.

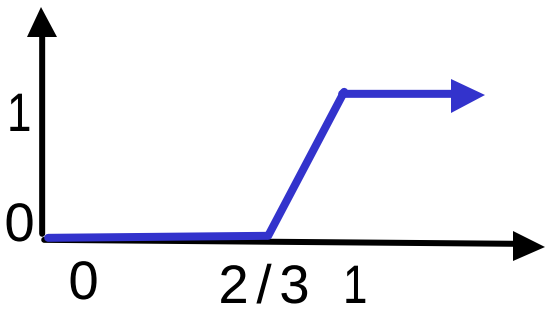
rate = 2



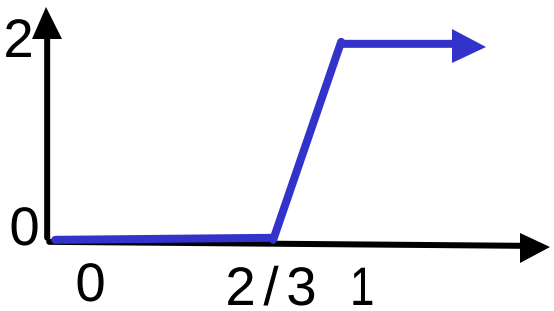
common latency = 1

common latency = 3

where:

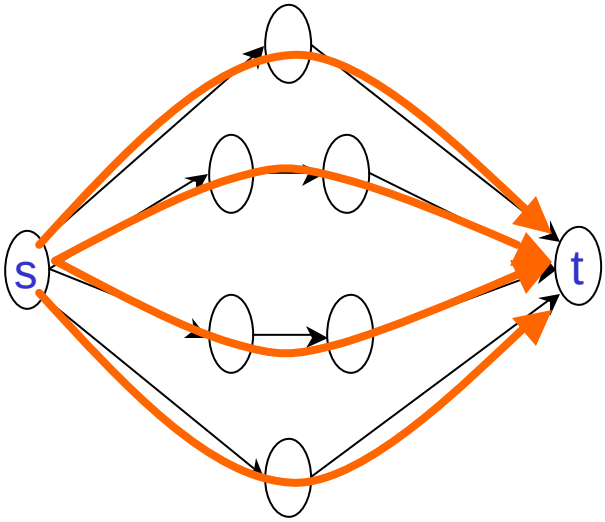


f(x)

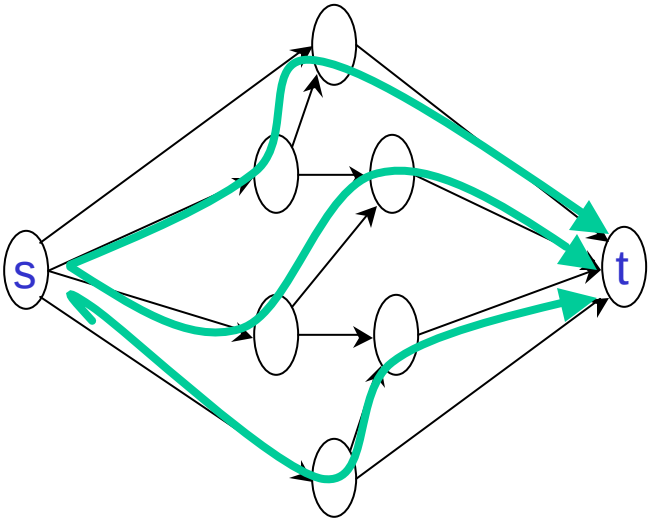


g(x)

An Infinite Family



common latency = 1



common latency = 4

Thm: [Roughgarden 01] adding edges to an n-vertex graph can increase common delay in the Nash flow by an $n/2$ factor.

A Matching Upper Bound

Thm: [Roughgarden 01] adding edges to an n-vertex graph increases common delay by at most an $n/2$ factor.

