Dynamic Algorithms for Detecting Vulnerability to the Braess Paradox in Multi-Commodity Networks

Dario Fiorenza[1]

[1]Sapienza University Of Rome

Abstract

Braess paradox is a well known and largely studied phenomenon in traffic networks. Determining whether the paradox occurs has been shown to be NP-hard. A purely graph-theoretic characterization of this phenomenon, introduced by Roughgarden in 2006 under the name of *vulnerability*, provides an alternative viewpoint. The offline vulnerability problem can be solved in polynomial time, with the fastest known algorithm running in $O(m^2)$ time for single commodity and $O(km^2)$ for k-commodity networks. We summarize our two recent works, in which we developed dynamic amortized $O(m)$-time algorithms for detecting vulnerability under edge insertions and deletions, and discuss the techniques underlying their design. As a new contribution, we extend these results to k-commodity networks, obtaining amortized $O(km)$-time dynamic algorithms.

Keywords

Braess Paradox, Routing Games, Algorithmic Game Theory, Dynamic Algorithms, Graph Theory

1. Braess Paradox and Vulnerability

In the context of *game theory*, we focus on *routing games*. Selfish agents travel through a network, modeled as a graph with a source node s and a destination node t. Each edge is associated with a latency function, and every agent aims to minimize its own travel time. The latency experienced on an edge depends on its congestion, which in turn is determined by the routing choices of all agents. A flow is said to be at *Wardrop equilibrium* if no agent can decrease its latency by unilaterally switching to a different path [1, 2]. Throughout the paper, we assume agents are *non-atomic*, that is, they can be viewed as an infinitesimal fluid.

A minimal example, which lies at the core of many characterizations that follow in the paper, is the so-called *Wheatstone network* shown in Fig. 1(1). The edges $u \to v$, $s \to v$, and $u \to t$ are associated with the constant latency functions 0, 1, and 1, respectively; therefore, their latency is independent of the amount of traffic traversing them. By contrast, the latency on edges $s \to u$ and $v \to t$ is linear in the amount of flow using them.

Assume a traffic demand of value 1. In the resulting Wardrop equilibrium, all selfish agents choose the path $s \to u \to v \to t$, yielding a latency of $1 + 0 + 1 = 2$. On the other hand, the socially optimal flow routes half of the traffic along $s \to u \to t$ and the other half along $s \to v \to t$, resulting in a smaller latency of $3/2$. However, this flow is not a Wardrop equilibrium in the original network. Interestingly, the same latency of $3/2$ is attained at Wardrop equilibrium in the subnet of the Wheatstone network depicted in Fig. 1(2). This phenomenon is known as *Braess paradox* [1, 3]: removing edges (even very 'efficient' ones, like the 0-latency edge $u \to v$ in our example) from a network may decrease the latency at Wardrop equilibrium.

Braess paradox has been studied over the last decades. From an algorithmic perspective, deciding whether a given network suffers from Braess paradox was shown to be *NP*-hard in [4]. Motivated by this hardness result, the author proposed a different perspective, focusing on the underlying graph structure rather than on a specific instance, that is, without fixing the assignment of latency functions and traffic demands. In particular, he introduced the notion of *vulnerable* networks, namely networks that admit at least one instance in which Braess paradox occurs. This shifts the focus from detecting the

ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy
fiorenza@di.uniroma1.it (D. Fiorenza)
0000-0001-8216-9886 (D. Fiorenza)

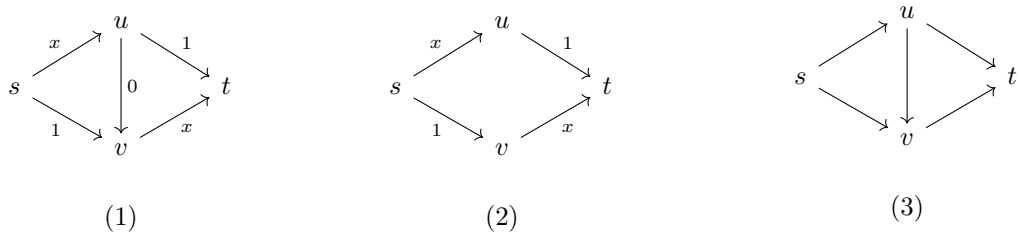


Figure 1: (1) The Wheatstone network; (2) Its optimal subnet; (3) The graph $\mathcal{W}$.

paradox in a fixed instance to characterizing and efficiently recognizing networks that are susceptible to it.

