Designing Networks for Selfish Users is Hard

Tim Roughgarden*

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 nodes 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. Roughly, this result implies that the presence of harmful extra edges in a network (a phenomenon that can lead to extremely poor performance in large networks with general latency functions) is impossible to detect efficiently.*

*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, existing 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., [6, 15, 18, 25, 32, 34]) to model the behavior of network users by a *noncooperative game* and to study the resulting *Nash equilibria* (see Owen [26], 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 [8]), 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" [27]) [18, 21, 30, 32], and influencing the behavior of selfish agents via pricing policies [5], network switch protocols [34], routing a small portion of the traffic centrally [15, 31], or algorithmic mechanism design [9, 23, 24].

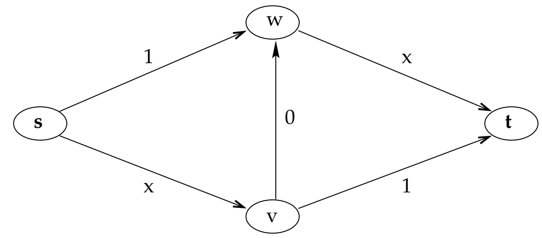
In this paper, we explore a different idea for ameliorating the degradation in network performance due to selfish routing: armed with the knowledge that our networks will be host to selfish users, how can we design them to minimize the inefficiency inherent in a user-defined equilibrium?

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 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).

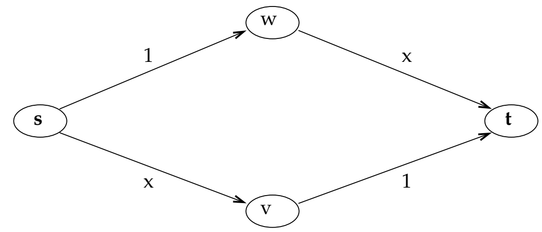
A natural measure for the performance of a network used by selfish agents is the common latency experienced by each user in a Nash equilibrium, as it navigates from s to t. It is a counterintuitive but well-known fact that removing edges from a network may *improve* its performance; this phenomenon is best illustrated by *Braess's Paradox*, as shown in Figure 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 arc (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, edge latency functions, 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 subgraph 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 [13]), 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 nodes 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.



(a) The Whole Graph



(b) The Optimal Subgraph

Figure 1. Braess's Paradox

Soon after Braess's Paradox was reported [4, 22], researchers attempted to solve variants of this network design problem (for example, one is alluded to in the work of Dafermos and Sparrow [6]), but scant progress has been made either computationally or theoretically. Indeed, early computational work either focused on very small networks [19] or admitted to ignoring congestion effects entirely, due to the difficulties involved [2, 7, 14, 28, 33]; in a 1984 survey, Magnanti and Wong describe the problem as "essentially unsolved" from a practical perspective [20, P.15]. On the theoretical side, there has been work classifying the network topologies and latency functions in which the deletion of a single edge can be helpful [11, 16, 35] and showing that certain edge deletion strategies cannot improve network performance [17], 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 nodes 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. Roughly, this result implies that the presence of harmful extra edges in a network (a phenomenon that can lead to extremely poor performance in large networks with general latency functions) is impossible to detect efficiently.

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).

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 $\mathcal{P}$. 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 $e \in E$ is given 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 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 [6, 32], 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 $\ell_{P_1}(f) \leq \ell_{P_2}(\tilde{f})$, 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 [37]).

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

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 flows at Nash equilibrium always exist and are essentially unique.

Lemma 2.3 ([1, 6, 32]) *If G is a network with continuous, nondecreasing latency functions ℓ and $r \geq 0$, then (G, r, ℓ) admits a feasible flow at Nash equilibrium. Moreover, if f, f' are feasible flows at Nash equilibrium, then $\ell_P(f) = \ell_P(f')$ for every s-t path P.*

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 (w.r.t.

[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.

f) of every s-t flow path of f. For a graph G in which there is no s-t path, we define $L(G, r, \ell) = +\infty$. 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 $L(H, r, \ell)$.

