Designing Networks for Selfish Users is Hard*

Tim Roughgarden[†]

January 13, 2002

Abstract

We consider a directed network in which every edge possesses a latency function specifying the time needed to traverse the edge given its congestion. Selfish, noncooperative agents constitute the network traffic and wish to travel from a source s to a sink t as quickly as possible. Since the route chosen by one network user affects the congestion (and hence the latency) experienced by others, we model the problem as a noncooperative game. Assuming each agent controls only a negligible portion of the overall traffic, Nash equilibria in this noncooperative game correspond to s-t flows in which all flow paths have equal latency.

A natural measure for the performance of a network used by selfish agents is the common latency experienced by each user in a Nash equilibrium. It is a counterintuitive but well-known fact that removing edges from a network may *improve* its performance; the most famous example of this phenomenon is the so-called *Braess's Paradox*. This fact motivates the following network design problem: given such a network, which edges should be removed to obtain the best possible flow at Nash equilibrium? Equivalently, given a large network of candidate edges to be built, which subnetwork will exhibit the best performance when used selfishly?

We give optimal inapproximability results and approximation algorithms for several network design problems of this type. For example, we prove that for networks with n vertices and continuous, nondecreasing latency functions, there is no approximation algorithm for this problem with approximation ratio less than $n/2$ (unless $P = NP$). We also prove this hardness result to be best possible by exhibiting an $n/2$-approximation algorithm. For networks in which the latency of each edge is a linear function of the congestion, we prove that there is no $(\frac{4}{3} - \epsilon)$-approximation algorithm for the problem (for any $\epsilon > 0$, unless $P = NP$); the existence of a $\frac{4}{3}$-approximation algorithm follows easily from existing work, proving this hardness result sharp.

Moreover, we prove that an optimal approximation algorithm for these problems is what we call the *trivial algorithm*: given a network of candidate edges, build the entire network. A consequence of this result is that Braess's Paradox (even in its worst-possible manifestation) is impossible to detect efficiently.

*A preliminary version of this paper appeared in the Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, October 2001.

[†]Department of Computer Science, Cornell University, Ithaca NY 14853. Supported by an NSF Graduate Fellowship, a Cornell University Fellowship, and ONR grant N00014-98-1-0589. Email: `timr@cs.cornell.edu`.

1 Introduction

Selfish Routing

A central and well-studied problem arising in the management of a large network is that of routing traffic to achieve the best possible network performance. Recently, researchers have started to confront the harsh reality that in many networks, it is difficult or even impossible to impose optimal or near-optimal routing strategies on network traffic, leaving network users free to act according to their own interests. For example, some Internet protocols place little restriction on how network traffic is routed, allowing network users to behave in a selfish or even malicious manner [3]. This realization has led many authors (e.g., [7, 17, 20, 27, 33, 35]) to model the behavior of network users with *noncooperative games* and to study the resulting *Nash equilibria* (see Owen [28], for example, for an introduction to basic game-theoretic concepts).

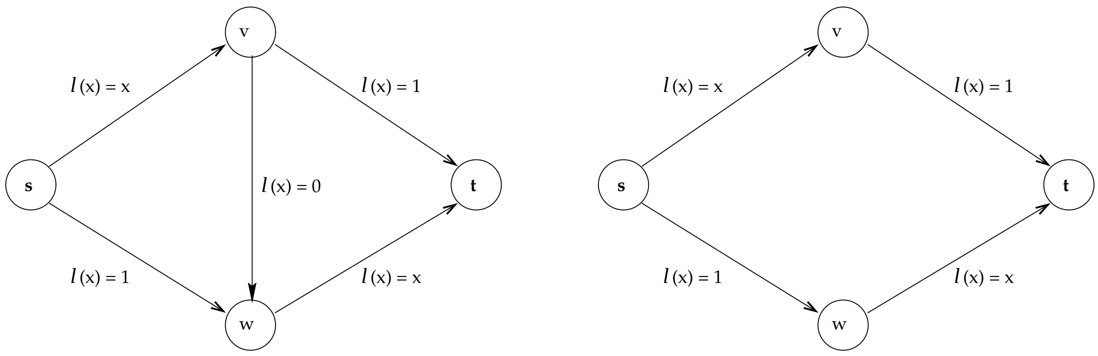
Motivated by the well-known fact that Nash equilibria may be *inefficient* (i.e., they need not optimize natural global objective functions [9]), researchers have proposed several different ways of *coping with selfishness* — that is, for ensuring that selfish behavior results in a desirable outcome. For example, previous approaches include bounding the worst-possible inefficiency of Nash equilibria (also known as "the price of anarchy" [29]) [6, 20, 23, 31, 33], and influencing the behavior of selfish agents via pricing policies [5], network switch protocols [35], routing a small portion of the traffic centrally [17, 32], or algorithmic mechanism design [10, 25, 26].

In this paper, we explore a different idea for ameliorating the degradation in network performance due to selfish routing: armed with the knowledge that our networks will be host to selfish users, how can we design them to minimize the inefficiency inherent in a user-defined equilibrium?

Braess's Paradox and Network Design

We consider a directed network in which each edge possesses a latency function specifying the time needed to traverse the edge given its congestion, and assume that all network traffic wishes to travel from a distinguished source vertex s to a sink vertex t. Selfish, noncooperative agents constitute the network traffic, and each wishes to travel from s to t as quickly as possible. Since the route chosen by one network user affects the congestion (and hence the latency) experienced by others, it will be useful to view the problem as a noncooperative game. Assuming each agent controls only a negligible portion of the overall traffic, an assignment of traffic to paths in the network can be modeled as *network flow*, with a Nash equilibrium in the noncooperative game corresponding to an s-t flow in which all flow paths have equal (and minimum-possible) latency (if a flow does not have this property, some agent can improve its travel time by switching from a longer flow path to a shorter one).

A natural measure for the performance of a network used by selfish agents is the common latency experienced by each user in a Nash equilibrium, as it navigates from s to t. It is a counterintuitive but well-known fact that removing edges from a network may *improve* its performance; this phenomenon is best illustrated by *Braess's Paradox*, as shown in Fig-



(a) The Whole Graph

(b) The Optimal Subgraph

Figure 1: Braess's Paradox

ure 1. In the figure, each edge is labeled with its latency function, as a function of the edge congestion x; for example, if there are two units of flow on the edge (s, v), then all of this flow experiences a latency of two when traversing the edge. Now suppose that one unit of flow needs to be routed from s to t in the network of Figure 1(a). In the unique flow at Nash equilibrium, all traffic follows the path $s \to v \to w \to t$ and experiences a latency of 2 (since the other two s-t paths also have latency 2 with respect to this flow, no user has an incentive to switch paths). On the other hand, suppose we remove the edge (v, w), thereby obtaining the network of Figure 1(b). Then, in the unique flow at Nash equilibrium, half of the flow travels on the upper path and the rest travels along the lower path; here, all agents experience a latency of $\frac{3}{2}$.

Braess's Paradox immediately suggests the following network design problem: given a network with latency functions on the edges and a traffic rate, which edges should be removed to obtain the best possible flow at Nash equilibrium? Equivalently, given a large network of candidate edges to build, which subnetwork will exhibit the best performance when used selfishly?

This problem is fundamentally different from most well-studied network design problems (such as those described by Goemans and Williamson [14]), which typically ask for the cheapest network satisfying certain desiderata such as high connectivity or small diameter. Problems of this sort are only non-trivial in the presence of costs on vertices and/or edges; otherwise, the best solution is to simply build the largest possible network. On the other hand, Braess's Paradox shows this approach to be suboptimal for our network design problem; even in the absence of costs, it is not at all clear which network should be preferred.

Ever since Braess's Paradox was reported [4, 24], researchers have attempted to solve variants of this network design problem (for example, one is alluded to in the work of Dafermos and Sparrow [7]), but scant progress has been made either computationally or theoretically. Indeed, early computational work either focused on very small networks [21] or admitted to ignoring congestion effects entirely, due to the difficulties involved [2, 8, 16, 30, 34]; in a 1984 survey, Magnanti and Wong describe the problem as "essentially unsolved" from a practical perspective [22, P.15]. On the theoretical side, there has been work partially classifying the network topologies and latency functions for which the deletion of a single edge can be helpful [12, 18, 36] and showing that certain edge deletion strategies cannot improve network performance [19], but no reported results on the general network design problem. More recently, this problem has received attention from the theoretical computer science community, and several researchers have asked if there are efficient exact or near-optimal algorithms for the problem.

Designing networks for selfish users has thus appeared difficult from a range of perspectives and to several research communities. In this paper, we present a theoretical explanation for this perceived difficulty.

Our Results

We give optimal inapproximability results and approximation algorithms for several network design problems of the following type: given a network with edge latency functions, a single source-sink pair, and a rate of traffic, find the subnetwork minimizing the travel time of all (selfish) network users in a flow at Nash equilibrium. Specifically, we prove the following for any $\epsilon > 0$ (assuming $P \neq NP$):

- General Latency Network Design: for networks with continuous, nonnegative, nondecreasing edge latency functions, there is no $(n/2 - \epsilon)$-approximation algorithm[1] for network design, where n is the number of vertices in the network. We also prove this hardness result to be best possible by exhibiting an $n/2$-approximation algorithm for the problem.

- Linear Latency Network Design: for networks in which the latency of each edge is a linear function of the congestion, there is no $(\frac{4}{3} - \epsilon)$-approximation algorithm for network design. The existence of a $\frac{4}{3}$-approximation algorithm follows easily from existing work, proving this hardness result optimal.

To the best of our knowledge, these problems were not previously known to be NP-hard.

Moreover, we prove that an optimal approximation algorithm for these problems is what we call the *trivial algorithm*: given a network of candidate edges, build the entire network. As a consequence of the optimality of the trivial algorithm, we prove that Braess's Paradox (i.e., the presence of harmful extraneous edges) is impossible to detect efficiently, even in its worst-possible manifestation.