A characterization of vulnerable undirected multigraphs was presented in [5], where it was proved that an undirected graph is vulnerable if and only if it is not *series-parallel* [6]. Recall that a two-terminal (st) graph is series-parallel if it can be obtained recursively from a single edge by repeated series and parallel compositions of two-terminal graphs. A series composition identifies the sink of one graph with the source of the other ($t' = s''$), whereas a parallel composition identifies corresponding terminals ($s' = s''$ and $t' = t''$). This characterization of vulnerability applies only to graphs in which every node and edge belongs to at least one simple st-path. In [7], the same characterization was extended to directed multigraphs satisfying the same condition, referred to as *irredundancy* therein. Indeed, redundant edges (i.e., edges that do not belong to any simple st-path) may destroy the series-parallel structure of a directed graph, yet they do not affect its vulnerability, since they do not belong to any simple st-path; hence, there exists a Wardrop equilibrium that does not use them.

While testing (ir)redundancy and computing the *Maximal Irredundant Subnet* (*MIS*) can be done efficiently in undirected graphs [8], the situation is different for directed graphs. In [9], it was shown that deciding whether a given edge is redundant in a digraph is an *NP*-hard problem. Moreover, in [10] the authors prove that vulnerability coincides with containing an irredundant subgraph homeomorphic to the graph $\mathcal{W}$ of Fig. 1(3). In [9] they provide a $\mathcal{O}(nm^2)$ algorithm to state if a given graph is vulnerable or not.

The fastest known algorithms for the vulnerability problem run in $O(m^2)$ time for the single-commodity case and $O(km^2)$ time for the k-commodity case (i.e., networks with k (s,t) pairs) [11]. In this work, we summarize two results [12, 13] in which we developed incremental and decremental dynamic algorithms to solve the single-commodity problem; both algorithms have an amortized cost of $O(m)$. As a novel result in this work we show how those algorithms can be extended to solve the k-commodity problem in $O(km)$ amortized time.

2. The Dynamic Algorithms

A challenging algorithmic question is how to handle vulnerability in the context of dynamically evolving graphs [14]. Dynamic graph algorithms face problems like graph connectivity [15], the minimum spanning tree and many other graph problems [16]. In our context, edges may be inserted or deleted over time, potentially affecting whether the network is vulnerable to Braess paradox.

We first observe that, as an easy consequence of the characterizations given by [10] and [7], adding an edge may turn a non-vulnerable network into a vulnerable one, but a vulnerable network can never become non-vulnerable. Dually, removing an edge may turn a vulnerable network into a non-vulnerable one, but not vice versa. We exploit this monotonicity for both the incremental and the decremental algorithms. We believe that the difficulty of designing a linear-time amortized fully dynamic algorithm lies in the absence of this monotonicity.

2.1. The Incremental Algorithm

In this section we summarize [12]. When we consider a network that evolves through edge insertions, we are in the *incremental* setting. The first property we exploit is the monotonicity of vulnerability with

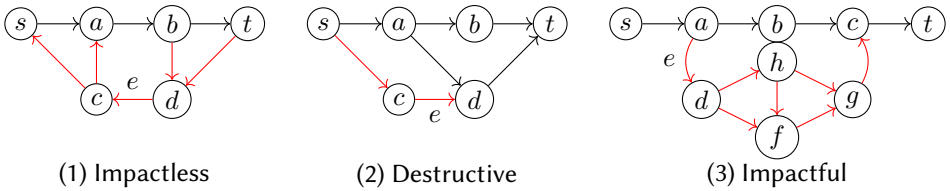


Figure 2: Types of edge insertions. The *MIS* is shown in black, and the $RCC(e)$ in red.