Our final preliminary result relates our objective function $L(H, r, \ell)$ to a second objective function that has been well-studied. 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*

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

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 [6, 11, 32], 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 [32]. 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 ([32]) *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 $L(G, r, \ell) \leq \frac{4}{3} L(H, r, \ell)$. ■

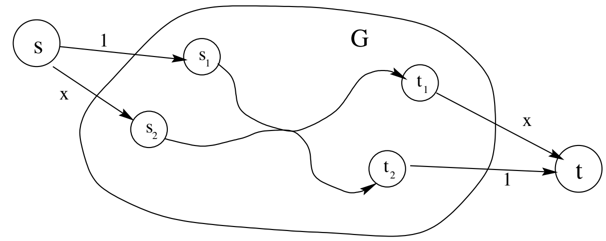


Figure 2. Proof of Theorem 3.3. In a "no" instance of 2DDP, existence of s_1-t_1 and s_2-t_2 paths implies the existence of an s_2-t_1 path.

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. [10]. 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 arcs $(s, s_1), (s, s_2), (t_1, t), (t_2, t)$ (see Figure 2). Denote the new network by $G' = (V', E')$ and endow the arcs of E' with linear latency functions ℓ as follows: all arcs of E are given the latency function $\ell(x) = 0$, arcs (s, s_2) and (t_1, t) are given the latency function $\ell(x) = x$, and arcs (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, 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_i^* be vertex-disjoint s_i-t_i paths in G $(i = 1, 2)$ 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 $L(H) = 2$. ■

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 non-decreasing functions. We begin by proving that the trivial algorithm achieves an approximation ratio of $\lfloor n/2 \rfloor$ (in contrast with other sections, this result does not trivially follow from previous work). We next introduce a new family of graphs generalizing the network of Braess's Paradox (Figure 1(a)), and then conclude 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. 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 (w.r.t. the total latency measure) as any other feasible flow for some "nice" (e.g., linear) function $f(\cdot)$, we would be done. Unfortunately, no such result can hold: a Nash flow may be *arbitrarily* more costly than other feasible flows, even when the network consists only of two nodes and two edges. To see why, we recapitulate an example from [32]: consider a network G with nodes 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 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 following result.

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.*

Prior to plunging into the proof, we give a slight extension of Lemma 2.2. While Lemma 2.2 implies that all s-t flow paths of a flow at Nash equilibrium have equal latency, the following lemma implies the same statement with s and t replaced by an arbitrary pair of vertices.

Lemma 4.2 *Let f be a flow at Nash equilibrium for (G, r, ℓ), and let P be a v-w path in G. Let $d(v), d(w)$ denote the lengths, with respect to edge lengths $\ell_e(f_e)$, of the shortest s-v and s-w paths in G, respectively. Then:*

(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: It suffices to prove the lemma when P is a single edge (for a general path, sum up the inequalities or equations 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 length 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 4.2 does *not* need to be a subpath of any flow path of f; put differently, the flow on different edges of P can be carried by distinct flow paths of f.

We are now prepared to prove the main result of this subsection.

Proof of Theorem 4.1. We may assume that n is odd (for n even, first subdivide some edge). Let f and f^* be flows at Nash equilibrium for (G, r, ℓ) and (H, r, ℓ), respectively, where H is a subgraph of G containing an s-t path. Put $L = L(G, r, \ell)$ and $L^* = L(H, r, \ell)$; we wish to prove that $L \leq \lfloor n/2 \rfloor \cdot L^*$.

We may assume that f is an acyclic flow (otherwise, we can remove the necessarily zero-latency flow cycles). Also, we will assume that every vertex is incident to a flow-carrying edge (this will be the worst case for our argument). For $v \in V(G)$, let $d(v)$ denote the length (w.r.t. edge lengths $\ell_e(f_e)$) of a shortest s-v path. Order the vertices $s = v_0, v_1, \ldots, v_{n-1} = t$ according to nondecreasing $d(v)$-value; if there is an arc $e = (v, w)$ with $f_e > 0$ and $\ell_e(f_e) = 0$ (so, by Lemma 4.2(b), $d(v) = d(w)$), break the tie by placing v before w in the ordering. Lemma 4.2(b) implies that this ordering is a topological one with respect to the flow f (i.e., whenever $f_e > 0$, e is a forward arc with respect to our ordering). Our proof approach will be to show, by induction on i, that $d(v_{2i}) \leq i \cdot L^*$ (the base case $i = 0$ is 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^* w.r.t. f^* (as every flow path of f^* has latency L^*) and hence latency at most L^* w.r.t. 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 states that every s-t cut consisting of a set of consecutive vertices (w.r.t. 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 [36]). Let $\delta^+(S)$ ($\delta^-(S)$) denote the edges with tail (head) inside S and head (tail) outside S. Since S is an s-t cut and f is an s-t flow of value r with no flow on arcs 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, $f_e \leq f_e^*$ 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 $j \leq 2(i-1)$. The previous claim immediately implies that k is well-defined with $k > 2(i-1)$ (consider the head of a light edge in $\delta^+(\{v_0, \ldots, v_{2(i-1)}\})$). In fact, we must have $k \geq 2i$: if $k = 2i - 1$ then all light arcs with tail in $\{v_0, \ldots, v_{2(i-1)}\}$ (and there must be one) have head v_{2i-1} and no light arc has tail v_{2i-1} (otherwise we would append such an arc to our maximal path), contradicting that $\delta^+(\{v_0, \ldots, v_{2i-1}\})$

