A Faster Algorithm for Recognizing Directed Graphs Invulnerable to Braess's Paradox

Akira Matsubayashi
Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

Yushi Saito
Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

Abstract

Braess's paradox is a counterintuitive and undesirable phenomenon, in which for a given graph with prescribed source and sink vertices and cost functions for all edges, removal of edges decreases the cost of a Nash flow from source to sink. The problem of deciding if the phenomenon occurs is generally NP-hard. In this paper, we consider the problem of deciding if, for a given graph with prescribed source and sink vertices, Braess's paradox does not occur for any cost functions. It is known that this problem can be solved in $O(nm^2)$ time for directed graphs, where n and m are the numbers of vertices and edges of the input graph, respectively. In this paper, we propose a faster $O(m^2)$ time algorithm solving this problem for directed graphs. Our approach is based on a simple implementation of a known characterization that the subgraph of a given graph induced by all source-sink paths is series-parallel. The faster running time is achieved by speeding up the simple implementation using another characterization that a certain structure is embedded in the given graph. Combined with a known technique, the proposed algorithm can also be used to design a faster $O(km^2)$ time algorithm for directed graphs with k source-sink pairs, which improves the previous $O(knm^2)$ time algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Network flows; Mathematics of computing → Graph algorithms; Mathematics of computing → Network flows

Keywords and phrases Braess's paradox, series-parallel graph, route-induced graph, Nash flow

Digital Object Identifier 10.4230/OASIcs.ATMOS.2023.12

Funding *Akira Matsubayashi*: This work was supported by JSPS KAKENHI Grant Number 23K10984.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments.

1 Introduction

Braess's paradox is a counterintuitive and undesirable phenomenon, in which for a given two-terminal graph G with prescribed vertices (or terminals) s and t, and nonnegative, continuous, nondecreasing cost functions $\{c_e\}$ for every edge e, removal of an edge decreases the cost of a Nash flow (or Wardrop flow) from s to t. Here, a network is modeled by the two-terminal graph G, in such a way that a pair of source and sink (or origin and destination) of the network is represented by the vertices s and t, respectively, and that the latency or any other cost depending on the amount x of users passing through each edge e is represented by the edge cost function $c_e(x)$. Nash flow is a flow from s to t in the graph reaching an equilibrium among selfish users, each of which chooses a route from s to t that incurs the minimum cost for the user.

Braess's paradox was first published in 1968 [2][1] and has been quite extensively studied in wide range of engineering, but it was not until 2001 that the computational complexity of detection of Braess's paradox was proved by Roughgarden [11, 13]. Specifically, given a two-terminal graph G and cost functions $\{c_e\}$, the problem of deciding if Braess's paradox occurs is NP-complete. Besides, for the problem of network design, i.e., finding a subgraph of G with the minimum Nash flow cost, $(4/3-\epsilon)$-approximation for linear cost functions and $(\lfloor n/2 \rfloor - \epsilon)$-approximation for general cost functions, where n is the number of vertices, are both NP-hard [11, 13]. Formal definitions of Nash flow and Braess's paradox are provided in Section 2.4.

Roughgarden [13] also raised a relaxed variation of the problem focusing on graph structure that can cause Braess's paradox without considering cost functions as input. This property of graphs was called *vulnerability* in [13]. Milchtaich [10] characterized undirected two-terminal graphs that are not vulnerable, or *paradox-free* [8], i.e., that do not admit Braess's paradox for any cost functions.

▶ **Theorem 1** ([10]). *Let G be an undirected two-terminal graph such that every edge is contained in a path from source to sink. Then, G is paradox-free if and only if G is series-parallel.*

A counterpart for directed graphs was proved by Chen, Diao, and Hu [8].

▶ **Theorem 2** ([8]). *Let G be a directed two-terminal graph such that every edge is contained in a path from source to sink. Then, G is paradox-free if and only if G is series-parallel.*

Theorems 1 and 2 raise a question: Can we decide in polynomial time if Braess's paradox occurs in a general given two-terminal graph G that may have vertices and edges contained in none of source-sink paths, for some cost functions? Obviously only edges contained in a (simple) source-sink path have a positive flow in any Nash flow. Therefore, a straightforward approach to this question would be to first construct the subgraph $\tilde{G}$ of G induced by all edges contained in a path from source to sink, called the *maximally irredundant* or *route-induced* subgraph,[2] and then to check if $\tilde{G}$ is series-parallel. This idea succeeds if G is undirected [8], because $\tilde{G}$ can be obtained through finding biconnected components of G in linear time [14], and (directed and undirected) series-parallel graphs can be recognized in linear time [15]. However, this approach fails for directed graphs, because computing the route-induced subgraph $\tilde{G}$ of a directed graph G is generally NP-hard [7]. This means that any polynomial time algorithm deciding if a given directed two-terminal graph G is paradox-free must (implicitly or explicitly) solve the problem of deciding if the route-induced subgraph $\tilde{G}$ is series-parallel in polynomial time without construction of $\tilde{G}$, unless P = NP. Although this seems to be intractable as conjectured in [8], Cenciarelli, Gorla, and Salvo [7] succeeded in designing a polynomial time algorithm. The authors of [7] presented a constructive (but somewhat complicated) proof for a characterization [8, 6] that directed vulnerable graphs contain a subgraph homeomorphic to the graph shown in Fig. 1, which is called the *Wheatstone network*, and derived from the constructive proof an $O(nm^2)$ time algorithm to detect such a subgraph, where n and m are the numbers of vertices and edges of G, respectively.

[1] English version of [2] (in German) is published as [3].
[2] In the terminology of [8, 7, 9], this graph is said to be maximum or maximal(ly) irredundant. We introduce and mainly use the more self-explanatory term "route-induced" in this paper.

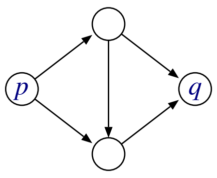

Figure 1 The Wheatstone network.

In this paper, we propose a faster $O(m^2)$ time algorithm for deciding if a given directed two-terminal graph G is paradox-free. Our approach is based on a simple implementation of Theorem 2, which decides if the route-induced subgraph $\tilde{G}$ of G is series-parallel. More specifically, we check if $\tilde{G}$ satisfies a recursive characterization of series-parallel graphs, i.e., $\tilde{G}$ is decomposed into single edges by a sequence of series and parallel decompositions. Since it is unrealistic to depend on the complete information of $\tilde{G}$, instead, we recursively try to decompose G into the maximum number of subgraphs $G_1, \ldots, G_\ell$ in such a way that $\tilde{G}$ is obtained either by series decompositions or by parallel compositions of the route-induced subgraphs $\tilde{G}_1, \ldots, \tilde{G}_\ell$ of $G_1, \ldots, G_\ell$, respectively. This can be performed in polynomial time without complete information of $\tilde{G}$ or $\tilde{G}_1, \ldots, \tilde{G}_\ell$ as we prove in this paper.

It was conjectured in [7] that not a characterization in terms of the route-induced subgraph (as Theorem 2) but a characterization in terms only of the input graph (as the inclusion of the Wheatstone network [8, 6]) would be necessary to design a polynomial time algorithm for checking vulnerability. Our implementation of Theorem 2 disproves this conjecture.

To achieve $O(m^2)$ time complexity, we also use the characterization of [8, 6] (but not the constructive proof in [7]) that G is vulnerable, i.e., $\tilde{G}$ is not series-parallel, if and only if G contains a subgraph homeomorphic to the Wheatstone network.

Our algorithm, as well as the algorithm of [7], can be used to design an algorithm for multicommodity networks modeled by directed $2k$-terminal graphs with $k > 1$ source-sink pairs. Chen et al. [8] defined a naturally extended concept of paradox-freeness for $2k$-terminal graphs with $k > 1$ (see [8] for the definition). Under the extended definition, they generalized Theorems 1 and 2 to undirected and directed $2k$-terminal graphs and proposed a polynomial time algorithm for deciding if a given undirected $2k$-terminal graph is paradox-free. For the directed case, Fiorenza, Gorla, and Salvo [9] presented an $O(knm^2)$ time algorithm. Their k-commodity algorithm for $2k$-terminal graphs calls the single-commodity algorithm of [7] for two-terminal graphs as a subroutine. The crucial property exploited by [9] is that if the input directed two-terminal graph G is paradox-free, then the single-commodity algorithm of [7] not only returns the result of decision but also produces the route-induced subgraph of G, with simple modification. Actually, the k-commodity algorithm of [9] can use any single-commodity algorithm with this property as a subroutine, and essentially runs in time of k executions of the subroutine. Our algorithm has this property as well; therefore, we can obtain a faster $O(km^2)$ time algorithm for deciding if a given directed $2k$-terminal graph is paradox-free (in the sense of the definition of [8]).