Finally, we show that our strong hardness results are not particular to the classes of general and linear latency functions; rather, for several additional classes of latency functions (such as polynomials of bounded degree) the trivial algorithm achieves the best possible performance guarantee (up to a constant factor).

[1]A *c-approximation algorithm* for a minimization problem runs in polynomial time and returns a solution no more than c times as costly as an optimal solution. The value c is the *approximation ratio* or *performance guarantee* of the algorithm.

Organization

In Section 2 we formally define our model of traffic equilibria in networks, review several important properties of the model, and prove some preliminary results needed in subsequent sections. In Sections 3-6, we prove matching upper and lower bounds on the approximability of network design for several classes of edge latency functions. Linear latency functions are considered in Section 3, general (continuous and nondecreasing) latency functions in Section 4, polynomials latency functions in Section 5, and a broader class of well-behaved latency functions in Section 6.

2 Preliminaries

2.1 The Model

We consider a directed network $G = (V, E)$ with vertex set V, edge set E, and a distinguished source vertex s and sink vertex t. We allow multiple edges between vertices but have no use for self-loops. We denote the set of (simple) s-t paths by $\mathcal{P}$, which we assume to be non-empty. A *flow* is a function $f : \mathcal{P} \to \mathcal{R}^+$; for a fixed flow f we define $f_e = \sum_{P:e\in P} f_P$. With respect to a finite and positive *traffic rate* r, a flow f is said to be *feasible* if $\sum_{P\in\mathcal{P}} f_P = r$. Each edge $e \in E$ possesses a load-dependent *latency* that we denote by $\ell_e(\cdot)$. The latency of a path P with respect to a flow f is then the sum of the latencies of the edges in the path, denoted by $\ell_P(f) = \sum_{e\in P} \ell_e(f_e)$. For each edge $e \in E$, we assume that the latency function ℓ_e is nonnegative, continuous, and nondecreasing. We call the triple (G, r, ℓ) an *instance*.

2.2 Flows at Nash Equilibrium

We will consider flows that represent an equilibrium among many non-cooperative agents—i.e., flows that behave in a "greedy" or "selfish" manner. Intuitively, we expect each unit of such a flow (no matter how small) to travel along the minimum-latency path available to it, where latency is measured with respect to the rest of the flow; otherwise, this flow would reroute itself on a path with smaller latency. Following [7, 33], we formalize this idea in the next definition.

Definition 2.1 *A flow f in G is at* Nash equilibrium *(or is a* Nash flow*) if for all $P_1, P_2 \in \mathcal{P}$ and $\delta \in [0, f_{P_1}]$, we have $\ell_{P_1}(f) \le \ell_{P_2}(\tilde{f})$, where*

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

Letting δ tend to 0, continuity and monotonicity of the edge latency functions give the following useful characterization of a flow at Nash equilibrium (first stated by Wardrop [38]).

Lemma 2.2 *A flow f is at Nash equilibrium if and only if for every $P_1, P_2 \in \mathcal{P}$ with $f_{P_1} > 0$, $\ell_{P_1}(f) \le \ell_{P_2}(f)$.*

In particular, if f is at Nash equilibrium then all s-t flow paths (i.e., s-t paths to which f assigns a positive amount of flow) have equal latency.

The following lemma states that, under our assumption of continuous, nondecreasing edge latency functions, flows at Nash equilibrium always exist and are essentially unique.

Lemma 2.3 ([1, 7, 33]) *An instance (G, r, ℓ) admits a feasible flow at Nash equilibrium. Moreover, if f, f' are feasible flows at Nash equilibrium for (G, r, ℓ), then $\ell_P(f) = \ell_P(f')$ for every s-t path P.*

2.3 Formalizing the Network Design Problem

By Lemmas 2.2 and 2.3, the following definition makes sense: for an instance (G, r, ℓ) admitting a (feasible) Nash flow f, we define $L(G, r, \ell)$ to be the common latency (with respect to f) of every s-t flow path of f. When no confusion results, we will abbreviate the expression $L(G, r, \ell)$ by $L(G)$.

We may thus formally state our network design problem as follows:

Given an instance (G, r, ℓ), find the subgraph H of G minimizing $L(H, r, \ell)$.

2.4 The Cost of a Flow

Our next preliminary result relates our objective function $L(H, r, \ell)$ to a second objective function that has been well-studied. This connection will be crucial for proving upper bounds on the performance guarantee of the trivial algorithm in Sections 3, 5, and 6.

Define the *cost* $C(f)$ of a flow f in G as the total latency incurred by f, that is,

$$C(f) = \sum_{P \in \mathcal{P}} \ell_P(f) f_P.$$

We immediately see that the cost of a flow at Nash equilibrium can be written in a particularly nice form.

Lemma 2.4 *If f is a feasible flow at Nash equilibrium for (G, r, ℓ), then*

$$C(f) = r \cdot L(G, r, \ell).$$

2.5 Minimum-Latency Paths and Acyclicity of Nash Flows

The goal of this final preliminary subsection is to prove that every instance (G, r, ℓ) admits a Nash flow without flow cycles (thereby strengthening the existence guarantee of Lemma 2.3). Along the way, we will prove some useful properties about the structure of minimum-latency paths with respect to a Nash flow. The results of this subsection are needed only in Subsection 4.1, where we prove an upper bound on the performance guarantee of the trivial algorithm for the General Latency Network Design problem.

We begin with an extension of Lemma 2.2. While Lemma 2.2 implies that all s-t flow paths of a flow at Nash equilibrium have minimum-possible latency, the following lemma implies the same statement with s and t replaced by an arbitrary pair of vertices.

Lemma 2.5 *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 $\ell_e(f_e)$, of a shortest s-v path in G. Then f is at Nash equilibrium if and only if for every pair v, w of vertices in G and every v-w path P:*

(a) $d(w) - d(v) \leq \sum_{e \in P} \ell_e(f_e)$

(b) if $f_e > 0$ for every edge $e \in P$, then $d(w) - d(v) = \sum_{e \in P} \ell_e(f_e)$.

Proof: First suppose f is feasible for (G, r, ℓ) and satisfies the two conditions of the lemma statement. Taking $v = s$ and $w = t$, properties (a) and (b) imply that every s-t flow path of f has minimum latency among all s-t paths (namely, $d(t)$). By Lemma 2.2, we conclude that f is at Nash equilibrium.

Conversely, suppose f is at Nash equilibrium for (G, r, ℓ). It suffices to prove that properties (a) and (b) hold when P is a single edge (for a general path, sum up the inequalities or equalities corresponding to the constituent edges). Then, (a) follows by definition of $d(v)$ and $d(w)$. To prove (b), consider an edge e with $f_e > 0$ and suppose for contradiction that $d(w) < d(v) + \ell_e(f_e)$. Let P_e denote an s-t path containing e with $f_{P_e} > 0$. We may obtain another s-t path P' via the union of a shortest s-w path and the w-t path contained in P_e. Since the latency of the s-w path contained in P_e is at least $d(v) + \ell_e(f_e) > d(w)$, we have $\ell_P(f) > \ell_{P'}(f)$, contradicting Lemma 2.2. ■

It is important to note that the path P in the statement of Lemma 2.5 does *not* need to be a subpath of any flow path of f; in particular, in property (b) the flow on different edges of P can be carried by distinct flow paths of f.

Call a flow f feasible for an instance (G, r, ℓ) *acyclic* if the subgraph of edges e for which $f_e > 0$ is a directed acyclic graph. We can now prove that every network with continuous, nondecreasing edge latency functions admits an acyclic Nash flow.

Lemma 2.6 *An instance (G, r, ℓ) admits an acyclic flow at Nash equilibrium.*

Proof: An instance (G, r, ℓ) admits a (not necessarily acyclic) Nash flow f by Lemma 2.3. We will first show that cycles of flow edges must comprise only zero-latency edges, and will then show how to remove such cycles.

Define the s-v distance $d(v)$ of a vertex v with respect to the flow f as in Lemma 2.5. By Lemma 2.5 and nonnegativity of edge latencies, if edge $e = (v, w)$ carries flow then $d(w) \geq d(v)$. Thus, in a directed cycle C of flow edges, all vertices of C have equal d-values and hence (again by Lemma 2.5) all edges of C must have zero latency with respect to f.

We next wish to remove zero-latency flow cycles from f; this is not entirely trivial as the flow on different edges of a flow cycle may be carried by different flow paths (recall f is defined as a function on paths, rather than on edges). Extract a new feasible flow $\hat{f}$ from f as follows: view f as a function on edges with $f_e = \sum_{P: e \in P} f_P$, repeatedly discard flow cycles from f to obtain an acyclic flow f' (still defined only on edges), and let $\hat{f}$ be an arbitrary path decomposition of f' (see Tarjan [37] for details on removing flow cycles and on path decompositions). The flow $\hat{f}$ is acyclic by construction and is feasible for (G, r, ℓ) since only flow cycles were removed from the feasible flow f; it remains only to show that $\hat{f}$ is at Nash equilibrium. For each edge e, we have either $f_e = \hat{f}_e$ or $\hat{f}_e < f_e$ with $\ell_e(f_e) = 0$

(and hence $\ell_e(\hat{f}_e) = 0$). It follows that $\ell_e(\hat{f}_e) = \ell_e(f_e)$ for every edge e, which in turn implies that the flows f and $\hat{f}$ induce identical d-values on the vertices of G. Appealing to the characterization of Nash flows given in Lemma 2.5, that f is a Nash flow implies that $\hat{f}$ is, as well. ■

3 Linear Latency Functions: An Approximability Threshold of $\frac{4}{3}$

We begin with the setting in which the latency of every edge of the network is a linear function of the congestion (that is, each latency function ℓ_e may be written $\ell_e(x) = a_e x + b_e$ for $a_e, b_e \geq 0$). This is a commonly studied scenario [7, 12, 33], and our proof of inapproximability is particularly simple in this special case.