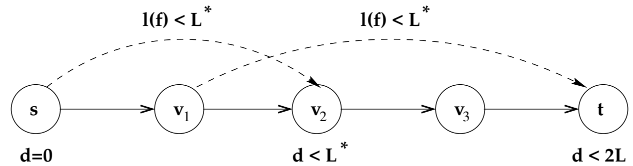


Figure 3. Proof of Theorem 4.1. If f is the flow sending one unit of flow on the four-hop path and f^* is the flow sending half a unit of flow on each of the other two paths, then the dashed edges are light.

must contain a light arc.

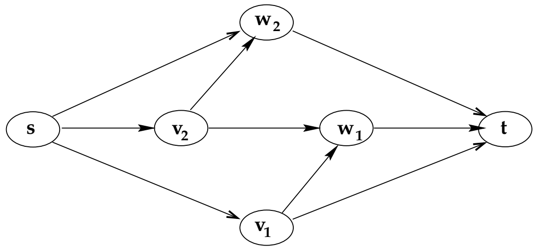
We have established the existence of a path P of light arcs from v_j to v_k with $j \leq 2(i-1)$ and $k \geq 2i$. Inductively, we have $d(v_j) \leq d(v_{2(i-1)}) \leq (i-1)L^*$; since $d(v_{2i}) \leq d(v_k)$, we can finish the inductive step and the proof by showing that $d(v_k) - d(v_j) \leq L^*$ (informally, $d(v_{2(i-1)})$ 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 4.2(b) to f^* in H yields $0 = d^*(s) \leq d^*(v_j) \leq d^*(v_k) \leq d^*(t) = L^*$. By Lemma 4.2(b), this implies that the length of P w.r.t. f^* is at most L^*; since all of these edges are light, it follows that the length of P w.r.t. f is at most L^*. Finally, an application Lemma 4.2(a) yields $d(v_k) - d(v_j) \leq L^*$, completing the inductive step and the proof. ■

4.2 The Braess Graphs

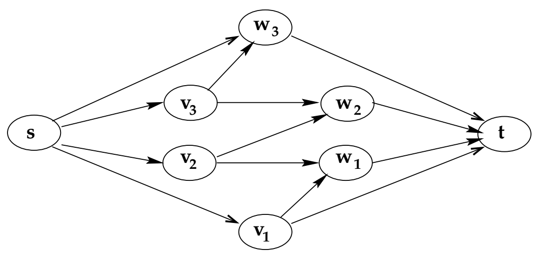
We seek to prove a lower bound on the approximability of network design (and in particular, on the performance of the trivial algorithm) that is linear in the number of nodes of the network. Toward this end, we will first construct an infinite family of networks on which the trivial algorithm performs poorly (i.e., networks in which the value of a flow at Nash equilibrium can be vastly improved by removing some arcs); 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 $V^k = \{s, v_1, \ldots, v_k, w_1, \ldots, w_k, t\}$ 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 discovered (Figure 1(a)).

We next define latency functions ℓ^k for the edges of B^k. For each edge of the form $e = (v_i, w_i)$, put $\ell_e^k(x) = 0$;

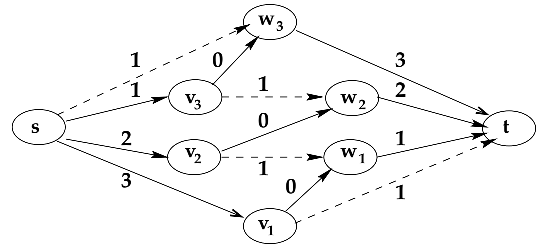


(a) B^2

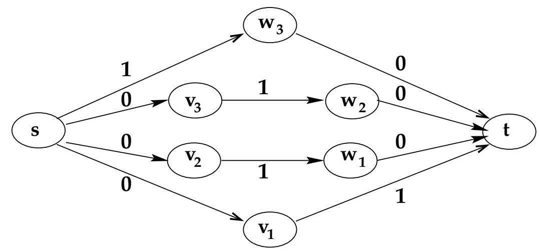


(b) B^3

Figure 4. The second and third Braess graphs



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



(b) Nash flow in the optimal subgraph

Figure 5. Proof of Lemma 4.3, 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.

