
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 465–488 (2026)
DOI: 10.7155/jgaa.v30i1.3025

On Existence of Must-Include Paths and Cycles in Undirected Graphs

Yefim Dinitz *Solomon Eyal Shimony*

Stein Faculty of Computer and Information Science, Ben-Gurion University of the Negev

Submitted: April 2023 Accepted: August 2026 Published: November 2026

Article type: Regular Paper Communicated by: G. Di Battista

Abstract. We study the following two closely related existence problems for an undirected graph $G = (V, E)$. Path Existence Problem (PEP): given graph vertices s, t, w_1, w_2, does there exist a simple path from s to t containing w_1 and w_2? Cycle Existence Problem (CEP): does there exist a simple cycle containing three given graph elements (vertices or edges)? We reduce the PEP to the CEP, and the general graph case of the CEP to that of a triconnected graph. We show that two simple necessary conditions for the existence of a simple cycle containing *three given edges* in a triconnected graph are sufficient (easy solutions to all other cases are known). In addition, we develop linear-time algorithms to solve the PEP and the CEP, which provide useful admissible heuristics in combinatorial search for longest constrained paths.

Keywords: Biconnected graphs, triconnected graphs, SPQR trees, must-include simple paths and cycles

1 Introduction

We define and examine the problem of determining whether a simple path from a source to a target through two other given vertices exists in an undirected graph. This problem, as well as its extension to finding all vertex pairs where such paths do not exist, arose from the need to compute a strong admissible heuristic in a combinatorial search for longest constrained paths [5]. The search for simple paths that include a given set of vertices is also of interest in communication networks [26]. In addition, this problem, as well as the similar question on a must-include simple cycle, are of theoretical interest.

E-mail addresses: dinitz@bgu.ac.il (Yefim Dinitz) shimony@cs.bgu.ac.il (Solomon Eyal Shimony)


This work is licensed under the terms of the CC-BY license.

1.1 Problem statements

Formally, we examine the following "must-include" problems on simple paths and cycles in connected undirected graphs, starting with the following problem (we use the standard notation: xy-path, meaning a path from x to y):

Definition 1 (Path Existence Problem (PEP))**.** *Given an undirected graph $G = (V, E)$ and distinct vertices $s, t, w_1, w_2 \in V$, does there exist a simple st-path containing w_1 and w_2?*

In triconnected graphs, the PEP always has an affirmative answer [5]. A more challenging variant of this problem that is also of interest in [5] is:

Definition 2 (Exclusion-Pair Enumeration (EPE))**.** *Given an undirected graph $G = (V, E)$ and vertices $s, t \in V$, find all pairs of vertices w_1, w_2 such that there is no simple st-path containing w_1 and w_2.*

For conciseness, we call the vertices and edges of a graph its *elements*. Another problem similar to PEP is:

Definition 3 (Cycle Existence Problem (CEP))**.** *Given an undirected graph and three of its elements, does there exist a simple cycle containing all these elements?*

Given a graph G with distinguished vertices s and t (either with or without edge (s, t)), we denote $G \cup (s, t)$ by G^+.[1] Note that PEP(G, s, t, w_1, w_2) is equivalent to CEP$(G^+, (s, t), w_1, w_2)$: their solutions differ just by adding / removing the edge (s, t). Due to this equivalence, any result for the CEP straightforwardly implies a similar result for the PEP, including algorithms and their run-times. Hence, in what follows, we provide theoretical results and algorithmic solutions only for the CEP, thus immediately covering the PEP as well.

For the CEP on a *triconnected* graph, the answer is known (as folklore), except for the case where three edges are given: in all but the latter case, such a cycle always exists. Where all given elements are vertices, this is covered by Dirac's theorem [9], while the case where at most two given elements are edges was proved in informal discussions [19, 21]. For the case of three given edges, we show that two simple necessary conditions for the existence of a cycle containing them are sufficient.

1.2 Related Work

Here we briefly discuss prior work on the PEP and related problems. Much of the literature on existence of simple paths centers on disjoint paths with certain endpoint requirements [2, 10]. Out of these, the most relevant to the PEP is the following: given k pairs of vertices (v_i, u_i), $1 \leq i \leq k$, find k (edge-wise or vertex-wise) disjoint $v_i u_i$-paths. These problems are NP-hard for general k [20], but for constant k, Robertson and Seymour [24] provide an $O(|V|^3)$ solution to the problem of k vertex-disjoint paths. However, as the authors admit, the algorithm stated therein is impractical. Given an algorithm for computing such k disjoint paths, one can solve the PEP by finding a path from s to w_1, a path from w_2 to t, and a path from a neighbor of w_1 to a neighbor of w_2, all disjoint. Merging these paths is a solution to the PEP. Making this scheme complete may require iterating

[1] Henceforth, we omit the braces { } for singleton sets in expressions involving set operators such as $\setminus$ and $\cup$, when unambiguous. Additionally, we apply set operators to a graph as shorthand for the operation applied to its vertex set or edge set, as appropriate from the context. For example, for a graph $G = (V, E)$, we denote by $G \cup (s, t)$ the graph $(V, E \cup \{(s, t)\})$.

over all pairs of neighbors, and also examining the case where the paths are s to w_2 and t to w_1. Still, the runtime is polynomial. Finding k "must-include" disjoint trees, which is more general than the k disjoint paths problem, can be done in a linear time for fixed k in graphs restricted to be planar [23].

The direct generalization of the PEP, where the must-include set consists of $k > 2$ vertices, has also been examined. For unbounded k, it is easily shown that this problem is NP-hard, by a reduction from Hamiltonian Cycle [14]. An algorithm involving small k was shown in [26], but that work does not provide explicit runtime guarantees and their algorithm does not guarantee finding a path if one exists.

Our PEP seemingly falls under the conditions for the existence of an integral multi-commodity flow in a network with four terminals stated by Seymour [25, statement (1.5)]. However, this result is not applicable to the PEP since: 1) the achieved paths are essentially only edge-disjoint while we require vertex-disjoint paths, and 2) it states only very special sufficient conditions while we need necessary and sufficient conditions.

1.3 Structure and Contributions of this Paper

We begin with a background on graph connectivity (Section 2): Menger's theorem, the decompositions of a graph into its 2- and 3-connected components, and the respective properties of simple paths within them; in passing, we show a simple reduction of our problems to the case of a biconnected graph. In addition, we focus in detail on SPQR trees, which are heavily used in this paper, and establish some new properties therein. In particular, we describe in detail the special case of an SPQR tree of a graph obtained from a triconnected graph by removing one of its edges (Section 2.2).

For CEP, we provide a structural solution to the case of three edges in *triconnected* graphs: simple necessary and sufficient conditions for cycle existence (Section 3). We then provide necessary and sufficient conditions for the existence of a cycle as in CEP (and thus of the path as in PEP) in general *biconnected* undirected graphs (Section 4). This is done by demonstrating that the CEP reduces to the existence of a cycle in an appropriately defined *central component* of the SPQR tree of the graph. The difficult subproblem here is in components of type R, where we use the above-mentioned conditions for CEP in triconnected graphs.

Finally, we develop efficient algorithms for both CEP and EPE (Section 5). Our CEP (and thus PEP) algorithm has runtime $O(|E|)$. For the EPE in the case where *implicit* output of the enumerated pairs is allowed (for example, the admissible heuristic in [5] only needs access to various counts of such pairs, and thus can use the implicit form directly), we provide an $O(|E|)$-time algorithm as well; it becomes $O(|V|^2)$-time if *explicit* enumeration is required.

A preliminary version of this paper appears in the CoRR archives [6].

2 Background on Graph Connectivity

We begin with some background on vertex connectivity and the supporting flow algorithms; edge connectivity is also mentioned as needed. See, e.g., books [12, 16, 18] for details. In this paper, we focus on connected undirected graphs $G = (V, E)$. For any non-empty $V' \subseteq V$, the *subgraph* $G(V')$ *induced by* V' is V' together with all edges of G between the vertices in V'. A subset $V' \subseteq V$ of vertices is called a (*vertex*) *cut* if the removal of the vertices in V' makes G disconnected and if no subset of V' has this property. We call a cut V' a k-*cut* if $|V'| = k$. A graph is (vertex) k-*connected* if $|V| \geq k + 1$ and no $k - 1$ or fewer of its vertices form a cut. We use the terms *biconnected*

and *triconnected graphs* meaning 2- and 3-connected graphs, respectively. In this paper, we use the term *disjoint* w.r.t. paths to mean that they are vertex-disjoint. Two or more paths are called *internally disjoint* if they are disjoint except possibly for their end-vertices.

2.1 Basics of Connectivity

The main tool for exploring k-connected graphs is Menger's theorem:

Theorem 1 (Menger [22])**.** *In any k-connected graph $G = (V, E)$, for every two vertices $s, t \in V$, there exist k simple, internally disjoint paths between s and t.*

Additionally, for any two sets of vertices V_s, V_t in such a graph, each of cardinality at least k, there exist k simple, fully disjoint paths between V_s and V_t. (Note that V_s and V_t need not be disjoint.) Throughout this paper, we assume that *no internal vertex of those paths is in V_s, V_t*; indeed, in such a case, the extra paths' prefixes and/or suffixes can be safely truncated. For conciseness, we henceforth take paths between two objects (such as paths, cycles) to mean paths between the vertex sets of these objects. Theorem 1 also naturally generalizes to the case where either $V_s = \{s\}$, or $V_t = \{t\}$: there exist k paths as above that are disjoint except at that singleton set. We refer to the above generalization of Theorem 1 as the *extension to Menger's theorem*.

Note that the converse of Menger's theorem also holds, since for every pair of vertices s and t, no deletion of $k' < k$ vertices is capable of destroying all k internally disjoint paths between s and t. Moreover, for similar reasons, removing any r elements (vertices and/or edges) of a k-connected graph, $r < k$, results in a graph that is $(k - r)$-connected. The following known "decremental" property of k-connected graphs is used in several places in this paper.