Note: cannot appeal to a result relating the total latency of Nash and OPT flows

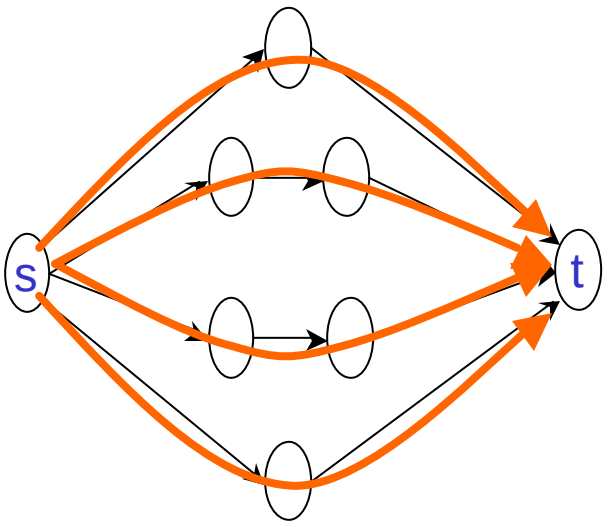
- no such bound exists with general latency fns
  - even as a fn of the network size

Summary of Braess's Paradox

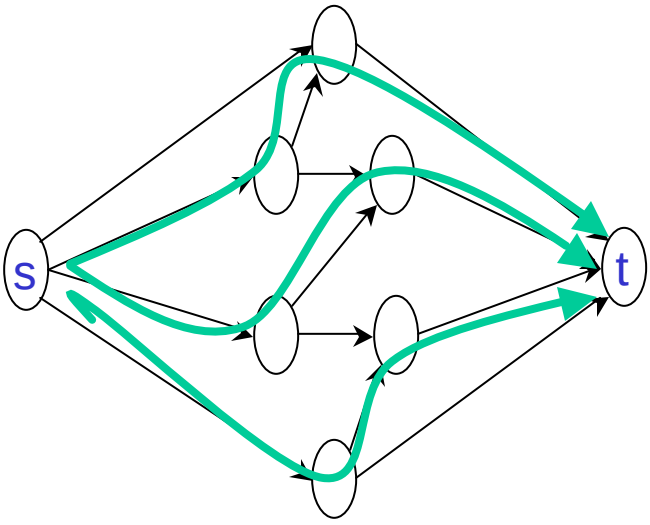
- **Braess's Paradox:** adding edges to a network can make the Nash flow worse.
- **Generalization:** can add edges to an n-node graph to make Nash worse by an $n/2$ factor.
- **Worst-case guarantee:** no worse examples are possible.

Open Question 1

Open: by adding only k edges, can you make the Nash flow worse by a factor of more than k+1?



common latency = 1



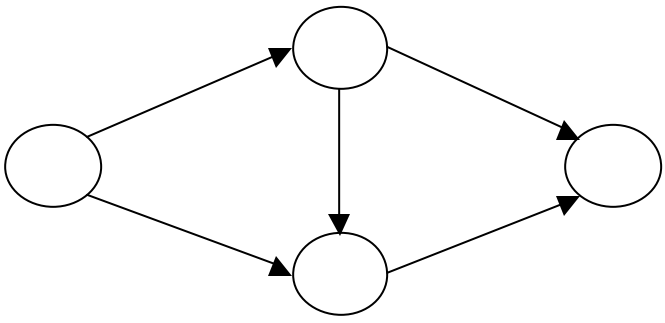
common latency = 4

Braess's Paradox and Embedded Subgraphs

Def: A graph is vulnerable if latency fns can be assigned to the edges so that Braess's Paradox occurs.

Fact: [Murchland 70]

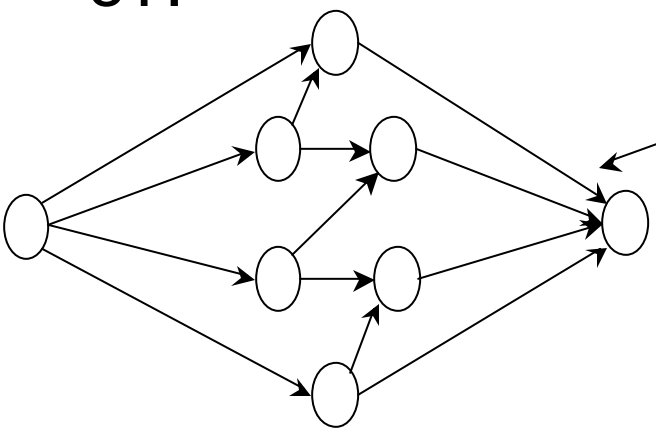
A graph is vulnerable if and only if it has a subgraph that is a subdivision of:



Open Question 2

Def: A graph is c-vulnerable if latency fns can be assigned to the edges so that Nash is worse by a factor $> c$ because of harmful extra edges.

Prove or disprove: A graph is c-vulnerable if and only if it has a subgraph that is a subdivision of:



size depends on c only