Recall that the *trivial algorithm*, when presented with instance (G, r, ℓ), outputs the network G (i.e., always decides to build the entire network). That the trivial algorithm is a $\frac{4}{3}$-approximation algorithm for Linear Latency Network Design will follow easily from the next result, previously proved by Roughgarden and Tardos [33]. The proposition states that in any network with linear latency functions, the total latency of a flow at Nash equilibrium is at most $\frac{4}{3}$ times that of any other feasible flow.

Proposition 3.1 ([33]) *Suppose (G, r, ℓ) is an instance with linear latency functions for which f^* is a feasible flow and f is a flow at Nash equilibrium. Then $C(f) \leq \frac{4}{3} C(f^*)$.*

Corollary 3.2 *The trivial algorithm is a $\frac{4}{3}$-approximation algorithm for* Linear Latency Network Design.

Proof: Consider any instance (G, r, ℓ) with linear latency functions, with subgraph H minimizing $L(H, r, \ell)$. Let f and f^* denote flows at Nash equilibrium for (G, r, ℓ) and (H, r, ℓ), respectively. By Lemma 2.4, we may write $C(f) = r \cdot L(G, r, \ell)$ and $C(f^*) = r \cdot L(H, r, \ell)$. Since f^* is also feasible for (G, r, ℓ), Proposition 3.1 implies that $C(f) \leq \frac{4}{3} C(f^*)$ and hence $L(G, r, \ell) \leq \frac{4}{3} L(H, r, \ell)$. ■

The main result of this section is that, unless $P = NP$, no better approximation is possible in polynomial time.

Theorem 3.3 *For any $\epsilon > 0$, there is no $(\frac{4}{3} - \epsilon)$-approximation algorithm for* Linear Latency Network Design *unless $P = NP$.*

Proof: We will make use of the problem 2 Directed Disjoint Paths (2DDP): 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? This problem was proved NP-complete by Fortune et al. [11]. We will show that a $(\frac{4}{3} - \epsilon)$-approximation algorithm for Linear Latency Network Design can be used to distinguish "yes" and "no" instances of 2DDP in polynomial time.

Consider an instance $\mathcal{I}$ of 2DDP, as above. Augment the vertex set V by an additional source s and sink t, and include directed edges $(s, s_1), (s, s_2), (t_1, t), (t_2, t)$ (see Figure 2).

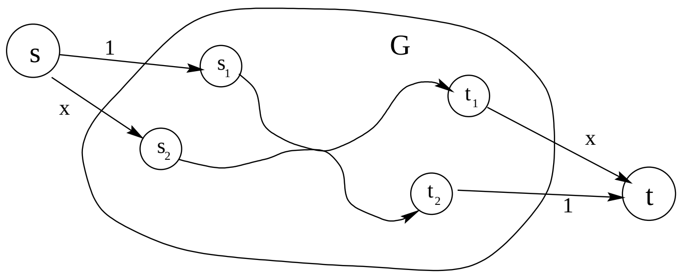


Figure 2: Proof of Theorem 3.3. In a "no" instance of 2DDP, existence of s_1-t_1 and s_2-t_2 paths implies the existence of an s_2-t_1 path.

Denote the new network by $G' = (V', E')$ and endow the edges of E' with linear latency functions ℓ as follows: all edges of E are given the latency function $\ell(x) = 0$, edges (s, s_2) and (t_1, t) are given the latency function $\ell(x) = x$, and edges (s, s_1) and (t_2, t) are given the latency function $\ell(x) = 1$.

To complete the proof, it suffices to show the following two statements: (i) if $\mathcal{I}$ is a "yes" instance of 2DDP, then there is a subgraph H of G' satisfying $L(H, 1, \ell) = \frac{3}{2}$; (ii) if $\mathcal{I}$ is a "no" instance, then for any subgraph H of G', $L(H, 1, \ell) \geq 2$.

To prove (i), let P_1 and P_2 be vertex-disjoint s_1-t_1 and s_2-t_2 paths in G, respectively, and obtain H by deleting all edges of G not contained in some P_i. Then, H is a subgraph of G' with exactly two s-t paths, and routing half a unit of flow along each yields a flow at Nash equilibrium in which each path has latency $\frac{3}{2}$ (cf., Figure 1(b)).

For (ii), we may assume that H contains an s-t path. If H has an s-t path P containing an s_2-t_1 path, then define a flow f by routing a single unit of flow on P; this is a flow at Nash equilibrium, with respect to which every s-t path has latency 2 (cf., Figure 1(a)), so $L(H) = 2$. Otherwise, since $\mathcal{I}$ is a "no" instance, there are only two remaining possibilities (see Figure 2): either for precisely one $i \in \{1, 2\}$, H has an s-t path P containing an s_i-t_i path, or all s-t paths P in H contain an s_1-t_2 path of G. In either case, routing one unit of flow along such a path P provides a flow at Nash equilibrium showing that $L(H) = 2$. ■

Corollary 3.2 and Theorem 3.3 imply that efficiently detecting Braess's Paradox (i.e., detecting whether or not network performance is hampered by harmful extraneous edges) in networks with linear latency functions is impossible, even in instances suffering from the most severe manifestations of the paradox. To make this statement precise, call an instance (G, r, ℓ) with linear latency functions *paradox-free* if $L(H, r, \ell) \geq L(G, r, \ell)$ for all subgraphs H of G (i.e., if the entire network is an optimal subnetwork) and *paradox-ridden* if for some subgraph H in G, $L(H, r, \ell) = \frac{3}{4} L(G, r, \ell)$. By Corollary 3.2, paradox-ridden instances are precisely those incurring a worst-possible loss in network performance due to Braess's Paradox. The construction in the proof of Theorem 3.3 then gives the following corollary.

Corollary 3.4 *Given an instance (G, r, ℓ) with linear latency functions that is either paradox-free or paradox-ridden, it is NP-complete to decide whether or not (G, r, ℓ) is paradox-ridden.*

4 General Latency Functions: An Approximability Threshold of $\lfloor n/2 \rfloor$

In this section we consider the problem of network design with the broadest possible class of latency functions (assuming we insist on the existence and uniqueness of flows at Nash equilibrium), the set of all continuous nondecreasing functions. We begin by proving in Subsection 4.1 that the trivial algorithm achieves an approximation ratio of $\lfloor n/2 \rfloor$, where n in the number of vertices in the network (in contrast with other sections, this performance guarantee does not trivially follow from previous work). In Subsection 4.2, we introduce a new family of graphs generalizing the network of the original Braess's Paradox (Figure 1(a))[2], and we conclude in Subsection 4.3 by using this family to prove an optimal hardness result matching the upper bound provided by the trivial algorithm.

4.1 An $\lfloor n/2 \rfloor$-Approximation Algorithm

Our goal in this subsection is to prove that the trivial algorithm is an $\lfloor n/2 \rfloor$-approximation algorithm for General Latency Network Design, where n is the number of vertices in the network. Before embarking on the proof, it is important to contrast the settings of general and linear latency functions. In particular, we saw in the proof of Corollary 3.2 that a known result upper bounding the total latency of a Nash flow relative to any other feasible flow immediately yielded an identical upper bound on the performance of the trivial algorithm. Thus, if we knew that a Nash flow in a network with n vertices and general latency functions was at most $f(n)$ times as costly (with respect to the total latency measure) as any other feasible flow for some "nice" (e.g., linear) function $f(\cdot)$, we would be done.[3] Unfortunately, no such result can hold: with general latency functions, a Nash flow may be *arbitrarily* more costly than other feasible flows, even in networks with only two vertices and two edges. To see why, we recapitulate an example from [33]: consider a network G with vertices s and t and two edges e_1, e_2 with latency functions $\ell_1(x) = 1$ and $\ell_2(x) = x^k$ for some very large integer k. Setting the traffic rate r to 1, we see that the cost of a Nash flow is 1 (all flow is routed on e_2 in the flow at Nash equilibrium) and that there is a feasible flow in G with very small cost: routing a near-zero amount of the flow on e_1 and the rest on e_2 yields a flow of near-zero cost (for large k).

However, this fact is not due cause for abandoning the goal of proving some kind of performance guarantee for the trivial algorithm; it merely indicates that a more delicate approach is required. In the previous example, the flow with near-zero cost was far from at equilibrium: a few martyrs were routed on edge e_1 for the benefit of the overwhelming majority of the flow. Indeed, all (non-empty) subgraphs H of G satisfy $L(H) = 1$. Thus, while any subgraph provides an optimal solution to our network design problem, we have no way of proving any finite approximation ratio!

By comparing the output of the trivial algorithm only to feasible flows at equilibrium in a subgraph of G (rather than to *all* feasible flows), we obtain the main result of this subsection.

---

[2] This family may be of independent interest, as (to the best of our knowledge) these networks give the first demonstration that the severity of Braess's Paradox can increase with the network size.

[3] Indeed, this argument will reoccur in Sections 5 and 6.

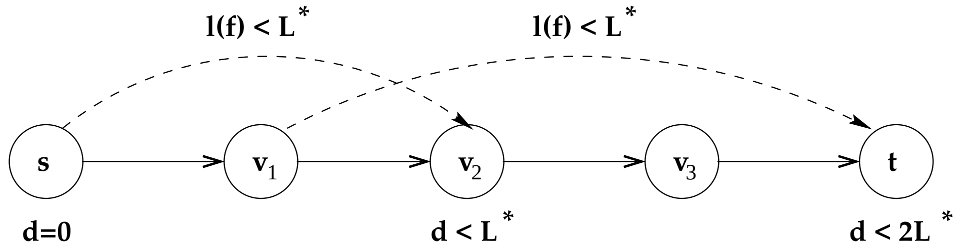


Figure 3: Proof of Theorem 4.1. If f is the flow sending one unit of flow on the four-hop path and f^* is the flow sending half a unit of flow on each of the other two paths, then the dashed edges are light.