Property 1. *Let G be a k-connected graph, and edge $(a, b) \in G$. Then, graph $G^- = G \setminus (a, b)$ is also k-connected if and only if there is no $(k-1)$-cut separating a from b in G^-.*

Indeed, according to the above observations, G^- is always $(k-1)$-connected. Assuming that there is a $(k-1)$-cut V' not separating a from b in G^- implies that V' remains a cut after adding edge (a, b) to G^-; that is, V' is a $(k-1)$-cut in G, a contradiction.

When we need to actually find the set of disjoint st-paths as in Menger's theorem, the following efficient algorithms are used. (See any book in basic graph algorithms as a source, such as [3, 12, 18].) To find a single st-path (the case $k = 1$ of Menger's theorem), efficient schemes execute a labeling algorithm (either BFS or DFS) scanning G from s up to labeling t, and then restore the path using the backward labels; this takes time $O(|E|)$.

When we need to find the set of several disjoint paths between s and t as in Menger's theorem, network flow techniques are used (see, e.g., [12, 18]). The given k-connected graph $G = (V, E)$ is turned into a (directed) flow network $N^{\mathrm{V}} = (\vec{G}^{\mathrm{V}}, s, t, c^{\mathrm{V}})$ with the flow source s and sink t. Every vertex v of G, except for s and t, is turned into two vertices v_1 and v_2 with a directed edge (v_1, v_2) of capacity $c^{\mathrm{V}}(v_1, v_2) = 1$ from v_1 to v_2 in $\vec{G}^{\mathrm{V}}$. Every (undirected) edge (u, v) of G is turned into two directed edges (u_2, v_1) and (v_2, u_1) of infinite capacity each in $\vec{G}^{\mathrm{V}}$.

The max-flow min-cut theorem [13] guarantees existence of a flow of size k in N^{V}. We find such a flow f in N^{V} by executing the Ford-Fulkerson algorithm [13]. The flow is guaranteed to be integral, that is the flow $f(e)$ in each edge e of N^{V} is either 0 or 1. The edges assigned with flow 1 constitute k disjoint paths from s to t in $\vec{G}^{\mathrm{V}}$, which naturally define k disjoint paths between s and t in G, as required.

The Ford-Fulkerson algorithm (FF) works in iterations, beginning from the zero flow. Each iteration of FF increases the flow size in N^{V} by 1; the runtime of each iteration is $O(|E|)$. That is,

for any constant k, the time of finding a flow of size k in N^{v}, and thus of finding k disjoint paths between s and t in G, is also $O(|E|)$.

Handling the extension of Menger's theorem is easy. If a set V_s is given instead of s, then, before constructing $\vec{G}^{\mathrm{v}}$, enhance G by adding an artificial source $\bar{s}$ with edges to every vertex in V_s. Likewise, add edges from the set V_t to a new vertex $\bar{t}$. After finding k disjoint paths between $\bar{s}$ and $\bar{t}$, we just truncate them by removing $\bar{s}$ and $\bar{t}$.

Biconnected components. A vertex a is called an *articulation vertex* (or *separator*) of G if $\{a\}$ is a 1-cut. We will also call a separator "separating vertex" or say that it "is separating", for convenience of writing. Removing an articulation vertex a from the graph results in two or more induced connected subgraphs $G(V_i)$, $1 \leq i \leq r$, $r \geq 2$, $\cup_i V_i = V \setminus a$. Continuing to recursively divide the induced connected subgraphs $G(V_i \cup a)$ by their separating vertices (each is a separating vertex of G as well), the final induced subgraphs are biconnected; they are called the *biconnected components*, or *blocks* of G. The tree structure defined by the above decomposition is called the *block tree* of G. The blocks and the block tree are defined canonically for any graph and can be found in a linear time $O(|E|)$.

Observe that any simple cycle in G must lie entirely in a single block; this is so because if a simple path crosses from one block to another, it must go via a separator, and then it has no way to return. Therefore, the CEP can be answered affirmatively only if all given elements are in the same block, B. In such a case, we can safely discard all other blocks of G, which can be done in $O(|E|)$ time, thus reducing the problem to the (biconnected) graph B. Due to the equivalence between the PEP and CEP as in Section 1.1, the PEP in a general graph can also be easily reduced to either a PEP or a CEP in a single block of $G^+ = G \cup (s,t)$. We thus assume henceforth that *graphs G for the CEP and G^+ for the PEP are biconnected*.

2-cuts and edge cuts. A biconnected graph can contain 2-cuts. Such a vertex pair $\{a,b\}$ that separates G is henceforth also called a *separation pair* of G. A similar, albeit more complex, sub-division of a biconnected graph G by its separation pairs into triconnected components and auxiliary structural elements is possible, see, e.g., [1]. This structure, named the SPQR tree of G, is described in Section 2.2.

In this paper, we also use the concept of an *edge cut*, which is an inclusion-minimal set of edges E' of G such that removing the edges in E' disconnects G. Unlike vertex cuts, any edge cut E' divides G into exactly two connected induced subgraphs, called the *sides of* E'; their vertex sets V', V'' form a 2-partition of V. Note that the edges in E' and only these edges connect vertices from V' and V''. An edge cut is called a k-edge cut if $|E'| = k$. To distinguish such cuts from vertex cuts, we always say "edge cut" in full in this paper, leaving "cut" as a nickname for a vertex cut.

Property 2. *For every edge cut E' and simple cycle L, the number of edges in $L \cap E'$ is* even.

Indeed, when traversing L, each edge in E', and no other edge, switches between the sides of E'. Therefore, to complete the cyclic traversal, an even number of such switches must occur.

2.2 Triconnected Components and SPQR trees

A neat representation of the sub-division of a biconnected graph $G = (V,E)$, $|V| \geq 3$, into components arranged into a special tree structure is called the *SPQR tree* of G [1]. Such a tree is of a linear size and can be constructed in a linear time $O(|E|)$ [15]. As we use SPQR trees extensively in this paper, we now delve into their details. There exist different definitions of SPQR trees, which

vary technically; we choose the *unrooted* version of the SPQR tree [11], which we consider less technical and more intuitive in applications. We assume that there are at least three vertices in G, and thus omit the nodes of type Q [15].

The SPQR tree of G, $\mathcal{T} = \mathcal{T}(G)$, consists of *nodes* and *structural edges*. Each node X is associated with a (multi-)graph C_X called a "triconnected component" of G [17] (henceforth called simply a *component*),[2] where $C_X = (V_X, E_X)$ with $V_X \subseteq V$; each edge in $E_X \setminus E$ is called a *virtual edge*. Nodes are of types R (rigid), S (series), and P (parallel), according to the graph type of the associated component. Each structural edge (X, Y) between nodes X and Y is associated with a 2-cut $\{a, b\}$ of G and with two auxiliary virtual edges (a, b), one in C_X and one in C_Y. Thus, the degree of any node X in $\mathcal{T}$ equals the number of virtual edges in C_X. In what follows, we call any non-virtual (i.e., inherited from G) edge in a component a *real edge*.

We call a pair of 2-cuts $\{a, c\}$ and $\{b, d\}$ *crossing* if each of them disconnects the vertices of the other 2-cut (as in Figure 1 (right)). A 2-cut $\{a, b\}$ is called *proper* if there is no 2-cut that crosses it. If G is a triconnected graph or a cycle, resp., the SPQR tree of $\mathcal{T}(G)$ is trivial, consisting of a single R or S *node* X, resp., associated with the R or S *component* $C_X = G$, resp. Otherwise, there are proper 2-cuts in G, and $\mathcal{T}(G)$ is obtained via the following recursive decomposition of G into the triconnected components by its proper 2-cuts. See Figure 1 for an illustration.

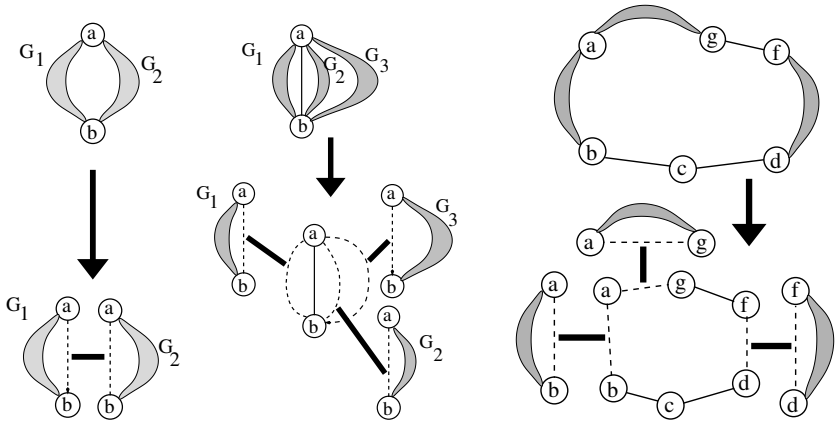


Figure 1: A proper 2-cut breaking G into exactly two parts, where $(a, b) \notin E$ (left), a P node creation (middle), and an S node in the SPQR tree (right). In order to focus on the virtual and structural edges, components and parts of G are shown instead of nodes; each structural edge (thick line) is shown as connecting the two virtual edges (dashed lines) associated with it.

Consider an arbitrary proper 2-cut $\{a, b\}$, disconnecting G into $r \geq 2$ connected subgraphs with vertex sets V_i, $1 \leq i \leq r$. We define r *parts* of G w.r.t. $\{a, b\}$: subgraphs $G(V_i \cup \{a, b\}) \setminus (a, b)$ with a *virtual edge* (a, b) added to each, denoted by G_i. For each part G_i, we distinguish the unique node Y_i of $\mathcal{T}(G_i)$ such that C_{Y_i} contains the virtual edge $e = (a, b)$. If $r = 2$ and $(a, b) \notin E$, then $\mathcal{T}(G)$ consists of $\mathcal{T}(G_1)$ and $\mathcal{T}(G_2)$ connected by a structural edge (Y_1, Y_2) associated with the cut $\{a, b\}$ and the virtual edges (a, b) in C_{Y_1} and C_{Y_2} (see Figure 1 (left) for an illustration).