This paper is organized as follows: We describe some notation and definitions in Section 2. In Section 3, we present our algorithm for deciding if a given directed two-terminal graph is paradox-free, together with analysis of the correctness and time complexity. We conclude this paper in Section 4.

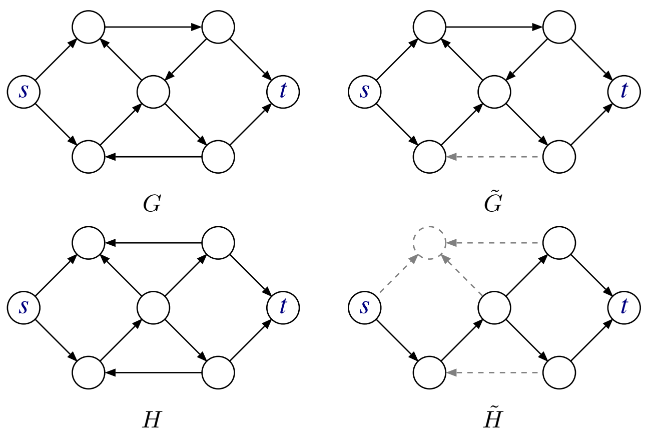

Figure 2 Two-terminal graphs G and H and their route-induced subgraphs $\tilde{G}$ and $\tilde{H}$ (solid vertices and edges), respectively.

2 Preliminaries

2.1 Graphs and Paths

Graphs considered in this paper are directed or undirected, and may have multiple (or parallel) edges joining the same pair of vertices, but no loops joining a single vertex. A *path* P in a directed (undirected, resp.) graph consists of a sequence of distinct vertices $(v_1, \dots, v_\ell)$ and directed (undirected, resp.) edges (v_i, v_{i+1}) for all $1 \le i < \ell$. The path may be *empty*, i.e., $v_1 = v_\ell$. The *end-vertices* of P are v_1 and v_ℓ. The *internal vertices* of P are vertices of P except its end-vertices, i.e., $v_2, \dots, v_{\ell-1}$. A directed path that starts from s and ends with t, i.e., has end-vertices s and t and edges leaving s and entering t, is called an *st-path.* For an undirected path with end-vertices s and t, we may call it an st-path or a ts-path. We define that two paths *intersect* if an internal vertex of one of the paths is also an internal vertex of the other path. Two paths that do not intersect are said to be *internally vertex disjoint*, or simply *disjoint*.

2.2 Two-terminal graphs and Route-Induced Subgraphs

A *two-terminal graph* (or a *single-commodity graph*) is a graph that has two prescribed distinct vertices representing *source* and *sink*. For a two-terminal graph G with source s and sink t, a vertex or an edge of G is said to be *irredundant* if it is contained in an st-path of G, *redundant* otherwise. The graph G is said to be *irredundant* if all edges (and hence all vertices) of G are irredundant, and *redundant* otherwise. The *maximally irredundant* or *route-induced* subgraph, denoted by $\tilde{G}$, is the subgraph of G induced by all irredundant edges. The route-induced subgraph $\tilde{G}$ is also defined as the subgraph obtained as the graph union of all st-paths of G. Examples are shown in Fig. 2. We note that route-induced subgraphs are not necessarily vertex-induced subgraphs, as the graphs $\tilde{G}$ and $\tilde{H}$ in Fig. 2.

2.3 Series-Parallel Graphs

Suppose that G_1 and G_2 are directed or undirected two-terminal graphs, such that for each $i \in \{1, 2\}$, G_i has source s_i and sink t_i. The *series composition of* G_1 *and* G_2 is to compose the new two-terminal graph from G_1 and G_2 by identifying t_1 and s_2, and by setting s_1 and

t_2 to the new source and sink, respectively. The *parallel composition of* G_1 *and* G_2 is to compose the new two-terminal graph from G_1 and G_2 by identifying s_1 and s_2 as the new source, and t_1 and t_2 as the new sink. A (two-terminal) series-parallel graph is recursively defined as follows.

▶ **Definition 3** (series-parallel graphs).
1. *A single edge* (s, t) *is a series-parallel graph with source* s *and sink* t*.*
2. *A graph obtained from two series-parallel graphs by series or parallel composition is series-parallel.*

As an example, the graph $\tilde{H}$ in Fig. 2 is series-parallel, but the rest in Fig. 2 are not.

2.4 Nash Flows and Braess's Paradox

Let $G = (V, E)$ be a directed or undirected two-terminal graph with source s and sink t, and for each edge $e \in E$, let $c_e : \mathbb{R}_+ \to \mathbb{R}_+$ be a nonnegative, continuous, nondecreasing cost function. We associate a *traffic rate* $r \geq 0$ with the source-sink pair. Let $\mathcal{P}$ be the set of all st-paths in G. We assume $\mathcal{P} \neq \emptyset$ in this paper. A *flow vector* (or simply *flow*) f is a nonnegative real vector $(f_P)_{P\in\mathcal{P}}$. A flow f is said to be *feasible* if $\sum_{P\in\mathcal{P}} f_P = r$. A *flow on an edge* e is defined as $f_e = \sum_{P\in\mathcal{P}:e\in P} f_P$. The *cost of a path* $P \in \mathcal{P}$ *with respect to a flow* f is defined as $c_P(f) = \sum_{e\in P} c_e(f_e)$. The *cost of a flow* f is defined as $c(f) = \sum_{P\in\mathcal{P}} c_P(f) f_P$, which is equal to

$$\sum_{P\in\mathcal{P}} \left(\sum_{e\in P} c_e(f_e) \right) f_P = \sum_{e\in E} \left(\sum_{P\in\mathcal{P}:e\in P} f_P \right) c_e(f_e) = \sum_{e\in E} c_e(f_e) f_e.$$

A feasible flow f is at *Nash equilibrium* (*Wardrop equilibrium*), or called a *Nash flow* (*Wardrop flow*), if and only if for all $P, P' \in \mathcal{P}$ with $f_P > 0$, $c_P(f) \leq c_{P'}(f)$. Note that this means that all paths in $\mathcal{P}$ with positive flows have the same cost in a Nash flow. It is known that there exists a Nash flow for any instance (G, r, c), where $c = (c_e)_{e\in E}$, and that all Nash flows have the same cost. For any Nash flows f and f', specifically, it follows that $c_e(f_e) = c_e(f'_e)$ for every edge $e \in E$, and hence, $c_P(f) = c_P(f')$ for any path $P \in \mathcal{P}$. See, e.g., [12] for further detailed discussion.

Braess's paradox occurs in the instance (G, r, c) if removal of some edges of G decreases the unique cost of a Nash flow, i.e., there exists a spanning subgraph $H = (V, E')$ of G such that

$$d(H, r, c) < d(G, r, c),$$

where $d(H, r, c)$ and $d(G, r, c)$ are the unique costs of Nash flows for the instances (H, r, c) and (G, r, c), respectively. If there exist a traffic rate r and cost functions $c = (c_e)_{e\in E}$ such that Braess's paradox occurs in the instance (G, r, c), then we define that *Braess's paradox can occur in* G, or G *is paradox-ridden* or *vulnerable*. Any graph that is not vulnerable is said to be *paradox-free*.

The following is a characterization of directed vulnerable graphs, which we use in our algorithm.

▶ **Theorem 4** ([8, 6]). *A directed two-terminal graph* G *with source* s *and sink* t *is vulnerable if and only if there is an* st-embedding $\langle\phi, \rho\rangle$ *of the Wheatstone network in Fig. 1 into* G*. Here,* ϕ *is an injective mapping from the vertices in Fig. 1 to the vertices of* G*, and* ρ *maps each edge in Fig. 1, denoted by* (u, v)*, to a* $\phi(u)\phi(v)$*-path in* G*, as well as constructs a (possibly empty)* $s\phi(p)$*-path and a (possibly empty)* $\phi(q)t$*-path, in such a way that all these paths are disjoint with each other.*

3 Algorithm for Directed Graphs

Our algorithm recursively performs series and parallel decompositions, which are similar to the inverse operations of series and parallel compositions, respectively. Specifically, there are two goals of series and parallel decompositions in our algorithm: One is to find the maximum number of two-terminal subgraphs $G_1, \dots G_k$ of an input graph G, in such a way that the route-induced subgraph $\tilde{G}$ of G is obtained either by series compositions or by parallel compositions of the route-induced subgraphs $\tilde{G}_1, \dots \tilde{G}_k$ of $G_1, \dots G_k$, respectively. The redundant vertices and edges in $G - \tilde{G}$ may or may not remain in the resulting subgraphs $G_1, \dots G_k$. The other goal is that the subgraphs $G_1, \dots G_k$ are almost separated, by which the running time is reduced. Actually, only terminals of each of the subgraphs may be shared by another subgraph. The series and parallel decompositions are implemented using depth-first search (DFS) on G, as defined in this section. To obtain an $O(m^2)$ time implementation for graphs with m edges, the parallel decomposition algorithm quits when an st-embedding of the Wheatstone network is detected. We decide that G is paradox-free, i.e., $\tilde{G}$ is series-parallel if and only if $\tilde{G}$ is decomposed into a collection of single edges by recursive executions of the series and parallel decompositions with detecting no st-embedding of the Wheatstone network. Because the series and parallel decomposition algorithms preserve irredundant edges as the first goal above, the proposed algorithm can produce the series-parallel $\tilde{G}$ if G is paradox-free.