Theorem 4.1 *For any instance (G, r, ℓ) with $|V(G)| = n$, the trivial algorithm returns a solution of value at most $\lfloor \frac{n}{2} \rfloor$ times that of the optimal solution.*

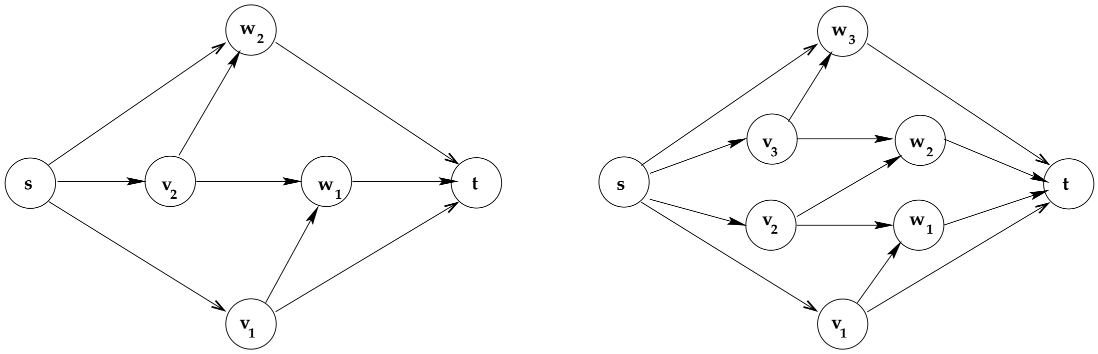
Proof: Let f and f^* be flows at Nash equilibrium for (G, r, ℓ) and (H, r, ℓ), respectively, with H a subgraph of G containing an s-t path. By Lemmas 2.3 and 2.6, we may assume that f is acyclic. Put $L = L(G, r, \ell)$ and $L^* = L(H, r, \ell)$; we wish to prove that $L \leq \lfloor n/2 \rfloor \cdot L^*$.

The rest of the proof will make crucial use of Lemma 2.5. Accordingly, define $d(v)$ for $v \in V(G)$ as in Lemma 2.5, as the length (with respect to edge lengths $\ell_e(f_e)$) of a shortest s-v path. Assume for simplicity that n is odd and that every vertex of G is incident to a edge e with $f_e > 0$ (extending the following argument to the general case is straightforward) and order the vertices $s = v_0, v_1, \ldots, v_{n-1} = t$ according to nondecreasing $d(v)$-value. If there is an edge $e = (v, w)$ with $f_e > 0$ and $\ell_e(f_e) = 0$ (so, by Lemma 2.5, $d(v) = d(w)$), break the tie by placing v before w in the ordering; this will always be possible since f is acyclic. Lemma 2.5 implies that this ordering is a topological one with respect to the flow f (i.e., whenever $f_e > 0$, e is a forward edge with respect to our ordering). Our proof approach will be to show, by induction on i, that $d(v_{2i}) \leq i \cdot L^*$ (with the base case $i = 0$ trivial).

Before considering the inductive step, we require a definition and a claim. Call an edge e *light* if $f_e \leq f_e^*$ and $f_e^* > 0$ (in particular, e must be present in H). Light edges are useful to us because they have latency at most L^* with respect to f^* (as every flow path of f^* has latency L^*) and hence latency at most L^* with respect to f (since latencies are nondecreasing); thus, vertices of G that are adjacent via a light edge differ in d-values by at most L^*. The next claim assures us of a healthy supply of light edges: every s-t cut consisting of a set of consecutive vertices (with respect to our topological ordering) contains a light edge (see Figure 3).

Claim: *Let $S = \{v_0, \ldots, v_k\}$ for some $k \in \{0, 1, \ldots, n-2\}$. Then some light edge has its tail in S and head outside of S.*

Proof: We require some basic notions from network flow theory (see, for example, Tarjan [37]). Let $\delta^+(S)$ denote the edges with tail inside S and head outside S, and $\delta^-(S)$ the edges with head inside S and tail outside S. Since S is an s-t cut and f is an s-t flow of value r with no flow on edges in $\delta^-(S)$ (as the vertices are topologically sorted according to f), $\sum_{e \in \delta^+(S)} f_e = r$. Since S is an s-t cut and f^* is an s-t flow, $\sum_{e \in \delta^+(S)} f_e^* \geq r$. Hence,



(a) B^2 (b) B^3

Figure 4: The second and third Braess graphs

$f_e \leq f_e^*$ for some $e \in \delta^+(S)$ with $f_e^* > 0$. ■

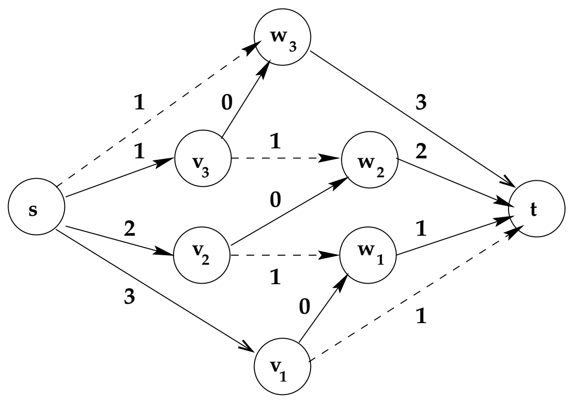
Now suppose $i \in \{1, \ldots, (n-1)/2\}$ and $d(v_{2(i-1)}) \leq (i-1)L^*$. Let k be the largest integer such that there is a path of light edges from v_j to v_k for some $j \leq 2(i-1)$; we will show that $k \geq 2i$. The previous claim immediately implies that k is well-defined with $k > 2(i-1)$ (consider the head of a light edge in $\delta^+(\{v_0, \ldots, v_{2(i-1)}\})$). To see that $k \geq 2i$, observe that if $k = 2i-1$ then all light edges in $\delta^+(\{v_0, \ldots, v_{2(i-1)}\})$ (and there must be one) have head v_{2i-1} and no light edge has tail v_{2i-1} (otherwise we would append such an edge to our maximal path), contradicting that $\delta^+(\{v_0, \ldots, v_{2i-1}\})$ must contain a light edge.

We have established the existence of a path P of light edges from v_j to v_k with $j \leq 2(i-1)$ and $k \geq 2i$. Inductively, we have $d(v_j) \leq d(v_{2(i-1)}) \leq (i-1)L^*$; since $d(v_{2i}) \leq d(v_k)$, we can finish the inductive step and the proof by showing that $d(v_k) - d(v_j) \leq L^*$ (informally, $d(v_{2(i-1)})$ and $d(v_{2i})$ are sandwiched between $d(v_j)$ and $d(v_k)$, so it suffices to upper bound the gap between the latter pair of numbers). Letting $d^*(v)$ denote the length of a shortest s-v path in H with respect to edge lengths $\ell_e(f_e^*)$, applying Lemma 2.5 to f^* in H yields $0 = d^*(s) \leq d^*(v_j) \leq d^*(v_k) \leq d^*(t) = L^*$. By Lemma 2.5, this implies that the latency of P with respect f^* is at most L^*; since all of these edges are light, it follows that the latency of P with respect to f is at most L^*. A final application of Lemma 2.5 then yields $d(v_k) - d(v_j) \leq L^*$, completing the inductive step and the proof. ■

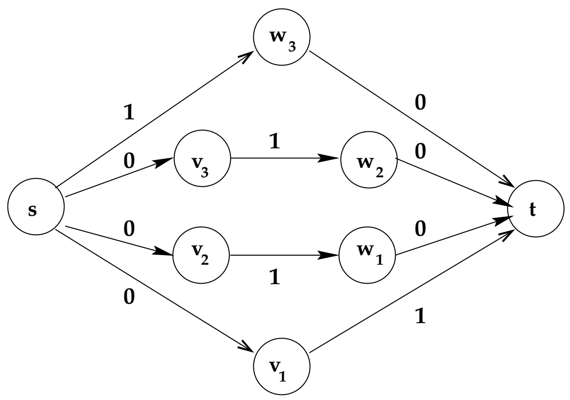
4.2 The Braess Graphs

We seek to prove a lower bound on the approximability of network design (and in particular, on the performance of the trivial algorithm) that is linear in the number of vertices of the network. Toward this end, we will first construct an infinite family of networks on which the trivial algorithm performs poorly (i.e., networks in which the value of a flow at Nash equilibrium can be vastly improved by removing some edges); hardness results (proved via similar but more involved arguments) will be presented in the next subsection.

We define the *kth Braess graph* B^k as follows: start with a set $V^k = \{s, v_1, \ldots, v_k, w_1, \ldots, w_k, t\}$



(a) Nash flow for $(B^3, 3, \ell^3)$



(b) Nash flow in the optimal subgraph

Figure 5: Proof of Proposition 4.2, when $k = 3$. Solid edges carry flow in the flow at Nash equilibrium, dashed edges do not. Edge latencies are with respect to flows at Nash equilibrium.

of $2k+2$ vertices and define E^k by $\{(s, v_i), (v_i, w_i), (w_i, t) : 1 \leq i \leq k\} \cup \{(v_i, w_{i-1}) : 2 \leq i \leq k\} \cup \{(v_1, t)\} \cup \{(s, w_k)\}$ (see Figure 4). We note that B^1 is the graph in which Braess's Paradox was first discovered (Figure 1(a)).