respect to edge insertion: if a network is vulnerable, then adding further edges preserves vulnerability. Therefore, without loss of generality, we focus on the case where the current network is not vulnerable, since whenever vulnerability is detected we can report it in constant time for all subsequent additions. The second observation is that, in the non-vulnerable case, the algorithms in [9, 11] also compute the *MIS* of the network, which is a series-parallel graph.

Our algorithm maintains an up-to-date *MIS* throughout the sequence of edge insertions, achieving amortized linear time complexity. The key observation is that inserting an edge may turn previously redundant edges into irredundant ones. Let e be the newly inserted edge. We define its *Redundant Connected Component*, denoted by $RCC(e)$, as the set of redundant edges connected to e via paths that avoid all nodes of the *MIS*.

We distinguish three types of edge insertions (Fig. 2):

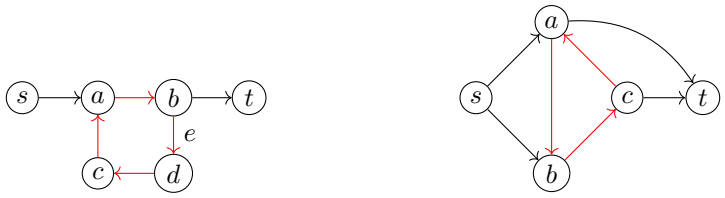
- *Impactless*: the $RCC(e)$ remains redundant in $G \cup \{e\}$ (all st-paths including e in Fig. 2 (1) are cyclic).
- *Destructive*: the $RCC(e)$ destroys the series-parallel structure of the *MIS* (adding e in Fig. 2 (2) creates a homeomorphic copy of $\mathcal{W}$).
- *Impactful*: the $RCC(e)$ preserves the series-parallel structure of the *MIS*, contains some edges that become irredundant in $G \cup \{e\}$, but the subnet induced by $RCC(e)$ may itself be vulnerable (see Fig. 2 (3) where the subnet $dfgh$ is $\mathcal{W}$).

In the first two cases, we provide an algorithm running in $O(m)$ time. In the third case, we invoke the algorithm of [11] to test whether the subnet induced by $RCC(e)$ is vulnerable. To maintain an amortized $O(m)$ complexity, we exploit the fact that the algorithm in [11], although having worst-case complexity $O(m^2)$, actually runs in $O(hm)$ time, where h is the number of irredundant edges. Since all edges in the processed $RCC(e)$ that become irredundant remain irredundant in all subsequent iterations of the incremental algorithm, the cost is incurred only once per edge, yielding an amortized $O(m)$ running time, assuming to have $\Theta(m)$ edge additions.

2.2. The Decremental Algorithm

In this section we summarize [13]. When we consider a network that evolves through edge deletions, we are in the *decremental* setting. In this case, monotonicity is reversed with respect to the incremental case: if a network is not vulnerable, then removing edges cannot make it vulnerable. Therefore, we focus on the case where the network is vulnerable, since once the network becomes non-vulnerable we can answer *Not Vulnerable* in constant time for all subsequent edge deletions.

To achieve amortized linear complexity, we build on the offline approach presented in [9]. The approach relies on the fact that st-connected redundant edges always belong to cycles. If the network is acyclic, then its st-connected part is irredundant, and vulnerability can be tested via a series-parallel check. To eliminate cycles, the algorithm repeatedly selects a cycle and analyzes it, with two possible outcomes: either a redundant edge is identified, or the entire network is declared vulnerable (Fig. 3). If a redundant edge is found, it can be ignored in all subsequent iterations, since edge deletions cannot turn a redundant edge into an irredundant one. Since each iteration either removes at least one edge



(1) (2)

Figure 3: The analyzed cycle is shown in red. In (1) the algorithm finds that e is redundant. In (2) the algorithm finds that the network is vulnerable.

from further analysis or concludes that the network is vulnerable, the offline procedure performs $O(m)$ cycle analyses.