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 $i \in \{1, 2, \dots, k\}$ 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.

Lemma 4.3 *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 subdivide an edge). Write $n = 2k+2$ for $k \in \mathcal{N}$ and consider the instance (B^k, k, ℓ^k). For $i = 1, \dots, k$, let P_i denote the path $s \to v_i \to w_i \to t$. For $i = 2, \dots, k$, 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 $P_1, \dots, P_k$ 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 $Q_1, \dots, Q_{k+1}$ 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. ■

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 arcs representing an instance $\mathcal{I}$ of the NP-hard problem PARTITION (see [12, SP12]).[2] 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

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

in a Nash equilibrium similar to that of Figure 5(a); and if $\mathcal{I}$ is a "yes" instance of PARTITION and an appropriate number of edges are removed, then the remaining network admits a Nash equilibrium similar to that of Figure 5(b).

Theorem 4.4 *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 nodes. (We can also restrict our instances to be simple networks and derive a nearly optimal inapproximability result — see the Remark below.) As in the proof of Lemma 4.3, 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 nodes enables us to differentiate between "yes" and "no" instances of PARTITION in polynomial time.

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 ($1/(p+k)$ 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 $x \geq \frac{1}{2}A + \frac{k+1}{k} + \delta$; 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 $\sum_{j=1}^{r} a_j = A/2$ for some $r \in \{1, 2, \ldots, p-1\}$. Obtain H from G by deleting all edges of the form e_i^j for $j > r$; 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 Lemma 4.3. Define a feasible flow f as follows: for $i = 1, \ldots, k$ and $j = 1, \ldots, r$, 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 $L(H, r, \ell) = 1$ (the picture is similar to Figure 5(b), with the additional "cross edges" of the form e_i^j all having latency 1).

In proving (ii), we focus on the case where H contains all of the edges not of the form e_i^j (i.e., H may be obtained from G by deleting only some of the parallel arcs), as 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 (analogous to Figure 5(a)), and proves that $L(H) = n/2$.

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 $L(H) \geq n/2$. ■

Remark: The previous reduction 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 $2p^k + 2$ nodes. Subdividing all parallel edges in the resulting multigraph yields a simple graph G (whose size is polynomial in that of $\mathcal{I}$) with $n = p^{k+1} + 2p^k + 2$ nodes satisfying $L(G, r, \ell) = p^k + 1$. Moreover, 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 $P = NP$.

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

In this section, 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. In this extended abstract, we consider only the extension of our hardness results to the setting where all latency functions are polynomials of bounded degree. Full proofs of these results and further extensions to additional classes of latency functions are described in the full version [29].

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 [30] by generalizing the techniques of Roughgarden and Tardos [32].

Proposition 5.1 ([30]) *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 this instance, 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 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.

Lemma 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.

The proof of Lemma 5.4 (omitted in this extended abstract) again makes 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.

Finally, we discuss how to extend our lower bound on the performance guarantee of the trivial algorithm to an inapproximability result. This task is somewhat 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.

The proof of following hardness result (which we omit) relies on arguments similar to those in the proofs of both Theorem 3.3 and Theorem 4.4.

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 $P = NP$.*

Acknowledgements

We thank Leonard Schulman for introducing us to Braess's Paradox and the LINEAR LATENCY NETWORK DESIGN problem, and Éva Tardos for helpful discussions and comments on an earlier version of this paper.

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] 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.

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

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

[9] 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.

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

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

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

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

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

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

[21] M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proceedings of the 323rd Annual ACM Symposium on the Theory of Computing*, pages 510–519, 2001.

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

[23] 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.

[24] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 129–140, 1999.

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

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

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

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

[29] T. Roughgarden. Designing networks for selfish users is hard. `http://www.cs.cornell.edu/timr`.

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

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

[32] 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 available at `http://www.cs.cornell.edu/timr`.

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

[34] 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.

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

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

[37] 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.