We next define latency functions ℓ^k for the edges of B^k; these functions will prove useful in Proposition 4.2 below. For each edge of the form $e = (v_i, w_i)$, put $\ell_e^k(x) = 0$; for an edge e of the form (v_i, w_{i-1}), (s, w_k), or (v_1, t), put $\ell_e^k(x) = 1$; for $i \in \{1, 2, \ldots, k\}$ and an edge e of the form (w_i, t) or (s, v_{k-i+1}), put $\ell_e^k(x)$ equal to any nonnegative, continuous, and nondecreasing function satisfying $\ell_e^k(\frac{k}{k+1}) = 0$ and $\ell_e^k(1) = i$ (thus, ℓ_e^k may be chosen to be convex and infinitely differentiable, if desired).

We can now show how to use the Braess graphs to construct instances on which the trivial algorithm for General Latency Network Design performs badly.

Proposition 4.2 *For any integer $n \geq 2$, there is an instance (G, r, ℓ) with $|V(G)| = n$ for which the trivial algorithm produces a solution with value at least $\lfloor \frac{n}{2} \rfloor$ times that of the optimal solution.*

Proof: We may suppose that n is even and at least four (for n odd, take a bad example for $n-1$ and add an isolated vertex). Write $n = 2k+2$ for $k \in \mathcal{N}$ and consider the instance (B^k, k, ℓ^k). For $i = 1, \ldots, k$, let P_i denote the path $s \to v_i \to w_i \to t$. For $i = 2, \ldots, k$, let Q_i denote the path $s \to v_i \to w_{i-1} \to t$; define Q_1 to be the path $s \to v_1 \to t$ and Q_{k+1} the path $s \to w_k \to t$. On one hand, routing one unit of flow on each of $P_1, \ldots, P_k$ yields a flow at Nash equilibrium for (B^k, k, ℓ^k) demonstrating that $L(B^k, k, \ell^k) = k+1$ (see Figure 5(a) for an illustration when $k = 3$). On the other hand, if H is the subgraph obtained from B^k by deleting all edges of the form (v_i, w_i), routing $\frac{k}{k+1}$ units of flow on each of $Q_1, \ldots, Q_{k+1}$ yields a flow at Nash equilibrium for (H, k, ℓ^k) showing that $L(H, k, \ell^k) = 1$ (see Figure 5(b)). Thus, $L(G)/L(H) = k+1 = n/2$, completing the proof. ■

To the best of our knowledge, the family of networks $\{(B^k, k, \ell^k)\}$ gives the first demonstration that the severity of Braess's Paradox can increase with the network size.

4.3 Proof of Hardness

We begin with an informal description of the reduction. The idea is to start with a Braess graph and replace the edges of the form (v_i, w_i) with a collection of parallel edges representing an instance $\mathcal{I}$ of the NP-hard problem PARTITION [13, SP12].[4] We will endow these edges with latency functions that simulate "capacities", with an edge representing an integer a_j of $\mathcal{I}$ receiving capacity a_j. Roughly speaking, if too many edges are removed from the network, there will be insufficient remaining capacity to send flow cheaply; if too few edges are removed, the excess of capacity results in a Nash equilibrium similar to that of Figure 5(a); and if $\mathcal{I}$ is a "yes" instance of PARTITION and an appropriate collection of edges is removed, then the remaining network admits a Nash equilibrium similar to that of Figure 5(b).

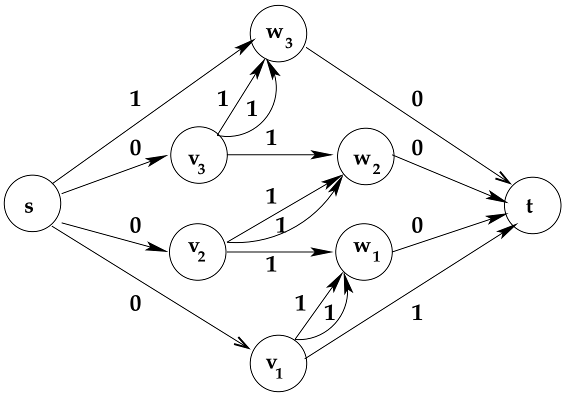
Theorem 4.3 *For $\epsilon > 0$, there is no $(\lfloor n/2 \rfloor - \epsilon)$-approximation algorithm for* GENERAL LATENCY NETWORK DESIGN *unless $P = NP$.*

Proof: We prove that for any fixed $n \geq 2$, there is no $(\lfloor \frac{n}{2} \rfloor - \epsilon)$-approximation algorithm for GENERAL LATENCY NETWORK DESIGN restricted to (multi)graphs with n vertices. (We can also restrict our instances to be simple networks and derive a nearly optimal inapproximability result — see Remark 1 below.) As in the proof of Proposition 4.2, we may assume that n is even and at least four. Write $n = 2k + 2$ for $k \in \mathcal{N}$. We will show that an $(\frac{n}{2} - \epsilon)$-approximation algorithm for graphs with n vertices enables us to differentiate between "yes" and "no" instances of PARTITION in polynomial time; for a nearly optimal inapproximability result derived from a strongly NP-complete problem, see Remark 2 below.

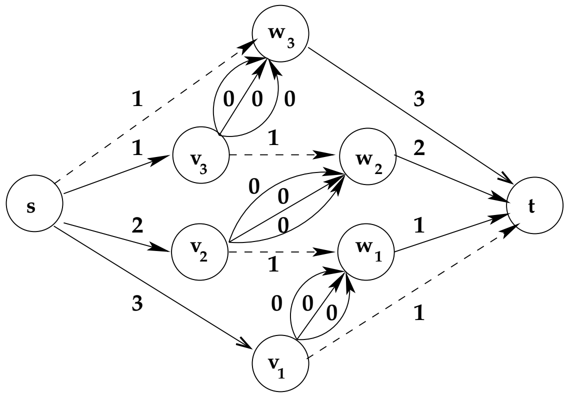
Consider an instance $\mathcal{I} = \{a_j\}_{j=1}^{p}$ of PARTITION, with each a_j a positive integer. We may assume that each a_j is even (scaling if necessary). Put $A = \sum_{j=1}^{p} a_j$; the traffic rate of interest to us is $r = k\frac{A}{2} + k + 1$. Obtain a graph G from the kth Braess graph B^k by replacing each edge of the form (v_i, w_i) by p parallel edges, and denote these by $e_i^1, e_i^2, \ldots, e_i^p$.

We now specify the edge latency functions ℓ, which are more complicated than in the previous subsection. We require a sufficiently small constant δ ($1/A(p + k)$ is small enough) and a sufficiently large constant M ($n/2$ is large enough). In what follows, the constant M should be interpreted as a substitute for $+\infty$, and is used to penalize a flow for violating an edge capacity constraint. We require the constant δ to transform step functions (the type of function that would be most convenient for our argument) into continuous functions (which are allowable in our model); δ provides a small "window" in which to "smooth out" the discontinuities of a step function. For each edge e of the form (v_i, w_{i-1}), (s, w_k), or (v_1, t), define $\ell_e(x) = 1$ for $x \leq 1$ and $\ell_e(x) = M$ for $x \geq 1 + \delta$ (ℓ_e may be defined arbitrarily on $(1, 1 + \delta)$, subject to the usual continuity and monotonicity restrictions). We say that these edges have *capacity* 1. For an edge e of the form (w_i, t) or (s, v_{k-i+1}) (where $i \in \{1, \ldots, k\}$), define $\ell_e(x) = 0$ for $x \leq \frac{1}{2}A + 1$, $\ell_e(x) = i$ when $x = \frac{1}{2}A + \frac{k+1}{k}$, and $\ell_e(x) = M$ for

[4]In an instance of PARTITION, we are given p positive integers $\{a_1, a_2, \ldots, a_p\}$ and seek a subset $S \subseteq \{1, 2, \ldots, p\}$ such that $\sum_{j \in S} a_j = \frac{1}{2} \sum_{j=1}^{p} a_j$.



(a) A good Nash flow corresponding to a "yes" instance of PARTITION, with $m = 2$



(b) A bad Nash flow in a network with excess capacity

Figure 6: Proof of Theorem 4.3. Solid edges carry flow in the flow at Nash equilibrium, dashed edges do not. Edge latencies are with respect to flows at Nash equilibrium.

$x \geq \frac{1}{2}A + \frac{k+1}{k} + \delta$; these edges have capacity $\frac{1}{2}A + \frac{k+1}{k}$. Finally, for an edge e of the form e_i^j, define $\ell_e(x) = 0$ for $x \leq a_j - \delta$, $\ell_e(a_j) = 1$, and $\ell_e(x) = M$ for $x \geq a_j + \delta$; thus e_i^j has capacity a_j.

Analogous to the proof of Theorem 3.3, it suffices to prove the following two statements: (i) if $\mathcal{I}$ is a "yes" instance, then G admits a subgraph H with $L(H, r, \ell) = 1$; and (ii) if $\mathcal{I}$ is a "no" instance, then $L(H, r, \ell) \geq n/2$ for every subgraph H of G.

To prove (i), suppose that $\mathcal{I}$ admits a partition, and reindex the a_j's so that $\sum_{j=1}^{m} a_j = A/2$ for some $m \in \{1, 2, \ldots, p-1\}$. Obtain H from G by deleting all edges of the form e_i^j for $j > m$; thus, for each $i = 1, \ldots, k$, the remaining edges of the form e_i^j have total capacity $A/2$. Define the paths $Q_1, \ldots, Q_{k+1}$ as in the proof of Proposition 4.2: for $i = 2, \ldots, k$, Q_i denotes the path $s \to v_i \to w_{i-1} \to t$, Q_1 is the path $s \to v_1 \to t$, and Q_{k+1} is the path $s \to w_k \to t$. Define a feasible flow f as follows: for each $i = 1, \ldots, k$ and $j = 1, \ldots, m$, route a_j units of flow on the unique path containing edge e_i^j, and route 1 unit of flow on the path Q_i for $i = 1, 2, \ldots, k+1$. The flow f is at Nash equilibrium for (H, r, ℓ) and proves that $L(H, r, \ell) = 1$ (see Figure 6(a)).