[2] This is a standard historically used term, although not all these components are triconnected graphs. Another term used in the literature for a component is "skeleton"; we prefer not to use it since it could be confusing: it seems more intuitive to us to refer to the entire SPQR tree as "the skeleton of 2-connectivity of G" (cf. the skeleton of S-connectivity of a graph introduced by Dinitz and Vainshtein in [7, 8]).

Otherwise, there is a *P node* X in $\mathcal{T}(G)$, such that the associated *P component* C_X contains the vertices a and b, r virtual edges (a, b), and also the real edge (a, b) if $(a, b) \in E$. Then, $\mathcal{T}(G)$ consists of $\mathcal{T}(G_i)$, $1 \leq i \leq r$, each connected to the node X by a structural edge (X, Y_i), associated with $\{a, b\}$, the virtual edge (a, b) in C_{Y_i}, and one of the virtual edges in C_X, bijectively (see Figure 1 (middle)). Note that the SPQR tree $\mathcal{T}(G)$ resulting from the described recursive decomposition does not depend on the order of segmentation of G by its proper 2-cuts.

For a concrete illustration, see Figure 2. The gross structure of the SPQR tree of the example graph in Figure 2 (top left) is shown in Figure 2 (bottom left): the rectangles represent the nodes with structural edges between them; the components associated with the nodes are shown in Figure 2 (right).

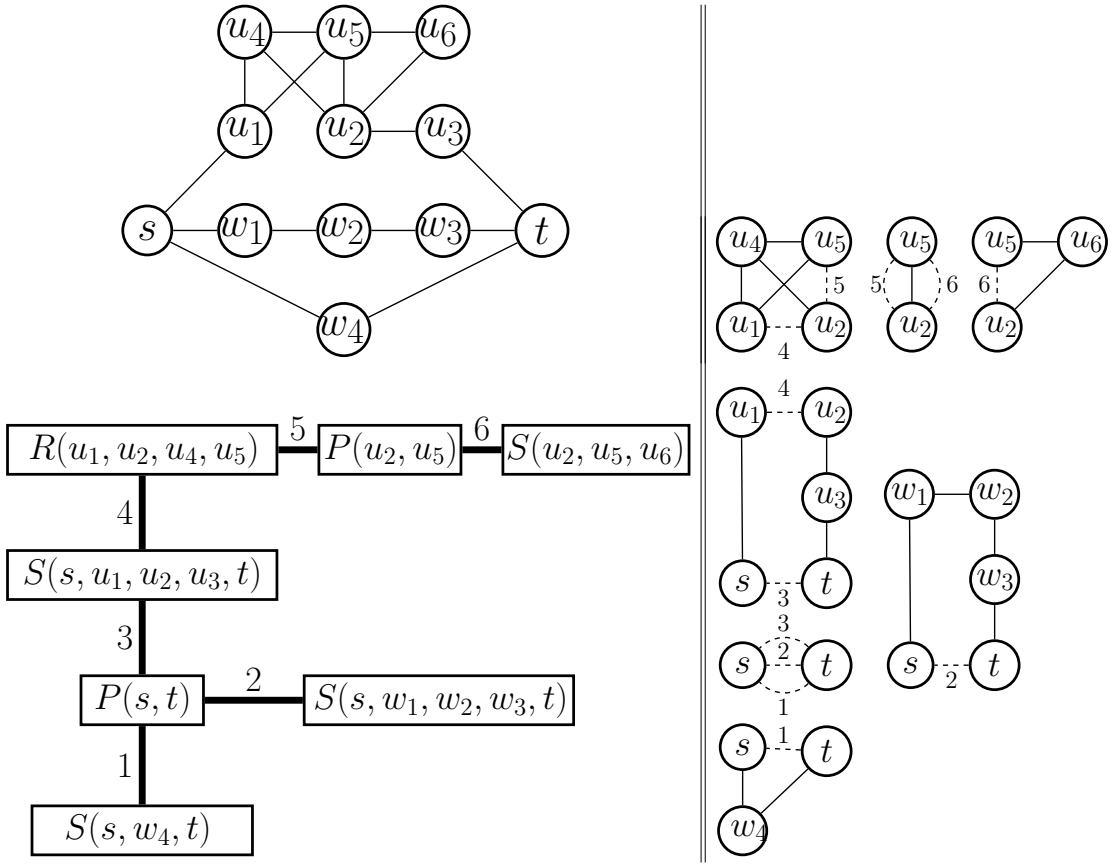


Figure 2: A biconnected graph (top left), its SPQR tree structure (bottom left), and its components (right). For better clarity, each node is labeled by its type and the list of vertices of the associated component; each structural edge (bottom left, thick line) is labeled by the same number as the two virtual edges associated with it (right, dashed lines).

The following simple observations are straightforward from the construction of $\mathcal{T}$. There are at least three edges in each P component; thus, there are no P nodes of degree 1, and the only case of a P node X of degree 2 is when there are one real and two virtual edges in C_X. No two S nodes and no two P nodes are adjacent in an SPQR tree. Any edge of G is a real edge of exactly one

component. Any vertex v of G is in at least one component, and it is in more than one component if and only if v is a member of a proper separation pair of G. In this paper, we use some known properties of SPQR trees, given here without proofs, and other properties which we establish here as lemmas, as follows.

Property 3. *For every pair of crossing 2-cuts $\{a, c\}$ and $\{b, d\}$ in G, the vertices a, b, c, d are in the same S component of $\mathcal{T}(G)$.*

Lemma 4. *A pair of edges e, e' forms a 2-edge cut in a biconnected graph G if and only if e and e' are real edges in the same S component of $\mathcal{T}(G)$.*

Proof: Let $\{e, e'\}$ be a 2-edge cut in G, with $e = (u, v)$, $e' = (u', v')$, so that u and u' are on the same side of $\{e, e'\}$. Assume initially that all four end-vertices of e, e' are distinct. Note that both $\{u, v'\}$ and $\{u', v\}$ are 2-cuts of G^-, since each one of them separates between the two other end-nodes of e, e'. By definition, these 2-cuts are crossing, so, by Property 3, all u, v, u', v' are in the same S component. Clearly, both real edges (u, v) and (u', v') are in that S component, as required.

Otherwise, w.l.o.g., u coincides with u'. If there is another vertex, z, on the same side of $\{e, e'\}$ as u, then u is an articulation vertex separating z from v, a contradiction to the biconnectivity of G. Hence, u is unique on its side of $\{e, e'\}$; thus, the only edges incident on u are e and e'. If G consists only of the triangle on u, v, v', we are done. Otherwise, $\{v, v'\}$ is a 2-cut of G. There are two cases. (1) If there is a 2-cut $\{u, w\}$ crossing $\{v, v'\}$ then the vertices u, v, v' are in the same S component, by Property 3, and we are done. (2) Otherwise, the 2-cut $\{v, v'\}$ is proper. Decomposing G by $\{v, v'\}$, as in the above definition of $\mathcal{T}(G)$, one of the parts w.r.t. $\{v, v'\}$ is the 3-cycle $L = (u, v, v')$. Consequently, this part constitutes an S node associated with the component L, which is included in $\mathcal{T}(G)$, as required.

The fact that any two real edges of any S component form a 2-edge cut of G is a known property of SPQR trees. □

Let $\varepsilon = (X, Y)$ be a structural edge in $\mathcal{T}$ and e be the virtual edge in C_X associated with ε. We denote by $B(X, e)$ the sub-tree of $\mathcal{T}$ hanging on ε from X. Consider the subgraph of G consisting of all vertices and edges of G appearing in the components associated with nodes of $B(X, e)$ as their real elements. In the literature, this subgraph is called the *expansion* of e in C_X, denoted henceforth by $Exp(C_X, e)$. We use the notation $Exp^+(C, e)$, where $e = (a, b)$, to denote $Exp(C, e)$ with added virtual edge (a, b). The following property states that the split of $\mathcal{T}$ by any of its structural edges (into two parts, obviously) induces a split of G into two parts as well.

Property 5. *For any structural edge (X, Y), $Exp(C_X, (a, b))$ and $Exp(C_Y, (a, b))$: (i) cover together the entire graph G and (ii) have no elements in common except for vertices a and b.*

Property 6. *For any component C and virtual edge e in C, the graph $Exp^+(C, e)$ is biconnected.*

As a consequence, we immediately get the following corollaries:

Corollary 7. *The graphs $Exp(C, (a, b))$, $Exp(C, (a, b))\backslash\{a\}$, and $Exp(C, (a, b))\backslash\{b\}$ are connected.*

Corollary 8. *Given any element x of $Exp(C, (a, b))$, there exists a simple ab-path in $Exp(C, (a, b))$ that contains x.*

The latter statement follows from Property 6 due to the extension of Menger's theorem applied to $\{a, b\}$ and either x, if x is a vertex, or $\{u, v\}$, if x is an edge (u, v), in $Exp^+(C_X, e)$. Note that by Property 5, the ab-path in Corollary 8 uses no element of C other than a and b. Property 6 can be further refined:

Lemma 9. *Let (X, Y) be a structural edge associated with virtual edge e in C_X. Then, graph $Exp(C_X, e)$ is biconnected unless Y is an S node.*

Proof: Let $e = (a, b)$, and let $e' = (a, b)$ be the virtual edge in C_Y associated with (X, Y). If Y is a P node, then C_Y has $r \geq 3$ edges (a, b). Each of them, except for e', gives rise to a simple ab-path in $Exp(C_X, e)$, all internally disjoint. Indeed, the real edge (a, b), if it exists, is such a path itself, and for each virtual edge $e'' \neq e'$, there is such an ab-path in $Exp(C_Y, e'') \subseteq Exp(C_X, e)$ by Corollary 7. For illustration, see Figure 2, where $X = S(s, w_4, t)$, $Y = P(s, t)$, and e and e' are labeled 1.

If Y is an R node, C_Y is triconnected. Hence, removing e' leaves two internally disjoint ab-paths in C_Y. Each such path P can be extended into a path in $Exp(C_X, e)$ by replacing every virtual edge (c, d) in P by a cd-path in $Exp(C_Y, (c, d))$, which exists by Corollary 7. For illustration, see Figure 2, where $X = S(s, u_1, u_2, u_3, t)$, $Y = R(u_1, u_2, u_4, u_5)$, and e and e' are labeled 4.