We define and analyze the series and parallel decomposition algorithms in Sections 3.1 and 3.2, respectively, and the main procedure of the proposed algorithm in Section 3.3. To simplify the discussion, we assume without loss of generality that the input graph G with m edges has $O(m)$ vertices. Note that this assumption is simply implied by weak connectivity, and hence affects neither the vulnerability of G nor the time complexity of our algorithm.

3.1 Series Decomposition

3.1.1 Idea and Definition of Series Decomposition Algorithm

Our series decomposition algorithm, called `Series_Decomposition`, is based on the simple observation that the route-induced subgraph $\tilde{G}$ of an input graph G is obtained by series composition of two graphs H_1 and H_2, identifying the sink t_1 of H_1 and the source s_2 of H_2, if and only if all st-paths in G contain the identified vertex $t_1 = s_2$ (Lemma 5). We call such a vertex, including s and t, an *st-articulation point*. All st-articulation points can be found in linear time using several algorithms [1, 4, 5] (Step 1). Lemma 5 implies that all st-articulation points appear in the same order on all st-paths. If $v_0 = s, v_1, \dots, v_{k-1}, v_k = t$ are st-articulation points appearing in this order on an st-path, then for $1 \le i \le k$, we define G_i as the graph induced by the vertices reachable from v_{i-1} with passing through neither v_i nor vertices reachable from v_{j-1} with $j < i$ (Step 2). In this way, the route-induced subgraph $\tilde{G}$ is series decomposed into the route-induced subgraphs $\tilde{G}_1, \dots, \tilde{G}_k$ as desired (Lemma 6). The following is a high level pseudocode of `Series_Decomposition`.

Algorithm `Series_Decomposition`(G, s, t)

Input A directed two-terminal graph G with source s and sink t.

Output The maximum number k of two-terminal subgraphs $G_1, \dots, G_k$ of G, such that $\tilde{G}$ is obtained by series composition of $\tilde{G}_1, \dots, \tilde{G}_k$, and that for each $1 \le i < k$, G_i and $\bigcup_{j>i} G_j$ share v_{i+1} only.

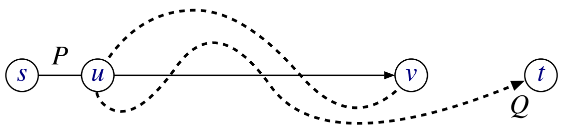

Figure 3 Intersecting sv-path P and vt-path Q.

1. Find all st-articulation points $v_0 = s, v_1, \ldots, v_{k-1}, v_k = t$ appearing in this order on an st-path.
2. For $i = 1$ to k, perform the following:
   a. Find a set V_i of vertices x such that there exists a $v_{i-1}x$-path in G consisting of edges neither leaving v_i nor entering a vertex in $\bigcup_{j=1}^{i-1} V_j$.
   b. Return the graph induced by V_i as G_i.

3.1.2 Analysis of `Series_Decomposition`

We prove the correctness of `Series_Decomposition` in Lemmas 5 and 6 below, together with the time complexity in Lemma 7.

▶ **Lemma 5.** *For a directed two-terminal graph G with source s and sink t, the route-induced subgraph $\tilde{G}$ is obtained by series composition of some graphs H_1 and H_2 if and only if there exists an st-articulation point $v \notin \{s,t\}$.*

Proof. The necessity ($\Rightarrow$) is immediate by the definition of series composition. Specifically, if $\tilde{G}$ is obtained by identifying the sink t_1 of a graph H_1 and the source s_2 of a graph H_2, then s and t must be contained in H_1 and H_2, respectively, and all st-paths of G must pass through the vertex $t_1 = s_2$ in G.

For the sufficiency ($\Leftarrow$), let $v \notin \{s,t\}$ be an st-articulation point. If an sv-path P and a vt-path Q intersect, then let u be the internal vertex of both P and Q that appears first on P. Then, we obtain the st-path avoiding v, which proceeds from s to u along P and then from u to t along Q (Fig. 3). This contradicts that v is an st-articulation point. Therefore, any sv-path and any vt-path are distinct. This means that $\tilde{G}$ is obtained by series composition of two subgraphs induced by all sv-paths and vt-paths. ◀

▶ **Lemma 6.** *`Series_Decomposition` returns the maximum number k of two-terminal subgraphs $G_1, \ldots, G_k$ of an input graph G such that $\tilde{G}$ is obtained by series composition of $\tilde{G}_1, \ldots, \tilde{G}_k$, and that for each $1 \le i < k$, G_i and $\bigcup_{j>i} G_j$ share v_i only.*

Proof. In Step 1, we find st-articulation points $v_0 = s, v_1, \ldots, v_{k-1}, v_k = t$ appearing in this order on an st-path. Lemma 5 implies that $\tilde{G}$ is obtained by series compositions of k graphs $H_1, \ldots, H_k$, where H_i has the source v_{i-1} and sink v_i, and that k is the maximum number of such graphs. We observe the following claim.

▷ Claim. For each $1 \le i \le k$, the graph G_i defined in Step 2 contains no vertices in $\bigcup_{j>i} H_j - v_i$.

Proof. The claim holds because for $1 \le i < j \le k$, any vertex $x \neq v_i$ contained in both G_i and H_j would yield an st-path avoiding v_i, which proceeds from s to v_{i-1} through $H_1, \ldots, H_{i-1}$, from v_{v-1} to x in G_i, and then from x to t through $H_j, \ldots, H_k$ (Fig. 4). ◁

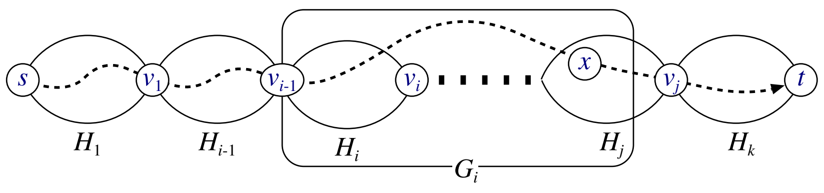

Figure 4 A vertex $x \neq v_i$ in both G_i and H_j with $j > i$ yields an st-path avoiding v_i.

Now we prove the lemma by showing that $\tilde{G}_i$ and H_i are identical for each $1 \leq i \leq k$. Since $\tilde{G}$ is induced by all the st-paths and obtained by series compositions of $H_1, \dots, H_k$, every vertex x of H_i is on a $v_{i-1}v_i$-path, denoted by Q^x, consisting of edges not leaving v_i. In addition, Q^x has no internal vertices in the graph $\bigcup_{j<i} G_j$ by the above claim. Thus, the $v_{i-1}x$-subpath of Q^xconsists of edges neither leaving v_i nor entering a vertex in the graph $\bigcup_{j<i} G_j$, implying that H_i is a subgraph of $\tilde{G}_i$. On the other hand, since $\tilde{G}_i$ is the subgraph induced by all $v_{i-1}v_i$-paths in G_i, every vertex y of $\tilde{G}_i$ is on a $v_{i-1}v_i$-path, denoted by Q^y. The path Q^y has no internal vertices in the graph $\bigcup_{j>i} H_j$ by the above claim. Moreover, Q^y has no internal vertices in the graph $\bigcup_{j<i} H_j$, since G_i contains no vertices in $\bigcup_{j<i} G_j$ by the definition of Step 2, and since H_j is a subgraph of $\tilde{G}_j$. Therefore, the path Q^y is included in H_i, and hence $\tilde{G}_i$ is a subgraph of H_i. Thus, the graphs $\tilde{G}_i$ and H_i are identical.

It is obvious by the definition of Step 2 that for each $1 \leq i < k$, G_i and $\bigcup_{j>i} G_j$ share v_i only. We thus conclude that `Series_Decomposition` returns desired subgraphs. ◀

▶ **Lemma 7.** *`Series_Decomposition` runs in $O(m)$ time for an input graph G with m edges.*

Proof. Step 1 can be executed in linear time using one of the algorithms in [1, 4, 5]. Step 2 can be implemented as graph search, e.g., DFS from v_{i-1} for each $1 \leq i \leq k$. Since no edge entering a vertex in V_j with $j < i$ is visited by the ith search from v_{i-1}, each edge is visited at most once. Therefore, Step 2 finishes also in linear time. Thus `Series_Decomposition` runs in $O(m)$ time. ◀

3.2 Parallel Decomposition

We describe two versions of our parallel decomposition algorithm. We first present a base version in Section 3.2.1, which depends on Theorem 2 but not on Theorem 4, and prove its correctness and the polynomial time complexity in Section 3.2.2. Our main purpose of presenting this version is to prove the correctness of the base idea of our algorithm. We then present a faster version with improved implementation using Theorem 4 in Section 3.2.3.