In proving (ii), we first consider only subgraphs H that contains all edges not of the form e_i^j (i.e., H may be obtained from G by deleting only some of the parallel edges); as we will see, this case captures all of the difficulties of the proof. There are two subcases to consider:

Case 1: Suppose for each $i = 1, \ldots, k$, the total capacity A_i of edges of the form e_i^j in H is at least $A/2$. Since $\mathcal{I}$ is a "no" instance and each a_j is even, $A_i \geq A/2 + 2$ for each i. Then, define a flow f in G as follows: for each $i = 1, \ldots, k$ and $j = 1, \ldots, p$ such that e_i^j is present in H, route $\frac{a_j}{A_i}(\frac{A}{2} + \frac{k+1}{k})$ units of flow along the unique s-t path containing e_i^j. The flow f is at Nash equilibrium and proves that $L(H) = n/2$ (see Figure 6(b)).

Case 2: Suppose for some $i \in \{1, \ldots, k\}$, the total capacity A_i of edges of the form e_i^j in H is less than $A/2$ (and thus is at most $A/2 - 2$). Here, we will exploit the fact that all edges of the network are (essentially) capacitated to prove that a flow at Nash equilibrium must have large cost. Call an edge e *oversaturated* by a flow f if f_e exceeds the capacity of e by at least δ (and thus $\ell_e(f_e) = M \geq n/2$). A key observation is that if f is at Nash equilibrium for (H, r, ℓ) and oversaturates some edge, then $L(H, r, \ell) \geq n/2$. Now, since the total capacity of edges out of v_i is at most $A/2 - 1$ (recall (v_i, w_{i-1}) has capacity 1), any flow that places at least $\frac{A}{2} - 1 + p\delta$ units of flow on (s, v_i) will oversaturate some edge out of v_i. On the other hand, the total capacity of edges incident to s is $k\frac{A}{2} + k + 2 = r + 1$, so any feasible flow must either place at least $\frac{A}{2} - 1 + p\delta$ units of flow on (s, v_i) or oversaturate some other edge out of s (for δ sufficiently small). We conclude that any flow feasible for (H, r, ℓ) oversaturates at least one edge, and hence $L(H) \geq n/2$.

Finally, suppose H fails to contain an edge that is not of the form e_i^j. If for some $i \in \{1, 2, \ldots, k\}$, the total capacity of edges of the form e_i^j is at most $A/2$, then the argument of Case 2 still applies to show that $L(H) \geq n/2$ (the previous argument merely required that any feasible flow oversaturates some edge, and this fact remains valid if we remove further edges). Also, if H fails to contain an arc of the form (s, v_i) or (w_i, t), then simple capacity considerations show that any feasible flow in H oversaturates some edge incident to s or t, respectively. If H contains all edges of the form (s, v_i) and (w_i, t) and the total capacity of edges of the form e_i^j in H is at least $A/2$ for each i, then the argument of Case 1 applies (by hypothesis, all edges used by the Nash flow in that case are present in H), showing that $L(H) = n/2$. This exhausts all possible cases, and the proof is complete. ■

The matching upper and lower bounds of Theorems 4.1 and 4.3 have strong negative consequences for the problem of detecting Braess's Paradox, as in the linear latency function setting (see Corollary 3.4). Defining an instance (G, r, ℓ) with general latency functions and n vertices to be *paradox-free* if $L(H, r, \ell) \geq L(G, r, \ell)$ for all subgraphs H of G and *paradox-ridden* if for some subgraph H in G, $L(H, r, \ell) = (\lfloor n/2 \rfloor)^{-1} L(G, r, \ell)$, we obtain the following corollary.

Corollary 4.4 *Given an instance (G, r, ℓ) with general latency functions that is either paradox-free or paradox-ridden, it is NP-complete to decide whether or not (G, r, ℓ) is paradox-ridden.*

Remark 1: The reduction of Theorem 4.3 also shows that, for any constant $\epsilon > 0$, there is no $O(n^{1-\epsilon})$-approximation algorithm for General Latency Network Design restricted to simple graphs (unless $P = NP$). To see why, choose a positive integer k satisfying $k > \frac{1}{\epsilon}$, and for a Partition instance $\mathcal{I}$ with p items, mimic the previous reduction beginning with the Braess graph B^{p^k} on $2p^k + 2$ vertices. Subdividing all parallel edges in the resulting multigraph yields a simple graph G (whose size is polynomial in that of $\mathcal{I}$) with $n = p^{k+1} + 2p^k + 2$ vertices. Defining r and ℓ as in the proof of Theorem 4.3, G has a subgraph H satisfying $L(H, r, \ell) = 1$ if $\mathcal{I}$ is a "yes" instance while $L(H, r, \ell) \geq p^k + 1$ for every subgraph H if $\mathcal{I}$ is a "no" instance. Thus, no $O(n^{(k-1)/k})$-approximation algorithm exists for General Latency Network Design restricted to simple graphs, unless $P = NP$.

Remark 2: By similar arguments, the non-existence of an $O(n^{1-\epsilon})$-approximation algorithm for network design on simple graphs (assuming $P \neq NP$) can be derived from the forthcoming constructions in the proofs of Theorems 5.5 and 6.4 (which give inapproximability results for instances with polynomial latency functions and with more general types of well-behaved latency functions, respectively). An advantage of the reductions in these two proofs is that they make use of the 2DDP problem of Section 3 (which is strongly NP-complete [13]) rather than PARTITION (which is not). On the other hand, the reductions are more complicated than that of the previous proof.

5 Polynomials of Bounded Degree: An Approximability Threshold of $\Theta(\frac{k}{\log k})$

In this section and the next, we aim to show that the strong hardness results of Sections 3 and 4 extend beyond the particular classes of linear and general latency functions, and seem intrinsic to the problem of designing networks for selfish users. This section considers a natural extension of the linear latency setting, where all latency functions are polynomials of bounded degree. The next section generalizes our results still further.

As in Section 3, we begin by observing that previous work bounding the worst-case inefficiency of flows at Nash equilibrium yields an upper bound on the performance guarantee of the trivial algorithm. The following result was recently proved by the author [31] by generalizing the techniques of Roughgarden and Tardos [33].

Proposition 5.1 ([31]) *Suppose $k \in \mathcal{N}$ and (G, r, ℓ) is an instance where each latency function is of the form $\ell(x) = a_k x^k + a_{k-1}x^{k-1} + \cdots + a_0$, with $a_i \geq 0$ for each i. If f^* is a feasible flow and f is a flow at Nash equilibrium for (G, r, ℓ), then $C(f) \leq (1 - k \cdot (k+1)^{-(k+1)/k})^{-1} \cdot C(f^*)$.*

We will say that such an instance has *polynomial latency functions of degree k* (with the understanding that all coefficients are nonnegative) and will call the corresponding network design problem POLYNOMIAL(k) LATENCY NETWORK DESIGN. For clarity, we will work with the following weaker form of Proposition 5.1.

Corollary 5.2 *There is a constant $c_1 > 0$ so that the following statement holds: if $k \geq 2$ and (G, r, ℓ) is an instance with polynomial latency functions of degree k for which f^* is feasible and f is a flow at Nash equilibrium, then $C(f) \leq c_1 \frac{k}{\ln k} \cdot C(f^*)$.*

As with linear latency functions (see Corollary 3.2), we immediately obtain an upper bound on the performance guarantee of the trivial algorithm.

Corollary 5.3 *There is a constant $c_1 > 0$ so that, for any $k \geq 2$, the trivial algorithm is a $c_1 \frac{k}{\ln k}$-approximation algorithm for* POLYNOMIAL(k) LATENCY NETWORK DESIGN.

We next work toward a proof of a matching hardness result. As in Section 4, we first give a family of networks (one network for each value of $k \geq 2$) on which the trivial algorithm performs poorly, and then describe how to obtain a general inapproximability result.

Proposition 5.4 *There is a constant $c_2 > 0$ so that, for any $k \geq 2$, the trivial algorithm has a (worst-case) performance guarantee of at least $c_2 \frac{k}{\ln k}$ for* Polynomial(k) Latency Network Design.

Proof: We will again make use of the Braess graphs of Subsection 4.2. In Section 4, we exploited the fact that general latency functions can be arbitrarily steep to construct a bad example for the trivial algorithm; here, we adapt the previous argument as best we can, given that only low-degree polynomials are available to us.

For a fixed integer $k \geq 2$, define a set of latency functions ℓ^p for the edges of B^p as follows (where p is a parameter, depending on k, to be chosen later): for each edge of the form $e = (v_i, w_i)$, put $\ell^p_e(x) = 0$; for an edge e of the form (v_i, w_{i-1}), (s, w_p), or (v_1, t), put $\ell^p_e(x) = 1$; for an edge e of the form (w_i, t) or (s, v_{p-i+1}) put $\ell^p_e(x) = ix^k$. Next, consider the instance (B^p, p, ℓ^p) and define paths $P_1, \ldots, P_p$ and $Q_1, \ldots, Q_{p+1}$ as in Proposition 4.2. On one hand, routing one unit of flow on each of $P_1, \ldots, P_p$ yields a flow at Nash equilibrium for (B^p, p, ℓ^p) showing that $L(B^p, p, \ell^p) = p + 1$ (as in Figure 5(a)). On the other hand, if H is the subgraph obtained from B^p by deleting all edges of the form (v_i, w_i), routing $\frac{p}{p+1}$ units of flow on each of $Q_1, \ldots, Q_{p+1}$ yields a flow at Nash equilibrium for (H, p, ℓ^p) showing that $L(H, p, \ell^p) = 1 + p(\frac{p}{p+1})^k$ (cf., Figure 5(b)). Thus,

$$L(H, p, \ell^p) = 1 + p(\frac{p}{p+1})^k \leq 1 + pe^{-k/(p+1)}.$$

Putting $p = \lfloor \frac{k}{2 \ln k} \rfloor - 1$ we obtain $L(H, p, \ell^p) \leq 2$ and $L(B^p, p, \ell^p) = \lfloor \frac{k}{2 \ln k} \rfloor$. Since $k \geq 2$ was arbitrary, the proof is complete. ■

Remark: In the proof of Proposition 5.4, we have avoided optimizing constants for the sake of readability. We will make this tradeoff repeatedly in the rest of the paper.