Thus, in both cases, there are at least two internally disjoint ab-paths in $Exp(C_X, e)$, which implies that there is no 1-cut separating a and b in $Exp(C_X, e)$. Since $Exp^+(C_X, e)$ is biconnected by Property 6, then, by Property 1, $Exp(C_X, e)$ is biconnected. □

In this paper, we use the following notion of *representation* of elements of G in components. Let C be an arbitrary component of $\mathcal{T}$. We say that any real element x of C represents itself in C. For any virtual edge $e = (a, b) \in C$, all elements of $Exp(C, e)$ except for a and b are represented in C by e. Thus, every element x of G is represented exactly once in C; we denote the unique object that represents it, either x itself or a virtual edge, by $r_C(x)$.

We now establish and prove the following decremental property of SPQR trees.

Lemma 10 (Decremental Lemma)**.** *Let G be a triconnected graph with edge (s, t) such that $G^- = G \setminus (s, t)$ is not triconnected. The SPQR tree $\mathcal{T}(G^-)$ consists of a single path $\Pi = (X_0, X_1, \ldots, X_k)$, $k \geq 1$, with the following properties. We denote the 2-cut associated with the structural edge (X_{i-1}, X_i) by $\{a_i, b_i\}$; we use C_i as a shorthand denoting C_{X_i}.*

- *The only virtual edges in the components of $\mathcal{T}(G^-)$ are (a_i, b_i), with a copy in each of C_{i-1} and C_i.*
- *Every 2-cut of G^- disconnects s from t; thus, neither s nor t are members of a separation pair.*
- *Vertex s is only in C_0, which is either an R component, or a 3-cycle S component (s, a_1, b_1) with real edges (s, a_1) and (s, b_1).*
- *Vertex t is only in C_k, which is either an R component, or a 3-cycle S component (a_k, b_k, t) with real edges (a_k, t) and (b_k, t).*
- *Component C_i, $1 \leq i \leq k-1$, is one of the following:*
  - *a 4-cycle S component $(a_{i-1}, a_i, b_i, b_{i-1})$ with real edges (a_{i-1}, a_i) and (b_{i-1}, b_i),*
  - *a 3-cycle S component consisting of either $(a_{i-1}, a_i, b_i = b_{i-1})$ with real edge (a_{i-1}, a_i), or $(a_{i-1} = a_i, b_i, b_{i-1})$ with real edge (b_i, b_{i-1}),*
  - *a P component on the vertices $a_{i-1} = a_i$, $b_{i-1} = b_i$ with real edge (a_i, b_i),*
  - *an R component.*

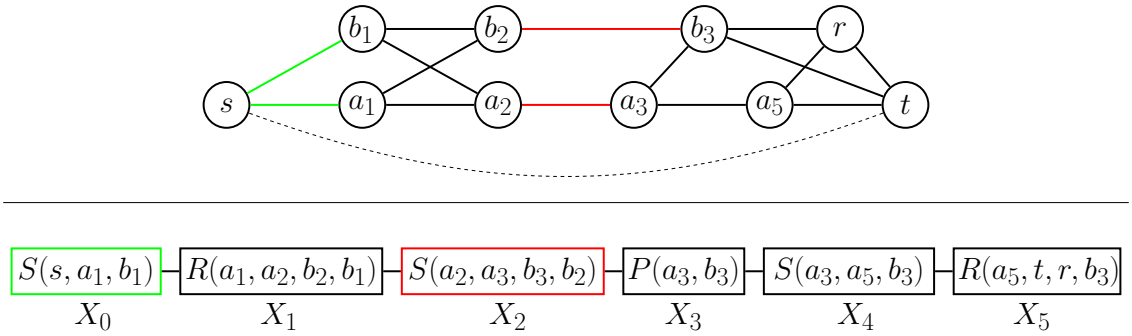


Figure 3: A triconnected graph G with edge (s, t) shown dotted. Note that $a_3 = a_4$, $b_3 = b_4 = b_5$. The SPQR tree of G^- is shown at the bottom. To illustrate Corollary 11, the two 2-edge cuts of G^- are shown in green and red (top), and the corresponding S nodes of the SPQR tree of G^- are shown with the same colors (bottom).

Proof: [3] For an illustration, see Figure 3. Since G is triconnected and G^- is not, $\mathcal{T}(G)$ is trivial: consisting of a single R node, while $\mathcal{T}(G^-)$ is non-trivial: consisting of more than one node and at least one structural edge. Recall that G^- is obtained from G by removing the edge (s, t). Therefore, assuming that a 2-cut V' of G^- does not disconnect s from t implies that V' is also a 2-cut of G, a contradiction. The following two situations are also impossible, as special cases of the above: a) either s or t (or both) are members of a separation pair, and b) either s or t (or both) lie in more than one component (since this implies (a)).

Let $\Pi = (X_0, X_1, \ldots, X_k)$, $k \geq 1$, be the path in $\mathcal{T}(G^-)$ between the two (distinct) components containing s and t, respectively. If there is a structural edge $\varepsilon \in \mathcal{T}(G^-)$ not lying on Π, then the 2-cut associated with ε does not separate s and t, a contradiction. So, $\mathcal{T}(G^-)$ coincides with Π, and the only virtual edges in the components are (a_i, b_i), with one instance residing in C_{i-1} and one in C_i.

No end-node of Π is of type P since the degree of any P node in an SPQR tree should be at least 2. Consider any internal node X_i of Π of type P. Since C_i contains exactly two virtual edges, while a P component must have at least three edges, C_i must contain the real edge (a_i, b_i).

Consider an intermediate S node X_i in Π. Clearly, all vertices $a_{i-1}, a_i, b_i, b_{i-1}$ lie on the cycle $L = C_i$. If there is another vertex v on L, then its two neighbors on L form a 2-cut of G^- separating v from the rest of G^-; thus, it does not separate s from t, a contradiction. If no two end-vertices of the virtual edges of C_i coincide, then we have the 4-cycle S component case from the statement of the lemma. If two of those vertices, w.l.o.g., b_{i-1} and b_i, coincide then we have the 3-cycle S component case therein. Now let C_0 be an S component. Note that $s \in C_0$ and that (a_1, b_1) is the only virtual edge therein. Both a_1, b_1 are distinct from s, as s cannot be a member of a separation pair of G^-. Thus, C_0 is a 3-cycle, because if it has a vertex v distinct from s, a_1, b_1 then its two neighbors form a 2-cut of G^- not separating s and t, a contradiction. The same argument holds for C_k, t, and (a_k, b_k). □

Lemmas 4 and 10 imply the following:

[3] Some of the items stated in Lemma 10 follow from a detailed examination of the incremental *InsertEdge* operation in maintaining triconnected components of a graph [1]; we show an independent proof here for better self-containment of this paper.

Corollary 11. *With the conditions and notation of Lemma 10, two edges form a 2-edge cut in G^- if and only if they are the two real edges of either a 4-cycle S component C_i, $1 \leq i \leq k-1$, or a 3-cycle S component C_0 or C_k. Any 2-edge cut of G^- separates s from t.*

2.3 PEP and CEP: Known Properties

If the graph happens to be triconnected, the answer to the PEP is always positive. That is due to the following theorem from [5]:

Theorem 2. *Let G be a triconnected graph. Then, for every $s, w_1, w_2, t \in V$, there exists a simple path in G from s to t that includes w_1 and w_2.*

For the CEP in a triconnected graph, the case of three vertices holds by Dirac's theorem [9]: in every k-connected graph, $k \geq 2$, given any set of k vertices, there exists a simple cycle containing these vertices. This theorem can be extended to any set of $m \in \{1, 2\}$ edges and $k-m$ vertices. This extension is known as "folklore" of internet discussions [21, 19]; as we could not find it formally published, we state a proof outline, as follows. For $k = 2$, given edges (u, u') and (v, v'), existence of a cycle follows immediately from the extension to Menger's theorem. Indeed, since there exist two disjoint paths from u, u' to v, v', they can be concatenated with the given edges, resulting in a cycle as required. The case where an edge and a vertex are given is similar. For $k > 2$, this is proved by induction. Assume that there exists a cycle L with the first given $k-1$ elements, including all (one or two) given edges. If the last vertex w is not already on L, there are k simple paths P_i from w to vertices in L disjoint except for at w. Since L has only $k-1$ path segments between its given $k-1$ elements, at least two of the P_i, w.l.o.g., P_1, P_2, end at distinct vertices v_1, v_2 lying in one of these segments, P. Then, the required cycle is the same as L where path P is spliced to include the detour P_1, w, P_2. (Throughout this paper, we use the term "splicing" to mean replacing a sub-path of a given path by another given path.) We thus obtain:

Theorem 3. *In a triconnected graph G, given a set S of three elements of G where S contains at most two edges, there exists a simple cycle in G that traverses all the elements of S.*

Therefore, as mentioned in the introduction, in a triconnected graph, the answer to the CEP is positive for any set of three elements in which at most two are edges.

3 Cycle Existence Problem in Triconnected Graphs

We have seen that the only nontrivial case for the CEP in triconnected graphs is when the three given elements are all edges, denoted e_1, e_2, e_3 below. Here we establish necessary and sufficient conditions for the existence of a simple cycle that contains them. Treatment of such a CEP is divided into two major cases: one where no end-vertices of the given edges are shared, and one where some or all of the given edges have common end-vertices. We begin with the latter, which is rather straightforward.

Let us first introduce some notation. For any path P, we denote by $b(P)$ its starting vertex, by $f(P)$ its ending vertex, and by $\bar{P}$ the path reversed to P. We denote by P_{xy} the path from a given collection with $b(P) = x$ and $f(P) = y$, if this does not lead to ambiguity, and by P_{yx} the path reverse to P_{xy}. For two vertices x' and y' lying in path P in that order, we denote by $P[x'..y']$ the sub-path of P from x' to y'. For paths P_1, P_2 where $f(P_1) = b(P_2)$, we denote by $P_1 \cdot P_2$ the concatenation of these paths.