We adapt this approach to the decremental setting. All edges marked as redundant remain redundant under further deletions and can therefore be ignored in the remainder of the dynamic process. When the decremental algorithm deletes an edge that has been previously classified as redundant, the vulnerability status remains unchanged and can therefore be reported in constant time. Otherwise, we remove the edge and rerun the offline algorithm on the remaining graph, excluding all edges previously identified as redundant and all the deleted edges.

The worst-case cost of each offline execution is $O(m)$ cycle analysis, matching the worst-case cost of the dynamic process, since each edge can be deleted or marked as redundant at most once. Therefore, the amortized complexity of the decremental algorithm reduces to the cost of cycle analysis. In [9], the cycle analysis has a cost of $O(nm)$. In [13] we refine that and obtain an $O(m)$ cycle analysis, thereby achieving an amortized $O(m)$ decremental algorithm assuming $\Theta(m)$ edge deletions.

2.3. The Duality Between the Two Algorithms

The two algorithms exhibit an interesting duality. The incremental algorithm is based on the *irredundant* part of the network, namely the *MIS*, since edge additions cannot turn irredundant edges into redundant ones. The decremental algorithm, instead, identifies and marks *redundant* edges, since edge deletions cannot make redundant edges irredundant.

Both algorithms exploit monotonicity in different ways, and we believe that the difficulty of designing a fully dynamic linear-time amortized algorithm lies in the need to handle both edge insertions and deletions simultaneously, thereby breaking this monotonic structure.

3. The Multi-Commodity Extension

In this paper, we have analyzed only *single-commodity* networks, i.e., networks with a single source s and sink t. We extend these results to multi-commodity networks. We define a k-commodity network as a tuple $(G, \{s_i\}_{i=1}^k, \{t_i\}_{i=1}^k)$, where each (G, s_i, t_i) for $i = 1, \ldots, k$ is a single-commodity network.

To solve *Vulnerability* in multi-commodity networks, we use the characterization in [17] based on [8]. We need to check two conditions:

- *Series-parallel condition*: each single-commodity *MIS* is series-parallel.
- *Coincidence condition*: the interaction between different commodities does not make the network vulnerable.

The coincidence condition can be checked in $O(km)$ time [17]. To check the series-parallel condition, we apply our two dynamic algorithms to each commodity separately. The *MIS* of a network is series-parallel if and only if the network is not vulnerable [7].

- *Incremental case*: we run the incremental algorithm independently on each commodity. If any commodity is vulnerable, then the whole multi-commodity network is vulnerable. Otherwise, we

obtain the *MIS* of each commodity. By [17], the *MIS* of the multi-commodity network is exactly the union of the *MIS*s of the individual commodities. Therefore, it only remains to check the coincidence condition, which can be done in $O(km)$ time. Since the amortized cost of maintaining the *MIS* for each commodity is $O(m)$, the overall amortized complexity is $O(km)$.

- *Decremental case*: we run the decremental algorithm on each commodity, keeping track of the edges marked as redundant separately. As long as at least one commodity is vulnerable, the coincidence condition does not need to be checked. When all commodities become non-vulnerable, we obtain their *MIS*, and thus the *MIS* of the multi-commodity network. From that point on, we only need to check the coincidence condition. The amortized cost of the decremental algorithm is $O(km)$, since the amortized cost per commodity is $O(m)$ and the coincidence condition is $O(km)$.