Finally, we extend our lower bound on the performance guarantee of the trivial algorithm to an inapproximability result. This task is more difficult than in Section 4; a crucial part of the hardness proof of that section leveraged the fact that general latency functions can model edge capacities. This is not entirely possible with low-degree polynomials, and we are forced instead to adapt the arguments of Section 3 to larger Braess graphs; in particular, our reduction is from the 2 Directed Disjoint Paths problem rather than from Partition. In essence, restricting the allowable class of latency functions forces us to encode the intractability of an NP-hard problem into the network topology of a network design instance rather than into the edge latency functions.

Theorem 5.5 *There is a constant $c_3 > 0$ so that the following statement holds: if $k \geq 2$ and $\epsilon > 0$, then no $(c_3 \frac{k}{\ln k} - \epsilon)$-approximation algorithm for* Polynomial(k) Latency Network Design *exists, unless $P = NP$.*

Proof: Fix an integer $k \geq 2$ and put $p = \lfloor \frac{k}{16 \ln k} \rfloor - 1$. For any $\epsilon > 0$, we will show that a $(\frac{p}{5} - \epsilon)$-approximation algorithm for Polynomial(k) Latency Network Design enables us to differentiate between "yes" and "no" instances of the 2 Directed Disjoint Paths (2DDP) problem (for a definition, see the proof of Theorem 3.3) in polynomial-time.

Consider an instance $\mathcal{I} = \{G, s_1, s_2, t_1, t_2\}$ of 2DDP; we construct an instance of Polynomial(k) Latency Network Design (G', p, ℓ) as follows (illustrated in Figure 7). To define the graph G', we begin with p copies of G; call them $G_1, \dots, G_p$ and denote the copy of s_i (t_i) in G_j by s_i^j (t_i^j). Next, add auxiliary vertices s, t, $v_1, \dots, v_{p-1}$, and $w_1, \dots, w_{p-1}$. The edge set of G' is as follows:

- each G_i inherits the edge set of G
- for $i = 1, \dots, p-1$, we include edges from s to v_i, from v_i to s_2^i and s_1^{i+1}, from t_2^i and t_1^{i+1} to w_i, and from w_i to t
- we include edges (s, s_1^1), (s, s_2^p), (t_1^1, t), (t_2^p, t).

We define latency functions on the edges of G' as follows:

(A) for edges of the form (v_i, s_2^i) or (t_1^{i+1}, w_i), put $\ell(x) = 1$

(B) for edges (s, s_1^1) and (t_2^p, t), put $\ell(x) = 2 + (1 + \frac{1}{p})^k x^k$

(C) for (s, s_2^p) and (t_1^1, t), put $\ell(x) = 1 + p(\frac{4(p+1)}{4p+1})^k x^k$

(D) for $i = 1, \dots, p-1$ and edges (s, v_i) and (w_{p-i}, t), put $\ell(x) = i(\frac{4(p+1)}{4p+1})^k x^k$

(E) for edges of the form (v_i, s_1^{i+1}) or (t_2^i, w_i), put $\ell(x) = 2 + (2 + \frac{2}{p})^k x^k$

(F) for edges in $G_1, \dots, G_p$, put $\ell(x) = 0$.

We will call edges of the form (v_i, s_2^i) or (t_1^{i+1}, w_i) *type A edges*, and so forth.

Next, we claim that if $\mathcal{I}$ is a "yes" instance of 2DDP, then there is a subgraph H of G' satisfying $L(H, p, \ell) \le 5$. To see why, let P_1^* and P_2^* denote vertex-disjoint s_1-t_1 and s_2-t_2 paths in G. Deleting all edges in G' that lie in some copy G_i of G but not on (the corresponding copy of) either P_1^* or P_2^*, we obtain a subgraph H of G' that is the union of $2p$ distinct s-t paths. Routing $\frac{p}{p+1}$ units of flow on the path containing s_1^1 and t_1^1 and on the path containing s_2^p and t_2^p, and $\frac{p}{2(p+1)}$ units of flow on each of the other $2p-2$ paths, we obtain a flow at Nash equilibrium for (H, p, ℓ). This flow proves that

$$L(H, p, \ell) = 4 + p\left(\frac{4(p+1)}{4p+1}\frac{p}{p+1}\right)^k = 4 + p\left(1 - \frac{1}{4p+1}\right)^k \le 4 + pe^{-k/(4p+1)} \le 5,$$

with the picture of this Nash flow somewhat analogous to Figure 5(b).

Finally, we show that if $\mathcal{I}$ is a "no" instance of 2DDP, then $L(H, p, \ell) \ge p$ for all subgraphs H of G'. We will prove this in two steps. First, we will show that unless H contains most of the edges in G', "capacity considerations" (similar to those used in the proof of Theorem 4.3) imply that $L(H)$ is large. Second, we show that if H contains most of the edges in G', then the flow at Nash equilibrium in H is similar to the bad Nash flow of Proposition 5.4, again showing $L(H)$ to be large.

Fix a subgraph H of G' containing an s-t path, and let f be an acyclic Nash flow in (H, p, ℓ) (see Lemma 2.6). We claim that if some type A or C edge of G' does not carry flow

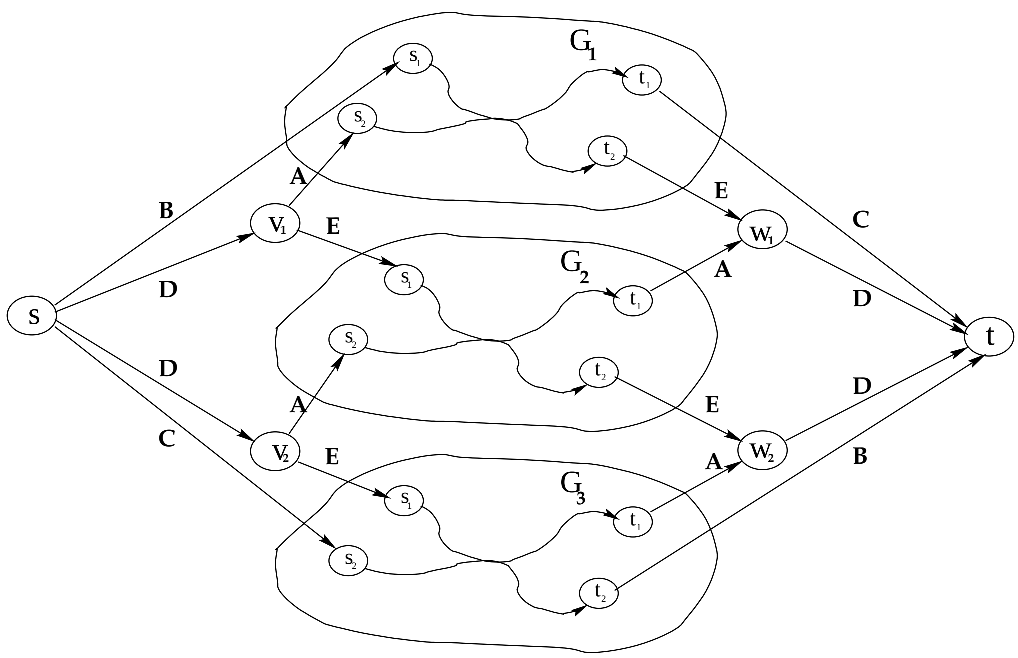


Figure 7: Proof of Theorem 5.5. Construction of (G', p, ℓ) when $p = 3$. Edges are labeled with their edge type.

in f (in particular, if some such edge is not in H), then $L(H) \geq p$. We will prove the claim for an edge of the form (v_i, s_2^i); the argument for an edge of the form (t_1^{i+1}, w_i) is symmetric, and the argument for type C edges is similar (and easier).

To prove this claim, we first observe that many edges of H are essentially capacitated, in the following sense. We assert that any of the following events forces $L(H) \geq k \geq p$ (using that $L(H) \geq \ell_e(f_e)$ for any edge e with $f_e > 0$):

(1) $f_e \geq \frac{8p+1}{8(p+1)}$ for a type B edge e

(2) $f_e \geq \frac{2p+1}{2(p+1)}$ for an edge e of type C or D

(3) $f_e \geq \frac{4p+1}{8(p+1)}$ for a type E edge e.

For example, we can derive

$$\left(\frac{4(p+1)}{4p+1}\frac{2p+1}{2(p+1)}\right)^k = \left(1 + \frac{1}{4p+1}\right)^k > \left[e^{1/(8p+2)}\right]^k = e^{k/(8p+2)} \geq k,$$

proving (2). The calculations for (1) and (3) are similar, so we omit them.

Now assume that edge (v_j, s_2^j) does not carry any flow in f. Then, either event (3) occurs (with edge (v_j, s_1^{j+1})) or else edge (s, v_j) carries at most $\frac{4p+1}{8(p+1)}$ units of flow; assume the latter. We claim that in this case, event (1) or event (2) must occur with some edge incident to s. For if not, edges incident to s carry at most

$$(p-1)\frac{2p+1}{2(p+1)} + \frac{4p+1}{8(p+1)} + \frac{8p+1}{8(p+1)} = \frac{8p^2+8p-2}{8(p+1)} < p$$

units of flow, contradicting that f is an s-t flow carrying p units of flow. We conclude that if edge (v_j, s_2^j) does not carry flow in f, then some event of the form (1), (2), or (3) occurs, proving that $L(H) \geq p$.