Proposition 12. *Let G be a triconnected undirected graph, and e_1, e_2, e_3 be three distinct edges such that not all their end-vertices are distinct. Then, there exists a simple cycle L in G that contains e_1, e_2, e_3 if and only if e_1, e_2, e_3 do not all share a common vertex.*

Proof: By examining all possible alternative configurations, beginning with the obvious cases:

1. e_1, e_2, e_3 all share a common end-vertex: then there is no simple cycle containing e_1, e_2, e_3.
2. e_1, e_2, e_3 form a cycle: trivially, the required cycle exists.
3. e_1, e_2, e_3 form a chain. W.l.o.g., let $e_1 = (v_0, v_1), e_2 = (v_1, v_2), e_3 = (v_2, v_3)$. Since G is triconnected, $G' = G \setminus \{v_1.v_2\}$ is connected. Therefore, there exists a simple v_3v_0-path P in G', so $e_1 \cdot e_2 \cdot e_3 \cdot P$ is a simple cycle in G.
4. Two edges meet, and the third edge is disjoint. W.l.o.g., let $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2), e_3 = (v_3, v_4)$. Since G is triconnected, $G' = G \setminus v_1$ is biconnected. So, there exist disjoint paths P_1, P_2 from $\{v_3, v_4\}$ to $\{v_0, v_2\}$ in G'. W.l.o.g., P_1 is a v_4v_0-path, and P_2 is a v_2v_3-path, so $P_1 \cdot e_1 \cdot e_2 \cdot P_2 \cdot e_3$ is a simple cycle in G.

□

We proceed to the main case where all six end-vertices of the given edges are distinct. Henceforth, we denote the *end-vertices of e_3 by s and t*. We begin with a lemma on cycle splicing.

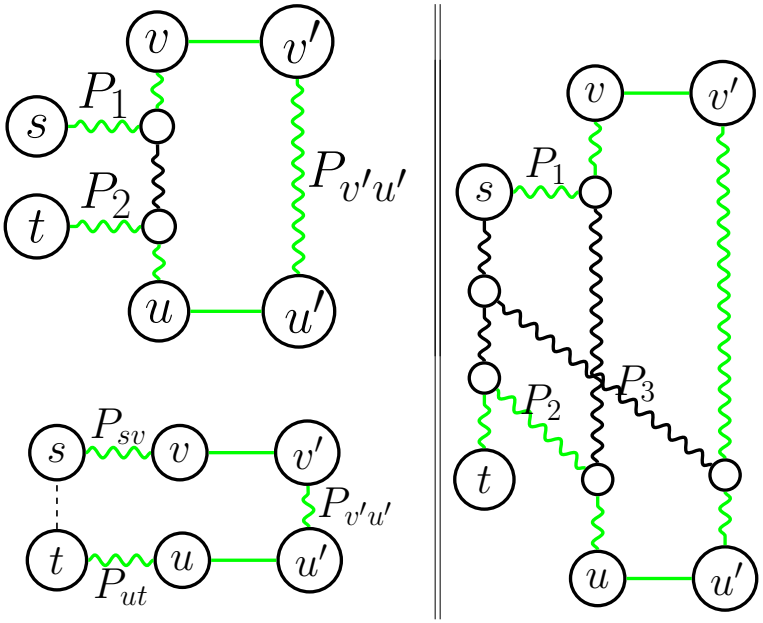


Figure 4: Illustration of Lemma 13 (top left), and of the proof of Proposition 14: case 1 (bottom left), case 2 (right). The constructed st-paths are shown in green.

Lemma 13. *Let $G = (V, E)$ be a graph with $s, t \in V$ and $e_1 = (u, u')$, $e_2 = (v, v') \in E$, where all above vertices are distinct in G. If there are a simple cycle $L = ((u, u'), P_{u'v'}, (v', v), P_{vu})$ in G,*

and mutually disjoint paths P_1 from s to L and P_2 from t to L such that P_i intersects L only at $f(P_i)$, $i = 1, 2$, either both ending at P_{vu} or both ending at $P_{u'v'}$, then there exists a simple st-path containing e_1 and e_2 in G.

Proof: Assume w.l.o.g. that $f(P_1), f(P_2) \in P_{vu}$ and that $f(P_1)$ is on the part of P_{vu} between v and $f(P_2)$. Then (see Figure 4 (top left) for illustration), the required st-path is:

$$P_1 \cdot P_{uv}[f(P_1)..v] \cdot (v, v') \cdot P_{v'u'} \cdot (u', u) \cdot P_{uv}[u..f(P_2)] \cdot \bar{P}_2.$$

□

The disjoint end-vertices case has two sub-cases, the simpler one where $G^- = G \setminus (s, t)$ is triconnected, and the other one where it is not.

Proposition 14. *Let $G = (V, E)$ be a graph with $s, t \in V$ and $e_1 = (u, u')$, $e_2 = (v, v')$ edges in E, where all above vertices are distinct in G and such that $G^- = G \setminus (s, t)$ is triconnected. Then, there exists a simple st-path containing e_1, e_2 in G^-, and, equivalently, a simple cycle containing $e_1, e_2, (s, t)$ in $G^- \cup (s, t)$.*

Proof: By the extension to Menger's theorem, there exist three disjoint paths from $\{s, u, u'\}$ to $\{t, v, v'\}$. Fix such a set of paths $\mathcal{P}$. Since both above vertex sets have cardinality 3, let M be the symmetric mapping between these vertex sets with $M(x) = y$ just when the path $(x...y) \in \mathcal{P}$. There are two possible cases, and in both cases we construct a simple path from s to t through e_1, e_2 in G^-, to which we then can add (s, t) to complete the required cycle in $G^- \cup (s, t)$. For illustration, see Figure 4.

Case 1: If $M(t) = u$ or $M(t) = u'$, then $M(s) = v$ or $M(s) = v'$. Assume the former in each case, w.l.o.g. (due to symmetry). Therefore, we have disjoint paths $P_{ut}, P_{sv}, P_{v'u'} \in \mathcal{P}$ with names denoting their respective endpoints. Then, $P_{sv} \cdot (v, v') \cdot P_{v'u'} \cdot (u', u) \cdot P_{ut}$ is a simple path from s to t as required.

Case 2: If $M(t) = s$, then, w.l.o.g., $M(v) = u$ and $M(v') = u'$. Thus, we have $P_{st}, P_{vu}, P_{u'v'} \in \mathcal{P}$, all disjoint. Therefore, $P_{vu} \cdot (u, u') \cdot P_{u'v'} \cdot (v', v)$ is a simple cycle, which we denote by L. Since $(s, t) \notin G^-$, the path P_{st} must have at least three vertices. Since L has at least four vertices, by the extension of Menger's theorem, there exist three disjoint paths P_1, P_2, P_3 from some vertices in P_{st} to L, so that L intersects P_i only at $f(P_i)$.

Then, at least two of $f(P_1), f(P_2), f(P_3)$ must be on either P_{vu} or $P_{v'u'}$. Assume, w.l.o.g., (the other cases are symmetrical) that $f(P_1), f(P_2) \in P_{vu}$ and that $b(P_1)$ lies before $b(P_2)$ on P_{st}. Then, $P'_1 = P_{st}[s..b(P_1)] \cdot P_1$ and $P'_2 = P_{ts}[t..b(P_2)] \cdot P_2$ are disjoint paths from s and t to P_{vu}. Now, we have the required simple st-path due to Lemma 13. □

We now move to the final sub-case, where G is triconnected but $G^- = G \setminus (s, t)$ is not. Note that attempting to use the same proof outline as in Proposition 14 fails. That is because if the only path P_{st} that is disjoint from L consists exactly of the edge (s, t) in Case 2, we cannot apply the extension to Menger's theorem to get the three paths from P_{st} to L, as the cardinality of the vertex set of P_{st} is only 2. To handle this complication, we use the structure of G^- given by the Decremental Lemma (Lemma 10).

Lemma 15. *Let $G = (V, E)$ be a triconnected graph with vertices s', t', t'' and edges $e_1 = (u, u')$, $e_2 = (v, v')$, where all above vertices are distinct in G. Then, there exists a simple path from s' to either t' or t'' containing e_1, e_2.*

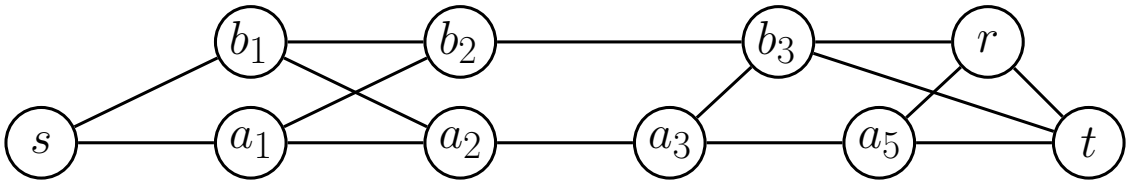


Figure 5: Graph G^- for the graph G shown in Figure 3. Recall that $a_3 = a_4$, $b_3 = b_4 = b_5$, $X_0 = S(s, a_1, b_1)$, $X_1 = R(a_1, b_1, a_2, b_2)$, $X_2 = S(a_2, b_2, a_3, b_3)$, $X_3 = P(a_3, b_3)$, $X_4 = S(a_3, b_3, a_5)$, $X_5 = R(b_3, a_5, r, t)$.

Proof: If $(s', t') \notin G$ then a simple st'-path P that includes e_1, e_2 exists by Proposition 14; likewise in the case where $(s', t'') \notin G$. Hence, we remain with the case where both $(s', t'), (s, t'')$ are in G. Here, we begin as in the proof of Proposition 14, up to its Case 2. We find three disjoint paths from $\{s', u, u'\}$ to $\{t', v, v'\}$. These paths either form a cycle that includes $\{(u, u'), (v, v'), (s', t')\}$, in which case we are done, or form a simple cycle L and an $s't'$-path $P_{s't'}$.