3.2.1 Idea and Definition of Parallel Decomposition Algorithm

To describe the idea of our parallel decomposition algorithm, it is convenient to represent st-paths of an input graph G as another undirected graph, called the *route intersection graph*, which is obtained by creating a vertex for each st-path and an edge for any two intersecting st-paths in G. On the basis of the route intersection graph, we introduce graph notion, such as adjacency and distance, into st-paths: Two st-paths are said to be *adjacent* to each other if they intersect in G, and the *distance* between two st-paths P and P' is the distance between them in the route intersection graph, i.e., the minimum number h of pairs of adjacent st-paths Q_{i-1} and Q_i, $1 \leq i \leq h$, such that $Q_0 = P$ and $Q_h = P'$. To avoid confusion, we use the terms *chains* and *chained* for the notions "paths" and "connected" in the route intersection graph, respectively. In particular, connected components in the route intersection graph are called *chained components* below.

A key observation is that the goal of our parallel decomposition algorithm is to decompose G into subgraphs, in such a way that st-paths are partitioned into the chained components (Lemma 9). To this end, in the base version called `Parallel_Decomposition`, we begin by finding a maximal number of disjoint st-paths $P_1, \ldots, P_\ell$ (Step 1). By the maximality of ℓ, any remaining st-path is adjacent to some P_i. We find st-paths adjacent to P_i, and put them together as one subgraph, by searching for vertices x such that there are ux-path and xv-path for certain vertices u and v on P_i (Step 2). At this point every st-path is contained in at least one of subgraphs emerged from paths $P_1, \ldots, P_\ell$ (Lemma 10). We then find subgraphs sharing an internal vertex and put them together as one subgraph (Step 3). This procedure merges paths P_i and P_j within distance 3, together with paths adjacent to them, into one subgraph (Lemmas 11 and 12). Conversely, we can prove that two subgraphs are merged by this procedure only if they contain such P_i and P_j within distance 3 (Lemma 13). The algorithm thus yields desired subgraphs, each of which contains st-paths composing a chained component in the route intersection graph (Lemma 14).

The following is a high level pseudocode of `Parallel_Decomposition`.

```
Algorithm Parallel_Decomposition($G, s, t$)
Input  A directed two-terminal graph $G$ with source $s$ and sink $t$.
Output  The maximum number $k$ of two-terminal subgraphs $G_1, \ldots, G_k$ of $G$, sharing $s$ and
  $t$ only, such that $\tilde{G}$ is obtained by parallel composition of $\tilde{G}_1, \ldots, \tilde{G}_k$.

1. Find a maximal number of disjoint $st$-paths $P_1, \ldots, P_\ell$ of $G$ greedily.
2. For each $1 \le i \le \ell$, let $V_i$ be the vertex set obtained from the vertex set of $P_i$ by adding
   every vertex $x$ satisfying the following condition.
   ▶ Condition 8. The vertex $x$ is not contained in $P_i$, and there exist (not necessarily
   distinct) vertices $u$ and $v$ in $P_i$, such that
   a. there are a $ux$-path and an $xv$-path in $G$, each of which is disjoint with $P_i$, and
   b. $u \notin \{s, t\}$ and $v \notin \{s, t\}$, or $u = s$ and $v \notin \{s, t\}$, or $u \notin \{s, t\}$ and $v = t$.
3. If $V_i$ and $V_j$ $(i < j)$ share a vertex that is neither $s$ nor $t$, then $V_i = V_i \cup V_j$ and $V_j = \emptyset$.
   Perform this process as long as two sets sharing a vertex neither $s$ nor $t$ exist.
4. For each $i$ such that $V_i \neq \emptyset$, return the subgraph of $G$ induced by $V_i$.
```

3.2.2 Analysis of `Parallel_Decomposition`

We prove the correctness of `Parallel_Decomposition` in Lemmas 9–14 below, together with the polynomial time complexity in Lemma 15.

▶ **Lemma 9.** *Let $C_1, \ldots, C_k$ be the sets of st-paths in all the chained components of the route intersection graph of an input graph G. Then, the route-induced subgraph $\tilde{G}$ of G is obtained by parallel compositions of the maximum number k of graphs $H_1, \ldots, H_k$, which are induced by the edges of the st-paths in $C_1, \ldots, C_k$, respectively.*

Proof. For each $1 \le i \le k$, let H_i be the subgraph of G induced by the edges of the st-paths in C_i. Then, $\tilde{G}$ can be obtained by parallel composition of the two graphs H_i and $\bigcup_{j \neq i} H_j$, because any st-path in G is contained in exactly one of the sets $C_1, \ldots, C_k$, and because any st-path in C_i and any st-path not in C_i are disjoint. Moreover, the graph H_i cannot be obtained by parallel composition of two smaller graphs, because for any partition of C_i into two non-empty disjoint subsets, there are two intersecting st-paths contained in the different subsets. Therefore, $\tilde{G}$ can be obtained by parallel compositions of $H_1, \ldots, H_k$ but not of more than k graphs. ◀

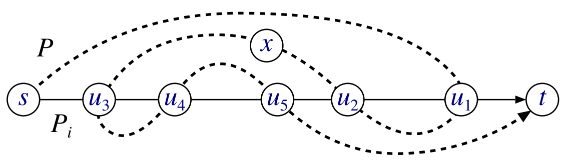

Figure 5 Intersecting st-paths P and P_i, and a vertex x in P but not in P_i.

▶ **Lemma 10.** *For any st-path P of a graph G input to `Parallel_Decomposition`, there exists $1 \leq i \leq \ell$ such that P and P_i intersect or $P = P_i$. Moreover, all vertices of P are contained in V_i for each such i after Step 2.*

Proof. If $P = P_i$ for some i, then V_i contains the vertices of P by definition, and therefore, the lemma holds for this case.

Assume otherwise. By the maximality of the number ℓ of disjoint st-paths, there exists $1 \leq i \leq \ell$ such that P and P_i intersect, i.e., share at least one internal vertex. For each such i, let $u_0 = s, u_1, \ldots, u_{h-1}, u_h = t$ ($h \geq 2$) be all the vertices shared by P and P_i and appearing in this order on P. For each $0 \leq j < h$, each internal vertex x on the subpath of P from u_j to u_{j+1} is not in P_i, and added to V_i at Step 2 because of the $u_j u_{j+1}$-path disjoint with P_i (Fig. 5). Here, we observe by $h \geq 2$ that $u_j \notin \{s,t\}$ and $u_{j+1} \notin \{s,t\}$ for $0 < j < h-1$, $u_j = s$ and $u_{j+1} \notin \{s,t\}$ for $j = 0$, and $u_j \notin \{s,t\}$ and $u_{j+1} = t$ for $j = h-1$. We thus have the lemma. ◀

▶ **Lemma 11.** *For any $1 \leq i < j \leq \ell$, if there is an st-path Q such that P_i and Q intersect, and Q and P_j intersect, then V_i and V_j are merged in Step 3.*

Proof. Under the assumption of the lemma, P_i and Q share an internal vertex x. By Lemma 10, x is contained in V_i after Step 2. Since Q and P_j intersect, x is also contained in V_j after Step 2 by Lemma 10. Therefore, V_i and V_j are merged in Step 3, since they share the vertex x that is neither s nor t. ◀

▶ **Lemma 12.** *For any $1 \leq i < j \leq \ell$, if there are two distinct st-paths Q and Q' such that P_i and Q intersect, Q and Q' intersect, and Q' and P_j intersect, then V_i and V_j are merged in Step 3.*

Proof. Under the assumption of the lemma, all vertices of Q are contained in V_i, and all vertices of Q' are contained in V_j, both after Step 2 by Lemma 10. Moreover, Q and Q' share an internal vertex x. This implies that x, which is neither s nor t, is contained in both V_i and V_j. Therefore, V_i and V_j are merged in Step 3. ◀

▶ **Lemma 13.** *If two vertex sets, denoted by V and V', are merged in Step 3, then there exist i and j with $1 \leq i \leq \ell$, $1 \leq j \leq \ell$, and $i \neq j$ such that the vertex sets of the st-paths P_i and P_j are included in V and V', respectively, and that at least one of the following conditions is satisfied.*

1. *There is an st-path Q such that P_i and Q intersect, and Q and P_j intersect.*
2. *There are st-paths Q and Q' such that P_i and Q intersect, Q and Q' intersect, and Q' and P_j intersect.*

Proof. Under the assumption of the lemma, the sets V and V' share a vertex x that is neither s nor t. Since st-paths $P_1, \ldots, P_\ell$ are disjoint with each other, x is contained in at most one of these paths and added to V and/or V' in Step 2. We assume without loss of generality that x is added to V in Step 2. Then, there exists $1 \le i \le \ell$ such that the vertices of P_i are included in V, and Condition 8 for x to be added to V_i in Step 2 is satisfied. Specifically, there are a ux-path Q^u and an xv-path Q^v for some vertices u and v of P_i, such that these paths are both distinct with P_i, and that $u \notin \{s,t\}$ and $v \notin \{s,t\}$, or $u = s$ and $v \notin \{s,t\}$, or $u \notin \{s,t\}$ and $v = t$. We prove by cases.