Theorem 3.1. *Vulnerability in a dynamically evolving k-commodity network can be checked under $\Theta(m)$ edge insertions (resp., deletions) in amortized $O(km)$ time per update.*

4. Conclusions

The fastest known algorithms for detecting vulnerability to Braess paradox in the offline setting run in $O(m^2)$ time for the single-commodity problem and $O(km^2)$ for the k-commodity case [11].

We studied the problem of detecting vulnerability to the Braess paradox in dynamically evolving networks. We summarize [12, 13] presenting two amortized $O(m)$ algorithms in the incremental and decremental settings, exploiting different monotonicity properties of irredundant and redundant structures. We further extended our approach to k-commodity networks, obtaining amortized $O(km)$ algorithms for both update models. Our results highlight a structural duality between the two dynamic settings: the incremental algorithm maintains the irredundant core of the network, while the decremental one focuses on eliminating redundant components.

A natural direction for future work is the design of a fully dynamic algorithm, where edge insertions and deletions are intermixed. This setting appears to be more challenging, as monotonicity can no longer be exploited, and thus requires different algorithmic techniques.

Declaration on Generative AI

Generative AI (ChatGPT) was used solely to improve the English language, including grammar and readability. It was not used to generate scientific content, analyses, interpretations, results, or conclusions. The author reviewed and approved the final manuscript and takes full responsibility for its content.

References

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

[2] M. Bell, Y. Iida, Transportation Network Analysis, Wiley, 1987.

[3] D. Braess, Über ein paradoxon aus der verkehrsplannung, Unternehmensforschung 12 (1968) 258–268.

[4] T. Roughgarden, On the severity of Braess's Paradox: Designing networks for selfish users is hard, J. Comput. Syst. Sci. 72 (2006) 922–953. An extended abstract appeared in the *Proc. of FOCS 2001*.

[5] I. Milchtaich, Network topology and the efficiency of equilibrium, Games and Economic Behavior 57 (2006) 321–346.

[6] J. Riordan, C. Shannon, The number of two-terminal series-parallel networks, Journal of Mathematics and Physiscs 21 (1942) 83–93.

[7] X. Chen, Z. Diao, X. Hu, Excluding braess paradox in nonatomic selfish routing, in: Proc. of SAGT15, volume 9347 of *LNCS*, Springer, 2015, pp. 219–230.
[8] X. Chen, Z. Diao, X. Hu, Network characterizations for excluding braess's paradox, Theory of Computing Systems (2016) 1–34.
[9] P. Cenciarelli, D. Gorla, I. Salvo, A polynomial-time algorithm for detecting the possibility of braess paradox in directed graphs, Algorithmica 81 (2019) 1535–1560.
[10] P. Cenciarelli, D. Gorla, I. Salvo, Inefficiencies in network models: A graph theoretic perspective, Information Processing Letters 131 (2018) 44–50.
[11] A. Matsubayashi, Y. Saito, A faster algorithm for recognizing directed graphs invulnerable to braess's paradox, in: D. Frigioni, P. Schiewe (Eds.), 23rd ATMOS, OASIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 12:1–12:19.
[12] D. Fiorenza, D. Gorla, I. Salvo, An incremental algorithm for checking the possibility of braess paradox in dynamic nets, 2026. Submitted.
[13] D. Fiorenza, D. Gorla, I. Salvo, A decremental algorithm for checking the possibility of braess paradox in dynamic nets, 2026. Submitted.
[14] C. Demetrescu, D. Eppstein, Z. Galil, G. F. Italiano, Dynamic graph algorithms, 2 ed., Chapman & Hall/CRC, 2010, pp. 1–28.
[15] J. H. Reif, A topological approach to dynamic graph connectivity, Inf. Process. Lett. 25 (1987) 65–70. doi:10.1016/0020-0190(87)90095-0.
[16] J. Holm, K. de Lichtenberg, M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity, J. ACM 48 (2001) 723–760.
[17] D. Fiorenza, D. Gorla, I. Salvo, Polynomial recognition of vulnerable multi-commodities, Information Processing Letters 179 (2023) 106282.