In the latter case, there exist three disjoint paths from s', t', t'' to L, where one of these paths can be degenerate (when t'' is on L). At least two of these paths, which we denote by P_1 and P_2, end up either on P_{uv} or on $P_{u'v'}$; w.l.o.g., let this be P_{uv}. By construction, neither P_1 nor P_2 uses any of the edges directly connecting s', t', t''. If P_1 and P_2 begin at t' and t'' then, applying Lemma 13, we have a simple $t't''$-path P that includes e_1, e_2 but not s'. Then, $(s', t') \cdot P$ is a simple $s't''$-path as required. Otherwise, w.l.o.g., P_1 begins at s' and P_2 at t'. Then, we can apply Lemma 13 getting a simple $s't'$-path as required. □

Proposition 16. *Let $G = (V, E)$ be a triconnected graph, $e_1 = (u, u')$, $e_2 = (v, v'), (s, t)$ edges in G, where all above vertices are distinct in G, such that $G^- = G \setminus (s, t)$ is not triconnected. Then, there exists a simple cycle in G containing $e_1, e_2, (s, t)$ unless $e_1, e_2, (s, t)$ form an edge cut of G.*

Proof: Throughout this proof, we use the terminology and notation of the Decremental Lemma. For uniformity, we set $a_0 = s$ and $a_{k+1} = t$. Refer to Figure 5 in what follows for an illustration of the proof text. The required edges $e_1 = (u, u'), e_2 = (v, v')$ can be real edges either in the same component of G^- or in different components.

Case 1: different components. W.l.o.g., let e_1 be a real edge of component C_i, and e_2 a real edge of C_j, with $i < j$. Denote $G_1 = Exp(C_{i+1}, (a_{i+1}, b_{i+1}))$ and $G_2 = Exp(C_i, (a_{i+1}, b_{i+1}))$; these subgraphs of G are disjoint, except for a_{i+1}, b_{i+1}, by Property 5. Observe that $e_1 \in G_1$ and $e_2 \in G_2$. Our approach is to build *disjoint* simple paths P_1 starting from s and containing e_1 in G_1 and P_2 containing e_2 and ending at t in G_2, so that $f(P_1) = b(P_2) \in \{a_{i+1}, b_{i+1}\}$. This suffices since concatenation $P_1 \cdot P_2 \cdot (t, s)$ is a simple cycle as required.

Initially, assume that C_i is of type S. Due to the Decremental Lemma, we can assume w.l.o.g. that $e_1 = (a_i, a_{i+1})$. If $i = 0$, we assign $P_1 = (a_0, a_1) = (s, a_1)$. Otherwise, Corollary 7 implies existence of a simple sa_i-path P_1' avoiding b_i in $Exp(C_i, (a_i, b_i))$, and we set $P_1 = P_1' \cdot (a_i, a_{i+1}) \subseteq G_1$, $e_1 \in P_1$. Since C_i is of type S, then X_{i+1} is not of type S; hence the subgraph G_2 is biconnected, by Lemma 9. By the extension of Menger's theorem applied to $\{a_{i+1}, t\}$ and the end-vertices of e_2, there exists a simple $a_{i+1}t$-path P_2 containing e_2 in G_2, as required. For illustration (Figure 5), consider the case where $e_1 = (a_2, a_3)$ in the S component C_2 and $e_2 = (b_3, r)$ in the R component C_5, with $P_1' = (s, b_1, a_2)$, $P_1 = (s, b_1, a_2, a_3)$, and $P_2 = (a_3, b_3, r, t)$.

Otherwise (C_i is not of type S), Property 6 implies existence of a simple $a_{i+1}t$-path P_2' containing e_2 in $G_2 \cup (a_{i+1}, b_{i+1})$. Assume initially that $(a_{i+1}, b_{i+1}) \notin P_2'$; then, set $P_2 = P_2' \subseteq G_2$. Since

C_i is not of type S, the subgraph G_1 is biconnected, by Lemma 9. Hence, there exists a simple sa_{i+1}-path P_1 containing e_1 in G_1, as required. Now, if $(a_{i+1}, b_{i+1}) \in P_2'$ then it must be its first edge. Then, $P_2 = P_2' \setminus (a_{i+1}, b_{i+1})$ is a simple $b_{i+1}t$-path in G_2 and $e_2 \in P_2$, as required. Similarly to the above, there exists a simple sb_{i+1}-path P_1 containing e_1 in G_1, as required. For illustration, consider the case where $e_1 = (a_3, b_3)$ in the P component C_3 and $e_2 = (r, t)$ in the R component C_5, with $P_2' = (a_3, b_3, r, t)$, $P_2 = (b_3, r, t)$, and $P_1 = (s, b_1, b_2, b_3)$.

Case 2: same component. Let $e_1, e_2 \in C_i$ for some $0 \leq i \leq k$. Then, C_i cannot be a P component, as a P component has at most one real edge. If C_i is an S component, then, by Corollary 11, its two real edges e_1, e_2 always form a 2-edge cut of G^- separating s and t, and thus $\{e_1, e_2, (s, t)\}$ is a 3-edge cut of G. By Property 2, no simple cycle that contains all edges of this cut exists (the exception case in this proposition).

Therefore, we only need to consider R components C_i. We construct the simple cycle as required using the following steps: first, we find a simple path P from either a_i or b_i to a_{i+1} or b_{i+1} containing e_1, e_2 in C_i; then, this path is made free of virtual edges and extended to reach s and t; finally, adding the edge (t, s) to the extended path results in a cycle as required.

In order to find the path P, consider first the case where, w.l.o.g., e_1 is incident on one of a_i, b_i, a_{i+1}, and b_{i+1}. Then, there exists a vertex pair: one of a_i, b_i and the other one of a_{i+1}, b_{i+1}, such that one is incident on e_1 and one is not. W.l.o.g., let e_1 be incident on a_i but not on a_{i+1} (when $i = k$, we can have e_1 incident on $a_{i+1} = t$ but not on a_i, which is symmetric). By the condition of the proposition, e_2 cannot also be incident on a_i. Therefore, we can apply Proposition 12 to the graph $C_i \cup (a_{i+1}, a_i)$, which implies the existence of a simple cycle L that includes $e_1, e_2, (a_{i+1}, a_i)$ in this graph. Then, the path $P = L \setminus (a_{i+1}, a_i)$ is as required.

So, assume that neither e_1 nor e_2 is incident on any of $a_i, b_i, a_{i+1}, b_{i+1}$. We find a path P as required by a reduction to Lemma 15. If $i = 0$, then the reduction is achieved by assigning $s' = s$, $t' = a_1$, $t'' = b_1$. Symmetrically, if $i = k$, then the reduction is by assigning $s' = t$, $t' = a_k$, $t'' = b_k$, so that P is found in the reverse direction. If $1 \leq i \leq k - 1$, then either all $a_i, b_i, a_{i+1}, b_{i+1}$ are distinct, or, w.l.o.g., the only coincidence is $b_i = b_{i+1}$. In either case, the reduction is made by assigning $s' = a_i$, $t' = a_{i+1}$, $t'' = b_{i+1}$.

Now, we get rid of any virtual edges in P as follows. If (a_i, b_i) is on P in C_i then it must be its first edge. Likewise, if (a_{i+1}, b_{i+1}) is on P in C_i then it must be its last edge. Removing these edges, if any, we have another path $P' = P \setminus \{(a_i, b_i), (a_{i+1}, b_{i+1})\}$ as required without the virtual edges.

Then, we extend P' as follows. If P' starts at a_i, we extend it at its beginning by concatenating it with a simple path from s to a_i avoiding b_i in $Exp(C_i, (a_i, b_i))$, which exists by Corollary 7; similarly, if P' starts at b_i. If P' ends at a_{i+1}, it is likewise extended at its end by concatenating it with a simple path from a_{i+1} avoiding b_{i+1} to t in $Exp(C_i, (a_{i+1}, b_{i+1}))$; similarly, if P' ends at b_{i+1}. The extended path is simple by construction and due to Property 5. For illustration, consider the case where $e_1 = (a_1, a_2)$ and $e_2 = (b_1, b_2)$ in the R component C_1, with $P' = P = (a_1, a_2, b_1, b_2)$ and the extending paths are (s, a_1) and (b_2, b_3, t). □

Now, we can state and prove the main result of this section.

Theorem 4. *Let $G = (V, E)$ be a triconnected graph, and $e_1, e_2, e_3 \in E$. Then, there exists a simple cycle L in G such that $e_1, e_2, e_3 \in L$ if and only if neither of the following conditions occur:*

1. *e_1, e_2, e_3 all share a common end-vertex.*
2. *$\{e_1, e_2, e_3\}$ is an edge cut of G.*

Proof: The only-if part for condition 1 is immediate; for condition 2 this is due to Property 2. The if part follows immediately by collecting all the cases in Propositions 12, 14, and 16. □

4 CEP on Biconnected Graphs

In Section 2.1, we argued that in general undirected graphs, the CEP (and thus the PEP) can be easily reduced to biconnected graphs. Above, we have determined necessary and sufficient conditions for the CEP in triconnected graphs. In this section, we analyze the CEP on an arbitrary biconnected graph $G = (V, E)$, using its partition into triconnected components provided by the SPQR tree $\mathcal{T} = \mathcal{T}(G)$. For notation, definitions, and properties of SPQR trees, refer to Section 2.2. In this section, for brevity, we identify node X of $\mathcal{T}$ and the corresponding component C_X of the graph if this does not lead to ambiguity. We begin with defining the crucial concept of a component *central* w.r.t. a given set of elements.

Definition 4 (Central Component)**.** *For any subset S of elements of G, we call a component C of $\mathcal{T}$* central *w.r.t. S, if all elements $r_C(x)$, $x \in S$, of C are distinct.*

A central component does not necessarily exist in general; and if it exists, it is not necessarily unique. For our purposes, it is sufficient that a central component always exists for sets of size 3.