1. Suppose that $u \notin \{s,t\}$ and x is contained in no st-path P_j with $j \ne i$ whose vertices are included in V'. Then, x is added to V' in Step 2, and therefore, there exists $j \ne i$ such that the vertices of P_j are included in V', and Condition 8 for x to be added to V_j in Step 2 is satisfied. Specifically, there are a yx-path Q^y and an xz-path Q^z for some vertices y and z of P_j, such that these paths are both distinct with P_j, and that $y \notin \{s,t\}$ and $z \notin \{s,t\}$, or $y = s$ and $z \notin \{s,t\}$, or $y \notin \{s,t\}$ and $z = t$.

   a. If Q^u and P_j intersect, then let r be the internal vertex of both Q^y and P_j that appears first on Q^u. Then, we obtain the st-path Q proceeding $s \to u \to r \to t$ (Fig. 6(a)). The paths Q and P_i intersect at u, and Q and P_j intersect at r.
   b. If Q^v and P_j intersect, then let r be the internal vertex of both Q^v and P_j that appears last on Q^v. Then, we obtain the st-path Q proceeding $s \to r \to v \to t$ (Fig. 6(b)). The paths Q and P_i intersect at v, and Q and P_j intersect at r.
   c. If Q^y or Q^z intersects with P_i, then we can prove the existence of an st-path Q intersecting with P_i and P_j as in the case 1a or 1b, with exchanged roles of i and j, u and y, and v and z. (Fig. 6(c)-i,ii).
   d. Assume that both Q^u and Q^v are disjoint with P_j, and both Q^y and Q^z are disjoint with P_i.
      i. If $z \ne t$, then let r be the vertex of Q^z appearing first on Q^u. The vertex r is identical with x if Q^u and Q^z are disjoint. Then, we obtain the st-path Q proceeding $s \to u \to r \to z \to t$ (Fig. 6(d)-i). The paths Q and P_i intersect at u, and Q and P_j intersect at z.
      ii. If $z = t$, then let r be the vertex of Q^z that appears first on Q^u, and let w be the vertex of Q^z that appears first on Q^y. Then, we obtain two st-paths Q proceeding $s \to u \to r \to t$ and Q' proceeding $s \to y \to w \to t$ (Fig. 6(d)-ii). The paths Q and P_i intersect at u, Q and Q' intersect at one of the vertices r and w that appears latter on Q^z, and Q' and P_j intersect at y.

2. Suppose that $u \notin \{s,t\}$ and x is contained in P_j for some $j \ne i$ whose vertices are included in V'. We can prove for this case as in the case 1a.
3. Suppose $u = s$, implying $v \notin \{s,t\}$. Let G' be the graph obtained from G by reversing the direction of every edge. Then, any path in G is a path in G' with reverse direction, and vice versa. Also, vertex sets $V_1, \ldots, V_\ell$ created and processed in the algorithm are exactly the same for G' as for G. Therefore, this case can be reduced to the case 1 or 2 with exchanged roles of s and t, u and v, and y and z.

In all cases, there is an st-path Q that intersect with P_i and P_j, or there are two intersecting st-path Q and Q' that intersect P_i and P_j, respectively. ◀

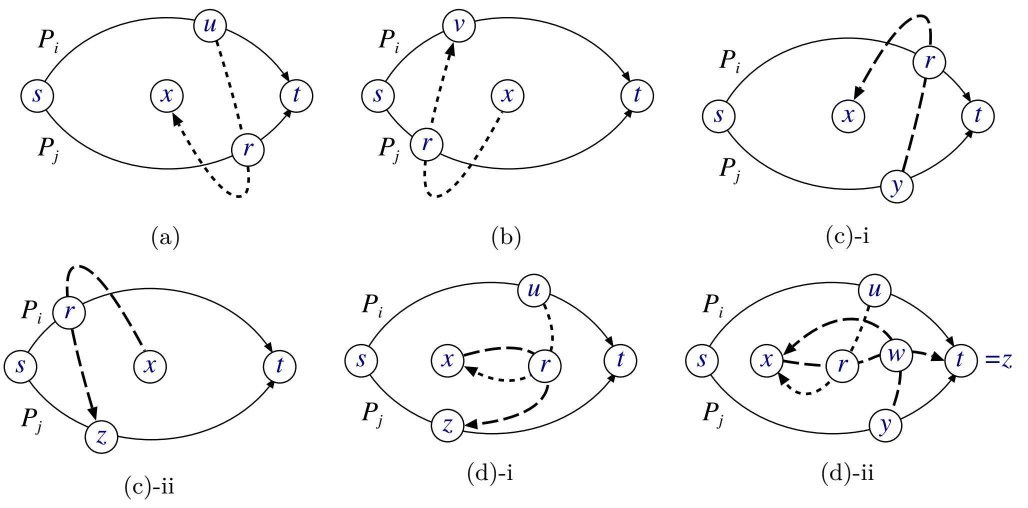

Figure 6 Paths P_i and P_j (solid arrows), Q^u and Q^v (dotted arrows), and Q^y and Q^z (dashed arrows).

▶ **Lemma 14.** *`Parallel_Decomposition` returns the maximum number k of subgraphs $G_1, \dots, G_k$ of an input graph G, sharing s and t only, such that $\tilde{G}$ is obtained by parallel composition of $\tilde{G}_1, \dots, \tilde{G}_k$.*

Proof. We begin with proving that the sets of st-paths in the returned subgraphs $G_1, \dots, G_k$ induce the chained components. To this end, we prove the following claims.

▷ Claim. Every st-path in G is contained in exactly one of the returned subgraphs.

Proof. By Lemma 10 and the property of `Parallel_Decomposition` that the vertex sets created in Step 2 are not divided in the subsequent steps, every st-path in G is contained at least one of the returned subgraphs. Since the returned subgraphs share s and t only by definition, the claim holds. ◁

▷ Claim. Any two chained st-paths are contained in one of the returned subgraphs.

Proof. By Lemma 10, any st-path P not in $\{P_1, \dots, P_\ell\}$ intersects with some P_i, and the vertices of P are added to V_i in Step 2. So it suffices to show that any chained P_i and P_j are contained in one of the returned subgraphs. Consider a chain from P_i to P_j in the route intersection graph, and suppose that the chain consists of c st-paths $Q_1, \dots, Q_c$. For each $1 \le h \le c$, if the hth st-path Q_h in the chain is in $\{P_1, \dots, P_\ell\}$, then let Q'_h be this hth path Q_h. Otherwise, let Q'_h be an st-path in $\{P_1, \dots, P_\ell\}$ adjacent to the hth st-path Q_h in the chain. Note $Q'_1 = P_i$ and $Q'_c = P_j$. For each $1 \le h < c$, the distance between Q'_h and Q'_{h+1} is at most 3, because Q_h and Q_{h+1} are adjacent, and the distances between Q_h and Q'_h and between Q_{h+1} and Q'_{h+1} are both at most 1. Moreover, the distance between Q'_h and Q'_{h+1} is at least 2, because P'_h and P'_{h+1} are in the set $\{P_1, \dots, P_\ell\}$ of disjoint st-paths, and hence at a distance more than 1. By Lemmas 11 and 12, therefore, the vertex sets of Q'_h and Q'_{h+1} are merged in Step 3. This means that the vertex sets of $Q'_1 = P_i$ and $Q'_c = P_j$ are merged in Step 3 as well. ◁

▷ Claim. Any two st-paths contained in one of the returned subgraphs are chained.