It remains to consider subgraphs H of G' in which all edges of type A or C carry flow in the Nash flow f of (H, p, ℓ), and to make use of our hypothesis that $\mathcal{I}$ is a "no" instance of 2DDP. The presence of these edges in H (all of which lie on s-t paths in H, since they carry flow in the acyclic flow f), together with the assumption that $\mathcal{I}$ is a "no" instance, imply that for each $i = 1, 2, \ldots, p$ there is an s-t path P_i in H containing the vertices s_2^i and t_1^i (cf., the proof of Theorem 3.3). Letting $r = \frac{p(4p+1)}{4(p+1)} < p$, the following flow is then at Nash equilibrium for (H, r, ℓ): for $i = 1, 2, \ldots, p$ route $\frac{4p+1}{4(p+1)}$ units of flow on P_i. This flow shows that $L(H, r, \ell) = p + 3$ (this Nash flow is essentially the same as the bad Nash flow of Proposition 5.4); since $L(H, \cdot, \ell)$ is an increasing function of the traffic rate (with H, ℓ fixed) [15], we have $L(H, p, \ell) \geq p + 3$.

We have shown that if $\mathcal{I}$ is a "no" instance of 2DDP, then $L(H, r, \ell) \geq p$ for all subgraphs H of G', and the proof is complete. ■

6 Further Extensions

In this section we accumulate further evidence that the intractability of designing networks for selfish users is not sensitive to the class of allowable latency functions. Before introducing the class of latency functions that we will consider, we note that identifying large classes of latency functions that behave better than general ones is a non-trivial task. For example, one natural idea is to require that network latency functions possess a bounded first derivative (or bounded first k derivatives, for some fixed integer k) on the domain of interest ($[0, r]$, where r is the traffic rate). However, any instance (G, r, ℓ) with latency functions of class C^k (i.e., latency functions that are k times continuously differentiable) can be "scaled down" to an instance $(G, r, \frac{1}{M}\ell)$ in which the first k derivatives of all edge latency functions are as small as desired (by taking M sufficiently large); moreover, the network design problem on the scaled instance is equivalent (from the viewpoint of approximation) to the problem on the original instance. Thus, the results of Section 4 apply to this setting, showing that the trivial algorithm is an $\lfloor n/2 \rfloor$-approximation algorithm and that no better approximation guarantee is possible in polynomial time (unless $P = NP$).

We thus require some "scale-invariant" parameter ensuring that a latency function will behave better than an arbitrary continuous, nondecreasing one. Toward this end, consider an instance (G, r, ℓ). We will introduce a quantity that gives a non-trivial upper bound on the cost of a Nash flow in (G, r, ℓ), relative to that of any other feasible flow (as in Corollaries 3.2 and 5.3, this gives an upper bound on the performance of the trivial algorithm). For an edge $e \in E$ and $x \in [0, r]$, define the quantity $\Gamma_e(x)$ by

$$\Gamma_e(x) = \begin{cases} \frac{x \cdot \ell_e(x)}{\int_0^x \ell_e(y)\, dy} & \text{if } \ell_e(x) > 0 \text{ and } x > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Since ℓ_e is nondecreasing, $\Gamma_e(x) \geq 1$ for all $e \in E$ and $x \in [0, r]$. Now define $\Gamma(G, r, \ell)$ by

$$\Gamma(G, r, \ell) = \max_{e \in E} \sup_{x \in [0,r]} \Gamma_e(x).$$

For example, the Γ-value of an instance with polynomial latency functions of degree k is at most $k + 1$.

These bizarre definitions are justified by the next result, due to Roughgarden and Tardos [33], which bounds the inefficiency of a flow at Nash equilibrium in instance (G, r, ℓ) by $\Gamma(G, r, \ell)$. Roughly, the following proposition is proved by using that a Nash flow optimizes a certain (not particularly natural) objective function over the set of all feasible flows [1], and proving that the value of this objective function and the cost $C(\cdot)$ of a flow differ by at most a $\Gamma(G, r, \ell)$ factor. The reader is referred to [33] for details.

Proposition 6.1 ([33]) *Suppose (G, r, ℓ) is an instance for which f^* is a feasible flow and f is a flow at Nash equilibrium. Then $C(f) \leq \Gamma(G, r, \ell) \cdot C(f^*)$.*

While Proposition 6.1 is not as strong as Propositions 3.1 and 5.1 in the special case of polynomials, it yields a non-trivial upper bound on the worst-case inefficiency of a Nash flow for a broad spectrum of latency functions. As usual, we obtain as a corollary an upper bound on the performance guarantee of the trivial algorithm.

Corollary 6.2 *The trivial algorithm is a γ-approximation algorithm for network design instances (G, r, ℓ) satisfying $\Gamma(G, r, \ell) \leq \gamma$.*

It is natural to ask if the upper bound of Corollary 6.2 is best possible. By following the approach of Proposition 5.4, replacing the function x^k by the function that is equal to $\frac{1}{\gamma}$ on $[0, \frac{\gamma}{\gamma+1}]$ and linear on $[\frac{\gamma}{\gamma+1}, 1]$ subject to $g(\frac{\gamma}{\gamma+1}) = \frac{1}{\gamma}$ and $g(1) = 1$, and setting the parameter p (which controls the size of underlying Braess graph) to be $\lfloor \gamma \rfloor$, we answer in the affirmative.

Proposition 6.3 *The trivial algorithm has a (worst-case) performance guarantee of at least $\gamma/2$ for network design restricted to instances (G, r, ℓ) satisfying $\Gamma(G, r, \ell) \leq \gamma$.*

Moreover, we can extend the lower bound on the performance of the trivial algorithm to an inapproximability result. The proof of this is quite similar to the proof of Theorem 5.5 (and is a bit easier, due to the greater flexibility available for defining latency functions), and is therefore omitted.

Theorem 6.4 *There is a constant $c > 0$ so that the following statement holds: if $\gamma \geq 1$ and $\epsilon > 0$, then there is no $(c \cdot \gamma - \epsilon)$-approximation algorithm for network design restricted to instances (G, r, ℓ) satisfying $\Gamma(G, r, \ell) \leq \gamma$, unless $P = NP$.*

Acknowledgements

We thank Leonard Schulman for introducing us to Braess's Paradox and the Linear Latency Network Design problem, Éva Tardos for helpful discussions and comments on an earlier version of this paper, and Jon Kleinberg for a useful conversation.

References

[1] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, 1956.

[2] D. E. Boyce and J. L. Soberanes. Solutions to the optimal network design problem with shipments related to transportation cost. *Transportation Research*, 13B(1):65–80, 1979.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. Network Working Group Request for Comments 2309, April 1998.

[4] D. Braess. Über ein paradoxon der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.

[5] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transactions on Networking*, 1(6):614–627, 1993.

[6] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proceedings of the 13th Annual Symposium on Discrete Algorithms*, pages 413–420, 2002.

[7] S. C. Dafermos and F. T. Sparrow. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards, Series B*, 73B(2):91–118, 1969.

[8] R. Dionne and M. Florian. Exact and approximate algorithms for optimal network design. *Networks*, 9(1):37–59, 1979.

[9] P. Dubey. Inefficiency of Nash equilibria. *Mathematics of Operations Research*, 11(1):1–8, 1986.

[10] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 218–227, 2000.

[11] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[12] M. Frank. The Braess Paradox. *Mathematical Programming*, 20:283–302, 1981.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[14] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 4, pages 144–191. PWS Publishing Company, 1997.

[15] M. A. Hall. Properties of the equilibrium state in transportation networks. *Transportation Science*, 12(3):208–216, 1978.

[16] H. H. Hoc. A computational approach to the selection of an optimal network. *Management Science*, 19(5):488–498, 1973.

[17] Y. A. Korilis, A. A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161–173, 1997.

[18] Y. A. Korilis, A. A. Lazar, and A. Orda. Capacity allocation under noncooperative routing. *IEEE Transactions on Automatic Control*, 42(3):309–325, 1997.

[19] Y. A. Korilis, A. A. Lazar, and A. Orda. Avoiding the Braess paradox in noncooperative networks. *Journal of Applied Probability*, 36(1):211–222, 1999.

[20] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413, 1999.

[21] L. J. LeBlanc. An algorithm for the discrete network design problem. *Transportation Research*, 9:183–199, 1975.

[22] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, 1984.

[23] M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 510–519, 2001.

[24] J. D. Murchland. Braess's paradox of traffic flow. *Transportation Research*, 4:391–394, 1970.

[25] N. Nisan. Algorithms for selfish agents: Mechanism design for distributed computation. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 1–15, 1999.

[26] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1/2):166–196, 2001. Preliminary version in *STOC '99*.

[27] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multi-user communication networks. *IEEE/ACM Transactions on Networking*, 1:510–521, 1993.

[28] G. Owen. *Game Theory*. Academic Press, 1995. Third Edition.

[29] C. Papadimitriou. Algorithms, games, and the Internet. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 749–753, 2001.

[30] T. M. Ridley. An investment policy to reduce the travel time in a transportation network. *Transportation Research*, 2(4):409–424, 1968.

[31] T. Roughgarden. The price of anarchy in networks with polynomial edge latency. Technical Report TR2001-1847, Cornell University, 2001.

[32] T. Roughgarden. Stackelberg scheduling strategies. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 104–113, 2001.

[33] T. Roughgarden and É. Tardos. How bad is selfish routing? In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 93–102, 2000. Full version to appear in *Journal of the ACM*.

[34] A. J. Scott. The optimal network problem: Some computational procedures. *Transportation Research*, 3(2):201–210, 1969.

[35] S. J. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3(6):819–831, 1995.

[36] R. Steinberg and W. I. Zangwill. The prevalence of Braess' paradox. *Transportation Science*, 17(3):301–318, 1983.

[37] R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM, 1983.

[38] J. G. Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institute of Civil Engineers, Pt. II*, volume 1, pages 325–378, 1952.