Lemma 17 (Central Component Lemma)**.** *For every set S of* three *distinct elements of G, there exists a component of $\mathcal{T}$ central w.r.t. S.*

Proof: Denote the three given elements of G by x_1, x_2, x_3. Let us choose three components C_i, $1 \leq i \leq 3$, such that $r_{C_i}(x_i) = x_i$, i.e. such that each x_i is a *real* element in C_i. If at least two of these components coincide, w.l.o.g., $C_1 = C_2 = C$, then C is central w.r.t. S. Indeed, by our assumption, $x_1 = r_C(x_1)$ and $x_2 = r_C(x_2)$ are distinct non-virtual elements. Additionally, either $r_C(x_3) = x_3$ or $r_C(x_3)$ is a virtual edge, which must in both cases be distinct from $r_C(x_1)$ and $r_C(x_2)$.

Otherwise, all three chosen components are distinct. If some C_i lies on the path in $\mathcal{T}$ between the two other components, then the two elements $r_{C_i}(x_j)$, $j \neq i$, must be distinct: this is trivial if at least one of them is real, and due to C_i being on the path as above if both are represented by virtual edges. Thus, C_i is central w.r.t. S here too.

Otherwise, there exists a unique component, C, in tree $\mathcal{T}$ such that all C_1, C_2, C_3 are in distinct sub-trees hanging from C in $\mathcal{T}$. Then, $x_i \in Exp(C, e_i)$, $i = 1, 2, 3$, where e_i are distinct virtual edges of C. The component C is central w.r.t. S since each of the three x_i is represented by e_i in C, unless x_i is a real element of C. □

We now state and prove a crucial proposition on central components and must-include cycles.

Proposition 18. *Let $S = \{x_1, x_2, x_3\}$ be a set of three elements of G, and C be any component central w.r.t. S. Then, there exists a simple cycle containing x_1, x_2, x_3 in G if and only if there exists a simple cycle containing $r_C(x_1), r_C(x_2), r_C(x_3)$ in C.*

Proof: (**If**) Let L' be a simple cycle in C containing all $r_C(x_1), r_C(x_2), r_C(x_3)$. If L' contains no virtual edges, we are done since L' is a cycle in G. Otherwise, replace every virtual edge $e \in L'$ by a path $P(e)$ as follows, to create the simple cycle L in G as required. Let $e = (a, b) \in L'$ be a virtual edge. If e represents no element of S, then by Corollary 7, there exists a simple path $P(e)$ from a to b in $Exp(C, e)$. Likewise, let e represent element $x_i \in S$; then, e represents no other element of

S, since C is a central component w.r.t. S. By Corollary 8, there exists a simple path $P(e)$ from a to b containing x_i in $Exp(C, e)$. By replacing every virtual edge e in L' by the simple path $P(e)$, we obtain a cycle, L, in G containing all elements of S. Cycle L is simple since graphs $Exp(C, e)$ as above are disjoint, except possibly for end-vertices of edges e, by Property 5; this exception cannot spoil simplicity of L, since L' is simple.

(**Only if**) Let L be a simple cycle in G containing all three elements of S. If L consists only of elements of C, it is a cycle in C, as required. Otherwise, cycle L is sub-divided into the inclusion-maximal sub-paths whose edges are either all in C or all in the same subgraph $Exp(C, (a, b))$, where (a, b) is a virtual edge of C. By Property 5, the end-vertices of each such sub-path in $Exp(C, (a, b))$ are a and b. Therefore, replacing each such sub-path by virtual edge (a, b) in C results in a cycle, L', in C. Cycle L' is as required, since if x_i is an element of $Exp(C, (a, b))$, then $r_C(x_i) = (a, b)$ is an edge of L'. □

The reader may be curious about the relation of multiple central components to Proposition 18. In fact, absence of a cycle as required in a component, C, is a quite special situation. This is impossible if C is an S component and happens in a P component C only if all three its must-include elements are edges. Since any R component is triconnected, by Theorem 3, such a cycle can be absent in an R component also only if all three must-include elements are edges. By the following lemma, in the case of three edges, the central component is unique.

Lemma 19. *If all three elements of S are represented by* edges *in a component C central w.r.t. S, then C is a unique central component w.r.t. S.*

Proof: Let distinct edges e_1, e_2, e_3 represent the elements $x_1, x_2, x_3 \in S$, respectively, in a component C central w.r.t. S. Consider another arbitrary component C'; necessarily, $C' \in B(C, e)$ for some virtual edge $e \in C$. Since e_1, e_2, e_3 are distinct edges, then only one of them can be the same as e, so suppose, w.l.o.g., that $e_1 \neq e$ and $e_2 \neq e$. This implies that neither x_1 nor x_2 are in $B(C, e)$. Therefore, both elements x_1, x_2 are represented in C' by the same virtual edge e', such that C is in the sub-tree $B(C', e')$. This implies that C' is not central w.r.t. S, as required. □

We conclude this section by summarizing the results on must-include cycles from Sections 3 and 4 as one theorem, and with some examples of how it is applied to finding such cycles.

Theorem 5. *Let G be a biconnected graph, and x_1, x_2, x_3 be three distinct elements of G. Let C be any central component of $\mathcal{T}(G)$ w.r.t. x_1, x_2, x_3. Then, a simple cycle in G including x_1, x_2, x_3 does not exist if and only if all of $r_C(x_1), r_C(x_2), r_C(x_3)$ are edges in C and at least one of the following conditions holds:*

1. *The edges $r_C(x_1), r_C(x_2), r_C(x_3)$ all have a common end-vertex in C.*
2. *C is an R component and the edges $r_C(x_1), r_C(x_2), r_C(x_3)$ form a 3-edge cut of C.*

Proof: The theorem statement is consistent since Lemma 17 guarantees existence of a central component C w.r.t. x_1, x_2, x_3 in $\mathcal{T}(G)$. Due to Proposition 18, it is sufficient to prove that the conditions of the theorem are necessary and sufficient for non-existence of a simple cycle in C that includes $r_C(x_1), r_C(x_2), r_C(x_3)$; note that these three elements of C are distinct since C is central w.r.t. x_1, x_2, x_3.

Consider now all possible types of the component C. If C is an S component, the cycle as required trivially always exists. Accordingly, no three edges of an S component (which is a cycle) share a common end-vertex, while exception 2 is not relevant. If C is a P component, the cycle

never exists if all three elements are represented by edges in C, which is covered by exception 1, and always exists otherwise, which also accords with the theorem statement. In the case where C is an R component (which is a triconnected graph), the theorem follows immediately from Theorem 3 (such a cycle always exists in a triconnected graph if not all elements are edges) and Theorem 4 (the conditions for existence of a cycle with three must-include edges in triconnected graphs). □

To illustrate the usage of Theorem 5 for finding must-include paths, if they exist, consider the graph G shown in Figure 2 (top left). In what follows, we use the reduction of the PEP to the CEP, the proof of Proposition 18, and the other parts of Figure 2 for illustration. For simplicity, we choose must-include path query examples where the source and destination vertices are already connected by an edge in the graph, so that we can use the SPQR tree already shown in Figure 2 (recall that the PEP query answer does not depend on presence/absence of such an edge).

Consider querying for a simple path from w_4 to x that must include u_1 and u_6. Here, the equivalent CEP is for elements $(w_4, x), u_1, u_6$, for which the central component is $C = R(u_1, u_2, u_4, u_5)$. The cycle $L' = (u_1, u_2, u_5)$ in C includes vertex u_1 and virtual edges $r_C(u_6) = (u_2, u_5)$ (labeled 5) and $r_C((w_4, x)) = (u_1, u_2)$ (labeled 6). By substituting edge (u_1, u_2) with path $(u_1, x, w_4, t, u_3, u_2)$ and edge (u_2, u_5) with path (u_2, u_6, u_5), we get cycle $L = (u_1, x, w_4, t, u_3, u_2, u_6, u_5)$. By removing edge (x, w_4) from L, we get the simple path $(w_4, t, u_3, u_2, u_6, u_5, u_1, x)$ in G, as required.

If queried for a must-include path from w_4 to x through u_1 and w_2, the equivalent CEP query elements are $(w_4, x), u_1, w_2$ and the central component is $C = P(x, t)$. The representatives of $(w_4, x), u_1, w_2$ in C are three distinct virtual edges. Since all these edges share both end-vertices, no simple cycle including $(w_4, x), u_1, w_2$ exists in G, due to exclusion 1 of Theorem 5. Therefore, the required must-include path also does not exist.

If queried for a must-include path from u_2 to u_4 via w_4 and u_6, the equivalent CEP elements are $(u_2, u_4), w_4, u_6$ and the central component is $C = R(u_1, u_2, u_4, u_5)$. The triple of representatives $(u_2, u_4), (u_2, u_1), (u_2, u_5)$ of the must-include CEP elements in C is covered by both exclusions 1 and 2 of Theorem 5, which implies that the answer to the query is negative.

5 Algorithms for PEP/CEP and EPE

We now turn to the algorithms for solving the CEP and EPE. Though both algorithms are stated to work for the biconnected graphs, the reduction provided in Section 2.1 implies similar results for the general graphs. To solve the PEP, we use the CEP algorithm and the reduction from PEP to CEP in Section 1.1.

5.1 Determining Must-Include Paths and Cycles

We can now state Algorithm 1 for the CEP. Although an algorithm follows from the proofs of Proposition 18 and Lemma 17, here we state the algorithm explicitly and analyze its complexity.