Proof. By Lemma 13, when two vertex sets are merged in Step 3, there are two st-paths at a distance 2 or 3, whose vertex sets are included in each of the two merged sets. This means that in Step 4, any two st-paths in one of the returned subgraphs are within a finite distance. $\triangleleft$

By the above claims, the sets of st-paths in $G_1, \ldots, G_k$ induce the chained components. Combined with Lemma 9, we have the lemma. ◀

▶ **Lemma 15.** *`Parallel_Decomposition` runs in $O(nm^2)$ time for an input graph G with n vertices and m edges.*

Proof. We prove a (naive) implementation of `Parallel_Decomposition` runs in polynomial time. For Step 1, we can find a maximal number ℓ of disjoint st-paths by $\ell + 1$ iterations of DFS on G with removal of the internal vertices (or possibly the single edge (s, t)) of an st-path found in each DFS. Step 1 thus finishes in $O(\ell m) = O(m^2)$ steps.

For Step 2, we can obtain the set of vertices satisfying Condition 8 as the intersection of (i) the vertices not in P_i and reachable from $P_i - t$ without passing through the vertices of P_i, and (ii) the vertices not in P_i from which $P_i - s$ are reachable without passing through the vertices of P_i. These vertex sets (i) and (ii) can be found in $O(m)$ steps using DFS with some ingenuity, such as avoiding P_i and traversing edges in the reverse direction for (ii). Merging the vertices of P_i and the intersection of the sets (i) and (ii) for each $1 \leq i \leq \ell$, Step 2 finishes in $O(\ell m) = O(m^2)$ steps.

An implementation of Step 3 is to iterate the process that we find a combination of a vertex $x \notin \{s, t\}$ and $1 \leq i < j \leq \ell$ such that $x \in V_i \cap V_j$, in $O(\ell n)$ steps, and merge V_i and V_j in $O(n)$ steps if such a combination is found. Because at most ℓ iterations of this process are enough, Step 3 finishes in $O((\ell n + n)\ell) = O(nm^2)$ steps.

Putting together, `Parallel_Decomposition` runs in $O(nm^2)$ steps. ◀

3.2.3 Implementation for Linear Time Parallel Decomposition

We describe intuitively (not precisely) the idea of a linear time implementation of `Parallel_Decomposition` using the characterization of Theorem 4.

The original Step 1, which finds a maximal number ℓ of disjoint st-paths $P_1, \ldots, P_\ell$, can be implemented as iterations of DFS on the input graph G that starts at source s and ends at sink t. By avoiding previously visited vertices and single edges (s, t) in each DFS, we can find desired paths in linear time.

For the original Step 2, which is, for each $1 \leq i \leq \ell$, adding all vertices x satisfying Condition 8 to the vertex set of P_i (the resulting vertex set is denoted by V_i), we define the following sets of vertices of G: S_i and T_i are the sets of vertices x not in P_i such that there exists an sx- and xt-path disjoint with P_i, respectively. In addition, O'_i and I'_i are the sets of vertices x not in P_i such that there exist an internal vertex u of P_i and a ux- and xu-path disjoint with P_i, respectively. The set of vertices satisfying Condition 8 is obtained as $X_i = (O'_i \cap I'_i) \cup (O'_i \cap T_i) \cup (I'_i \cap S_i)$. Each of the sets O'_i, I'_i, S_i, and T_i can be found in linear time using DFS with some ingenuity, such as, traversing edges in the reverse direction for I'_i and T_i, which we call *reverse DFS*, and avoiding vertices of P_i. However, it possibly takes a super-linear $\Theta(\ell m)$ time to find these sets for all $1 \leq i \leq \ell$ for graphs with m edges and $\ell = \omega(1)$. To reduce this running time, we modify the original Steps 2 and 3.

In the original Step 3, two sets V_i and V_j are merged if they share a vertex neither s nor t, i.e., $V_i \cap V_j \neq \{s,t\}$.[3] Merging the sets is necessary to complete the parallel decomposition. However, this process is not necessary to check if the route-induced subgraph $\tilde{G}$ of G is series-parallel, because if $V_i \cap V_j \neq \{s,t\}$, then there is an st-embedding of the Wheatstone network into G (Lemmas 19 and 17), and hence $\tilde{G}$ is not series-parallel by Theorems 2 and 4. In this case, therefore, we quit our algorithm with the return value "No" (meaning $\tilde{G}$ is not series-parallel). Otherwise, i.e., if $V_i \cap V_j = \{s,t\}$ for all $1 \leq i < j \leq \ell$, then we return the subgraph G_i induced by V_i for each $1 \leq i \leq \ell$. Note that not all st-embeddings of the Wheatstone network can be detected in this way; each of the returned subgraphs may have an st-embedding of the Wheatstone network.

To implement the modified Steps 2 and 3 in linear time, we find $S = \bigcap_{i=1}^{\ell} S_i$ and $T = \bigcap_{i=1}^{\ell} T_i$ using (reverse) DFS avoiding vertices of $P_1, \ldots, P_\ell$. Then, for $i = 1$ to ℓ, we find $O_i \subseteq O'_i$ by DFS, starting at internal vertices of P_i in turn and avoiding vertices in P_i and in O_j with each $j < i$, as well as vertices already found as elements of O_i. During DFS for O_i, if we reach an internal vertex of P_j with $j < i$, then $V_i \cap V_j \neq \{s,t\}$, and therefore we quit with "No". In addition, if we reach a vertex $O_j \cap T$ with $j < i$, then $V_i \cap V_j \neq \{s,t\}$, and therefore we quit with "No". We find $I_i \subseteq I'_i$ similarly (using reverse DFS). In this way, we can actually obtain the set X_i of vertices satisfying Condition 8 as $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ (Lemma 18).

The following is a high level pseudocode of the implementation, called `Fast_Parallel_Decomposition`.

Algorithm `Fast_Parallel_Decomposition`(G, s, t)

Input A directed two-terminal graph G with source s and sink t.

Output Either the maximum number k of two-terminal subgraphs $G_1, \ldots, G_k$ of G, sharing s and t only, such that $\tilde{G}$ is obtained by parallel composition of $\tilde{G}_1, \ldots, \tilde{G}_k$, or "No" meaning $\tilde{G}$ is not series-parallel.

```
1. Set $i = 1$ and suppose that there are $d$ edges leaving $s$, denoted by $(s,u_1), \ldots, (s,u_d)$.
   For $j = 1$ to $d$, perform the following.
   a. If $u_j = t$, then we define $P_i$ as the $st$-path consisting of the single edge $(s,u_j)$ and
      increment $i$ by 1.
   b. If $u_j \neq t$, then perform DFS starting at $s$, traversing the edge $(s,u_j)$ first, and avoiding
      vertices visited by previous DFS for smaller $j$. In the current DFS, if we reach a vertex
      incident to an edge entering $t$, then we define $P_i$ as the $st$-path visited by the DFS.
      We then quit the DFS and increment $i$ by 1. If we backtrack to $s$ with no $st$-path
      found, then we just quit the DFS.
2. Suppose that we have $\ell$ $st$-paths $P_1, \ldots, P_\ell$. For each $1 \leq i \leq \ell$, let $U_i$ be the set of
   internal vertices of $P_i$, In addition, let $U_i^{\mathrm{o}}$ and $U_i^{\mathrm{i}}$ be the set of vertices that are not in $P_i$
   and incident to an edge leaving and entering a vertex in $U_i$, respectively.
3. Find the set $S$ of vertices $x$, excluding $s$, such that there exists an $sx$-path disjoint with
   $P_1, \ldots, P_\ell$, by performing DFS starting at $s$ and avoiding vertices in $\bigcup_{i=1}^{\ell} U_i \cup \{t\}$.
4. Find the set $T$ of vertices $x$, excluding $t$, such that there exists an $xt$-path disjoint with
   $P_1, \ldots, P_\ell$, by performing *reverse DFS* (i.e., DFS traversing edges in the reverse direction)
   starting at $t$ and avoiding vertices in $\bigcup_{i=1}^{\ell} U_i \cup \{s\}$.
5. For $i = 1$ to $\ell$, perform the following.
```

[3] Note that both V_i and V_j contain s and t.

   a. Find the set O_i of the vertices, visited by DFS starting at every vertex v in U_i^{o} and avoiding vertices in P_i or in $\bigcup_{j<i} O_j$ and vertices already found as elements of O_i. During the DFS, we perform the following.
      i. If we reach a vertex in U_j with $j \neq i$, then return "No".
      ii. If we reach a vertex incident to an edge entering a vertex in $O_j \cap T$ for some $j < i$, then return "No".
   b. Find the set I_i of the vertices, visited by reverse DFS starting at every vertex v in U_i^{i} and avoiding vertices in P_i or in $\bigcup_{j<i} I_j$ and vertices already found as elements of I_i. During the DFS, we perform the following.
      i. If we reach a vertex in U_j with $j \neq i$, then return "No".
      ii. If we reach a vertex incident to an edge leaving a vertex in $I_j \cap S$ for some $j < i$, then return "No".

6. For each $1 \leq i \leq \ell$, define $V_i = \{s,t\} \cup U_i \cup (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ and return the graph induced by V_i as G_i.

We prove the correctness of `Fast_Parallel_Decomposition` in Lemmas 16–20 below.

▶ **Lemma 16.** *The paths $P_1, \ldots, P_\ell$ found in Step 1 are the maximal number ℓ of disjoint st-paths.*

Proof. In the DFS starting with the edge (s, u_j) in Step 1, when we reach a vertex x incident to an edge (x,t), we define P_i as the ith st-path found. At this point, there is no st-path that contains a vertex visited by this DFS before we reach x and is disjoint with previously found st-paths $P_1, \ldots, P_{i-1}$ (for otherwise, we should have found another vertex x' incident to an edge (x',t) before we reach x). This means that the number ℓ of st-paths is maximal. Because the DFS starting with the edge (s,u_j) avoids vertices visited by previous DFS, the found st-paths $P_1, \ldots, P_\ell$ are disjoint. We thus have the lemma. ◀

▶ **Lemma 17.** *If `Fast_Parallel_Decomposition` returns "No", then there exists an st-embedding of the Wheatstone network into G.*

Proof. There are four cases that "No" is returned. If "No" is returned in Step 5(a)i or 5(b)i, then it is implied that there is a path, denoted by Q, from a vertex in U_i to a vertex in U_j or from a vertex in U_j to a vertex in U_i, containing neither s nor t. Since U_i and U_j are the sets of internal vertices of disjoint st-paths P_i and P_j by Lemma 16, in either case, P_i, P_j, and Q constitute an st-embedding of the Wheatstone network (Fig. 7(a)).

Suppose that "No" is returned in Step 5(a)ii due to a vertex $x \in O_i$ incident to an edge entering a vertex $y \in O_j \cap T$ for some $j < i$. By the conditions on x and y, there exist a path Q_i^y from a vertex $u \in U_i$ to y via vertices in O_i, and a path Q_j^t from a vertex $w \in U_j$ to t via vertices in O_j (including y) and in T. We observe the following.

- Q_i^y and P_i are disjoint, because O_i is found by DFS avoiding vertices in P_i.
- Q_j^t and P_i are disjoint and share only one vertex t. For otherwise, some vertex z in Q_j^t is contained in U_i. The vertex z is not contained in T, because T is found by DFS avoiding vertices in $U_i \cup \{s\}$. The vertex z is not contained in O_j either, because the algorithm should have quit at Step 5(a)i if a vertex in U_i such as z was visited by DFS for finding O_j. Therefore, there exists no such vertex z.

Let r be the vertex of Q_j^t appearing first on Q_i^y. By the observations above, P_i, the ur-subpath of Q_i^y, and the st-path obtained by concatenating the sw-subpath in P_j and Q_j^t constitute an st-embedding of the Wheatstone network (Fig. 7(b)).

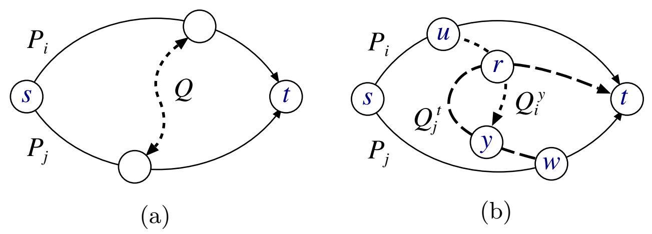

Figure 7 Paths constituting an st-embedding of the Wheatstone network.

The case that "No" is returned in Step 5(b)ii can be reduced to the case for Step 5(a)ii with the paths in the reverse direction and with the exchanged roles of O_i and I_i, and T and S. ◀

▶ **Lemma 18.** *If* `Fast_Parallel_Decomposition` *does not return "No", then the set* $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ *in Step 6 equals the set of vertices satisfying Condition 8.*

Proof. Let S_i, T_i, O'_i, and S'_i be the sets of vertices defined as follows:

$$S_i = \{x \mid x \text{ is not in } P_i\text{, and there exists an } sx\text{-path disjoint with } P_i\}$$
$$T_i = \{x \mid x \text{ is not in } P_i\text{, and there exists an } xt\text{-path disjoint with } P_i\}$$
$$O'_i = \{x \mid x \text{ is not in } P_i\text{, and there exist a vertex } u \in U_i \text{ and a } ux\text{-path disjoint with } P_i\}$$
$$I'_i = \{x \mid x \text{ is not in } P_i\text{, and there exist a vertex } v \in U_i \text{ and an } xv\text{-path disjoint with } P_i\}$$

By these definitions, the set of vertices x satisfying Condition 8 is $X_i = (O'_i \cap I'_i) \cup (O'_i \cap T_i) \cup (I'_i \cap S_i)$. Because S, T, O_i, and I_i found in the algorithm are obviously subsets of S_i, T_i, O'_i, and I'_i, respectively, it follows that $X_i \supseteq (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$. We prove $X_i \subseteq (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ by observing that if there is a vertex in X_i but not in $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$, then the algorithm returns "No".

Suppose $x \in X_i \setminus ((O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S))$. Then, x is contained in at least one of the sets $O'_i \setminus O_i$, $I'_i \setminus I_i$, $T_i \setminus T$, and $S_i \setminus S$.

1. If $x \in O'_i \setminus O_i$, then DFS for finding O_i either quits before visiting x at Step 5(a)i or 5(a)ii, or does not visit x because of $x \in O_j$ for some $j < i$. In the former possibility, we are done. In the latter possibility, $x \in (O'_i \setminus O_i) \cap O_j$, the vertex x is contained in I'_i or T_i.
   a. If $x \in (O'_i \setminus O_i) \cap O_j \cap I'_i$, then it is implied that there is a path from a vertex in U_j to a vertex in U_i via vertices in $O_j \cup I'_i$, and hence the algorithm quits at Step 5(a)i during DFS for O_h with minimum $h \leq j$ such that $O'_h \cap U_i \neq \emptyset$.
   b. If the vertex $x \in (O'_i \setminus O_i) \cap O_j$ is contained in T_i, then either $x \in T$ or $x \in T_i \setminus T$.
      i. If $x \in (O'_i \setminus O_i) \cap O_j \cap T$, then there exists a path from a vertex v in U_i^{o} (defined in Step 2) to x. This path is obtained by concatenating $i - j + 1$ (possibly empty) paths: Q_i from v to a vertex y_i via vertices in O_i, Q_h from y_{h+1} to a vertex y_h via vertices in O_h for each $j < h < i$, and Q_j from y_{j+1} to x via vertices in O_j. Note that all vertices in Q_j are also contained in T. Therefore, the algorithm quits at Step 5(a)ii during DFS for O_h with the minimum h $(j < h \leq i)$ such that Q_h is not empty.
      ii. If the vertex $x \in (O'_i \setminus O_i) \cap O_j$ is contained in $T_i \setminus T$, then it is implied that there exists a path from a vertex in U_i to a vertex in U_h for some $h \neq i$. For the minimum such h, the algorithm quits at Step 5(a)i during DFS for O_i if $i < h$, or at Step 5(b)i during DFS for I_h if $h < i$.

2. If $x \in T_i \setminus T$, then $x \in O'_i$. If $x \in O'_i \setminus O_i$, then the algorithm returns "No" as proved in the case 1. If $x \in (T_i \setminus T) \cap O_i$, then it is implied that there exists a path from a vertex in U_i to a vertex in U_j for some $j \neq i$, and hence the algorithm quits as in the case 1(b)ii.
3. If $x \in I'_i \setminus I_i$ or $x \in S_i \setminus S$, then we can prove that the algorithm returns "No" as in the case 1 or 2, respectively, with the paths in the reverse direction and with the exchanged roles of $O_i^{(\prime)}$ and $I_i^{(\prime)}$, and $T_{(i)}$ and $S_{(i)}$.

We thus conclude that $X_i = (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ in Step 6. ◀

▶ **Lemma 19.** *If `Fast_Parallel_Decomposition` does not return "No", then $V_i \cap V_j = \{s, t\}$ for V_i and V_j with any $i \neq j$ in Step 6.*

Proof. We prove that if there exists a vertex in $V_i \cap V_j$ but not in $\{s, t\}$ for some $i \neq j$, then the algorithm returns "No". Suppose that $x \in (V_i \cap V_j) \setminus \{s, t\}$ for some $i > j$. Then, x is contained in one of the sets $U_i \cap X_j$, $U_j \cap X_i$, and $X_i \cap X_j$, where $X_i = (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ and $X_j = (O_j \cap I_j) \cup (O_j \cap T) \cup (I_j \cap S)$.

If $x \in U_i \cap X_j$, then because neither T nor S contains any vertex in U_i, it follows that $x \in U_i \cap O_j \cap I_j$. Therefore, the algorithm quits at Step 5(a)i during DFS for O_j. Similarly, if $x \in U_j \cap X_i$, then $x \in U_j \cap O_i \cap I_i$. Note that this also implies $U_i \cap O_j \neq \emptyset$ and $U_i \cap I_j \neq \emptyset$. Therefore, the algorithm quits at Step 5(a)i during DFS for O_j.

Suppose $x \in X_i \cap X_j$. If $x \in O_i \cap I_j$ or $x \in I_i \cap O_j$, then it is implied that there is a path from a vertex in U_i to a vertex in U_j via vertices in I_j, or from a vertex in U_j to a vertex in U_i via vertices in O_j. Therefore, the algorithm quits at Step 5(a)i or at Step 5(b)i during DFS for O_j or I_j. The remaining possibilities are $x \in O_i \cap O_j \cap T$ and $x \in I_i \cap I_j \cap S$, by which the algorithm quits at Step 5(a)ii or 5(b)ii during DFS for O_i or I_i. ◀

▶ **Lemma 20.** *`Fast_Parallel_Decomposition` returns either desired graphs or "No" meaning $\tilde{G}$ is series-parallel in $O(m)$ time for an input graph G with m edges.*

Proof. If `Fast_Parallel_Decomposition` returns "No", then $\tilde{G}$ is not series-parallel by Lemma 17 and Theorems 2 and 4. Otherwise, by Lemmas 16, 18 and 19, $V_1, \ldots, V_\ell$ defined in Step 6 are exactly the sets obtained in Step 2 of `Parallel_Decomposition`, and $V_i \cap V_j = \{s, t\}$ for any $i \neq j$. Therefore, by Lemma 14, the desired subgraphs are returned. `Fast_Parallel_Decomposition` runs in $O(m)$ time, because each edge is searched at most constant times. ◀

3.3 Main Procedure

The following is a high level pseudocode of main procedure, called `SP_Test`.

```
Algorithm SP_Test($G, s, t$)
Input  A directed two-terminal graph $G$ with source $s$ and sink $t$.
Output "Yes" if the route-induced subgraph $\tilde{G}$ of $G$ is series-parallel, "No" otherwise.

1. If $\tilde{G}$ is a single edge $(s, t)$, then return "Yes".
2. Perform Series_Decomposition($G, s, t$).
3. For each subgraph $G'$ and its source $s'$ and sink $t'$ returned by Series_Decomposition($G$, $s, t$), perform the following.
   a. Perform Fast_Parallel_Decomposition($G', s', t'$). If it returns "No" or a single subgraph whose route-induced subgraph is not a single edge $(s', t')$, then return "No".
   b. For each subgraph $G''$ and its source $s''$ and sink $t''$ returned by Fast_Parallel_Decomposition($G', s', t'$), perform SP_Test($G'', s'', t''$) recursively.
```

4. If all executions of `SP_Test` in Step 3 return “Yes”, then return “Yes”. Otherwise, return “No”.

▶ **Theorem 21.** *`SP_Test` correctly decides if the route-induced subgraph of an input graph with m edges is series-parallel in $O(m^2)$ steps.*

Proof. `SP_Test` makes decision depending on whether or not an input graph G is decomposed into subgraphs whose route-induced graphs consist only of a single edge by `Series_Decomposition` and `Fast_Parallel_Decomposition`. Correctness of these decomposition algorithms are proved in Lemmas 6 and 20. In particular, if the route-induced subgraph $\tilde{G}$ of G is not series-parallel, then G or its subgraph appearing at some recursive step has the route-induced subgraph obtained by neither series nor parallel composition. Such a graph may be either input to `SP_Test`, which is possibly a recursive step of the parent process, or returned by `Series_Decomposition` in Step 2. In either case, `Fast_Parallel_Decomposition` in Step 3 receives a graph whose route-induced subgraph neither is a single edge nor can be parallel decomposed into smaller graphs, and therefore, returns “No” or a single subgraph whose route-induced subgraph is not a single edge. Since `SP_Test` returns “No” for such a case, it makes decision correctly.

We analyze the time complexity of `SP_Test`. We can decide that the route-induced subgraph $\tilde{G}$ is a single edge (s, t) by checking if G has an edge (s, t) and no st-paths avoiding the edge (s, t), using DFS. Combined with Lemma 7, we can perform Steps 1 and 2 in $O(m)$ steps.

Suppose that `Series_Decomposition` in Step 2 returns graphs $G_1, \ldots, G_k$ such that for each $1 \leq i \leq k$, G_i has m_i edges. Since G_i and $\bigcup_{j<i} G_j$ share only one vertex by Lemma 6, it follows that $\sum_{i=1}^{k} m_i \leq m$. By Lemma 20, therefore, Step 3 finishes in $\sum_{i=1}^{k} O(m_i) = O(m)$ steps. Since subgraphs returned by `Fast_Parallel_Decomposition` are edge-disjoint by Lemma 20, the sum of the numbers of edges of all the subgraphs returned by all executions of `Fast_Parallel_Decomposition` in Step 3 is at most m. This means that the total number of recursive executions of `SP_Test` is at most the number of vertices of a tree with m leaves. Thus, `SP_Test` runs in $O(m^2)$ steps. ◀

We observe two remarks on `SP_Test`.

▶ Remark 22. If we use `Parallel_Decomposition` instead of `Fast_Parallel_Decomposition` in `SP_Test`, then by Lemma 15 and a slightly modified proof of Theorem 21, we obtain an $O(nm^3)$ time algorithm based only on the characterization of Theorem 2.

▶ Remark 23. If we modify `SP_Test` so that we mark the single edge in Step 1, then after all recursive executions of `SP_Test` finish with “Yes”, we can obtain the series-parallel route-induced subgraph of an input graph as the subgraph induced by all the marked edges.

4 Conclusion

In this paper, we presented an $O(m^2)$ time algorithm for deciding if, for a given directed two-terminal graph with m edges, its route-induced subgraph is series-parallel. On the basis of the characterization proved in [8], our algorithm decides if the given graph does not admit Braess’s paradox for any cost functions. Our approach is based on a simple implementation of the characterization of [8]. Since this implementation runs in polynomial time, we disproved a conjecture in [7] that another characterization in terms of the input graph (not of the route-induced subgraph) would be necessary to design a polynomial time algorithm. The faster $O(m^2)$ running time is achieved by speeding up the simple implementation using another characterization proved in [8, 6] that the Wheatstone network is embedded in the given graph. The proposed algorithm is faster than the previous $O(nm^2)$ time algorithm presented in [7], where n is the number of vertices of the given graph. Combined with the technique of [9], the proposed algorithm can also be used to design a faster $O(km^2)$ time algorithm for the k-commodity case, which solves a question posed in [9] by improving the $O(knm^2)$ time algorithm presented in [9]. As future work, it would be interesting to design an even faster algorithm, such as a linear time algorithm.

References

1 Stephen Alstrup, Dov Harel, Peter W. Lauridsen, and Mikkel Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999. doi:10.1137/S0097539797317263.

2 Dietrich Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968. doi:10.1007/BF01918335.

3 Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a paradox of traffic planning. *Transportation Science*, 39(4):446–450, 2005. doi:10.1287/trsc.1050.0127.

4 Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. doi:10.1137/070693217.

5 Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, and Jeffery R. Westbrook. Corrigendum: A new, simpler linear-time dominators algorithm. *ACM Transactions on Programming Languages and Systems*, 27(3):383–387, 2005. doi:10.1145/1065887.1065888.

6 Pietro Cenciarelli, Daniele Gorla, and Ivano Salvo. Inefficiencies in network models: A graph-theoretic perspective. *Information Processing Letters*, 131:44–50, 2018. doi:10.1016/j.ipl.2017.10.008.

7 Pietro Cenciarelli, Daniele Gorla, and Ivano Salvo. A polynomial-time algorithm for detecting the possibility of Braess paradox in directed graphs. *Algorithmica*, 81:1535–1560, 2019. doi:10.1007/s00453-018-0486-6.

8 Xujin Chen, Zhuo Diao, and Xiaodong Hu. Network characterizations for excluding Braess's paradox. *Theory of Computing Systems*, 59:747–780, 2016. doi:10.1007/s00224-016-9710-4.

9 Dario Fiorenza, Daniele Gorla, and Ivano Salvo. Polynomial recognition of vulnerable multi-commodities. *Information Processing Letters*, 179:106282, 2023. doi:10.1016/j.ipl.2022.106282.

10 Igal Milchtaich. Network topology and the efficiency of equilibrium. *Games and Economic Behavior*, 57(2):321–346, 2006. doi:10.1016/j.geb.2005.09.005.

11 Tim Roughgarden. Designing networks for selfish users is hard. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 472–481, 2001. doi:10.1109/SFCS.2001.959923.

12 Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

13 Tim Roughgarden. On the severity of Braess's Paradox: Designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922–953, 2006. doi:10.1016/j.jcss.2005.05.009.

14 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

15 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1137/0211023.