Theorem 6. *Given any biconnected graph $G = (V, E)$, Algorithm 1 always answers the CEP correctly, and has time complexity $O(|E|)$.*

Proof: The correctness of Algorithm 1 follows from Theorem 5. The complexity of computing the SPQR tree of G is $O(|E|)$. Computing the representatives of x_i can be done by traversing $\mathcal{T}$ from any of its nodes in $O(|E|)$ time as well. It is clear from examining the rest of the steps of Algorithm 1 that they all take time $O(|E|)$, with some of the steps in fact only requiring a constant time. □

Algorithm 1: Determining Must-Include Cycle Existence (CEP)

Input: A biconnected graph G and three of its elements x_1, x_2, x_3

Compute $\mathcal{T}$, the SPQR tree of G
Compute representatives $r_C(x_1), r_C(x_2), r_C(x_3)$ for all components $C \in \mathcal{T}$
Find a component C central w.r.t. x_1, x_2, x_3
if $r_C(x_i)$ is a vertex for at least one $1 \leq i \leq 3$ or C is of type S **then**
    return true
else if C is of type P **then**
    return false
end if
/* Assert: C is of type R and all $r_C(x_i)$ are edges */
if $r_C(x_1), r_C(x_2), r_C(x_3)$ all have a common end-vertex
    or $r_C(x_1), r_C(x_2), r_C(x_3)$ form an edge cut of C **then**
    return false
else
    return true
end if

Due to the equivalence between PEP and CEP stated in Section 2.1, PEP(G, s, t, w_1, w_2) can be solved by running Algorithm 1 for CEP$(G \cup (s,t), (s,t), w_1, w_2)$, with the same time bound.

5.1.1 Finding the Cycle

It may be of interest to *actually find* a simple cycle that includes the given elements x_1, x_2, x_3 when it exists. Naturally, we begin by constructing $\mathcal{T}(G)$ and finding the representatives of x_1, x_2, x_3 and a central component C, as in Algorithm 1. Then, consider any element x_i that is not a real element in C. Finding the portion of the required cycle outside C that contains x_i entails finding a path between the ends of the virtual edge $e = r_C(x_i)$ in $Exp(C, e)$. This can be done in linear time by finding two disjoint (except at x_i, if x_i is a vertex) paths between x_i and the ends of e in the subgraph $Exp(C, e)$ using the Ford-Fulkerson algorithm (see Section 2.1).

Finding the part of the cycle in C that includes all the $r_C(x_i)$ (to be spliced with the paths as above) is trivial for S and P components. If C is of type R and at least one of the elements x_i is a real vertex of C, this can be done as described in Section 2.3. The non-trivial task is finding the portion of the cycle as above in the central component when C is of type R and all $r_C(x_i)$ are edges. Here, there are many cases to handle, so we specify this part of the algorithm only implicitly. Observe that the proof of Theorem 4, including its required lemmas, specifies the required cycle in a constructive way, except for finding paths as in Menger's theorem; the latter is covered in Section 2.1. Thus, by following the cases of the proof, we (implicitly) get an $O(|E|)$ algorithm for finding a simple cycle containing x_1, x_2, x_3. By the reduction in Section 2.1, this also implies an $O(|E|)$ algorithm for finding a simple st-path containing w_1 and w_2.

5.2 Enumerating Exclusion Pairs

Consider now the EPE problem for any graph $G = (V, E)$ such that $G^+ = G \cup (s,t)$ is biconnected. The idea here is to visit every component C' that could potentially be central in $\mathcal{T} = \mathcal{T}(G^+)$ w.r.t. (s,t) and some two vertices not in C', and for each such component emit all possible vertex pairs

excluded from being on a simple cycle containing (s,t) in G^+ by Theorem 5.

Again, for brevity, we identify the SPQR tree nodes with the components associated with them in the description and analysis of the Algorithm 2 for the EPE. The algorithm starts with the component C that has (s,t) as a real edge, assigning C to be the root of $\mathcal{T}$, and visits each component C' recursively using procedure TRAVERSE. In each of such C', we consider all virtual edges other than $r_{C'}((s,t))$ as potentially generating exclusions. For convenience, we use s', t' to denote the ends of $r_{C'}((s,t))$ in C'. Observe that for any child C'' of C', the structural edge (C', C'') is associated with the virtual edge $r_{C''}((s,t))$ in C''. This allows us to cheaply supply $r_{C''}((s,t))$ as an argument in the recursive calls in the procedure TRAVERSE.

Algorithm 2: Exclusion-Pair Enumeration (EPE)

Input: a biconnected graph $G = (V, E)$ and two of its vertices s, t

Let $G^+ = G \cup (s,t)$
Compute $\mathcal{T} = \mathcal{T}(G^+)$, the SPQR tree of G^+
Root $\mathcal{T}$ at C, the unique component of $\mathcal{T}$ where $r_C((s,t)) = (s,t)$ is a real edge
Call TRAVERSE$(C, (s,t))$
exit
procedure TRAVERSE$(C', (s', t'))$
    Let $(C')^- = C' \setminus (s', t')$ and E^{virt} be the set of virtual edges of $(C')^-$
    if C' is a P node **then**
        if $|E^{\text{virt}}| \geq 2$ **then**
            EMITPAIRS(C', E^{virt})
        end if
    else if C' is a non-leaf R node **then**
        for every $u \in \{s', t'\}$ **do**
            Let $E_{EX} = \{(u,v) | (u,v) \in E^{\text{virt}}\}$
            if $|E_{EX}| \geq 2$ **then**
                EMITPAIRS(C', E_{EX})
            end if
        end for
        Let $Cuts$ = FIND2EDGECUTS$((C')^-, (s', t'))$
        for all $E_{EX} \in Cuts$ s.t. $E_{EX} \subseteq E^{\text{virt}}$ **do**
            EMITPAIRS(C', E_{EX})
        end for
    end if
    for every child C'' of C' in $\mathcal{T}$ **do**
        Call TRAVERSE$(C'', r_{C''}((s,t)))$
    end for
end procedure

In each component C', given some set $E' \subseteq E^{\text{virt}}$, we use the subroutine EMITPAIRS(C', E') to emit exclusion vertex pairs, as follows. For any virtual edge e in C', denote by $V(C', e)$ the set of vertices of $Exp(C', e)$ excluding the end-vertices of e. Then, for every pair of distinct virtual edges $\{e_1, e_2\} \subseteq E'$, the exclusion vertex pair $\{v_1, v_2\}$ is emitted for every pair of vertices $v_1 \in V(C', e_1)$ and $v_2 \in V(C', e_2)$. EMITPAIRS(C', E') can work either implicitly, i.e., just output pointers to C' and to the set of virtual edges E', or explicitly, i.e., output each exclusion vertex pair $\{v_1, v_2\}$

explicitly.

EmitPairs is called exactly as indicated by Theorem 5. The S nodes do not cause exclusion pairs, because they are cycles and thus there always trivially exists a cycle containing any set of elements therein. So, the algorithm does not call EmitPairs for S nodes. If C' is a P node, all pairs of virtual edges in $(C')^-$ cause exclusions, which are emitted.

If C' is an R node, exclusions are emitted (in line 16) for every (maximal) set of cardinality at least 2 of virtual edges of $(C')^-$ that are all incident on s', or all incident on t'. Additional exclusions are emitted for every pair E_{EX} of virtual edges forming a 3-edge cut of C' together with edge (s', t'), unless the edges in E_{EX} are both incident on s', or both incident on t', in which case we do not need to emit them again (already emitted in line 16). Note that $\tilde{E}$, $(s', t') \in \tilde{E}$, is a 3-edge cut in C' if and only if $\tilde{E} \setminus (s', t')$ is a 2-edge cut separating s' and t' in $(C')^- = C' \setminus (s', t')$.

We now turn to the only remaining part of our solution to EPE: finding the 2-edge cuts separating vertices s' and t' in an arbitrary graph H^- where the graph $H = H^- \cup (s', t')$ is triconnected (see the call to procedure Find2EdgeCuts therein). Note that in a general graph, it is not trivial to list all edge-cuts of a given cardinality separating two given vertices, even in the easier case when they are edge mincuts; moreover, there might be exponentially many of such cuts. However, our case is quite specific: if H^- is triconnected then, by Lemma 4, there are no 2-edge cuts in H^-; otherwise, Lemma 10 (the Decremental Lemma) and Corollary 11 explicitly describe the 2-edge cuts of H^-. In the latter case, accordingly, the routine Find2EdgeCuts is as follows. The 2-edge cuts that are incident on s' or t', if any, can be found straightforwardly, but are ignored because these cuts were already handled in line 16. For example, the green edges (s, a_1) and (s, b_1) in Figure 3 form an edge-cut, which we ignore here. The following simple scheme finds all other 2-edge cuts in a time linear in the size of H^-: construct $\mathcal{T}(H^-)$; if it consists of a single R node, return an empty set; otherwise, for each of its S components that is a 4-cycle, pick its (only) pair of real edges as a 2-edge cut. For example, the red edges in Figure 3 are the real edges of such an S component.

Since the total size of the SPQR data structure of G is linear in $|E|$, the total time of all runs of Find2EdgeCuts in Algorithm 2 on G is $O(|E|)$. *Remark*: Note that the constant in the linear runtime bound of computing SPQR trees might not be so low. For the case where a practically efficient algorithm is needed, a fast algorithm for finding all 2-edge $s't'$-cuts in H^- based on the Ford-Fulkerson algorithm is suggested in the preliminary version of this paper [6].

Theorem 7. *Given any graph $G = (V, E)$ with vertices s, t such that $G^+ = G \cup (s, t)$ is biconnected, Algorithm 2 emits all the exclusion pairs w.r.t. (G, s, t), and only these pairs, without duplication. The algorithm has a time complexity $O(|E|)$ for implicit output and $O(\max\{|E|, N\}) = O(|V|^2)$ for explicit output, where N is the number of exclusion pairs emitted.*

Proof: The correctness of Algorithm 2 is shown as follows. First, recall that whenever Traverse$(C', (s', t'))$ is called, we have $(s', t') = r_{C'}((s, t))$. So, by Theorem 5 and by the description of EmitPairs, all the emitted vertex pairs are exclusion pairs w.r.t. s, t. Next, let us show that every exclusion pair $\{w_1, w_2\}$ w.r.t. s, t is emitted. Let $\{w_1, w_2\}$ be such an exclusion pair. Then, Theorem 5 implies that there exists a component C of type P or R central w.r.t. $w_1, w_2, (s, t)$ in $\mathcal{T}$, such that all $r_C(w_1), r_C(w_2), r_C((s, t))$ are edges. In both cases, due to Theorem 5, $w_1 \in V(C, r_C(w_1))$, $w_2 \in V(C, r_C(w_2))$, and EmitPairs emits all pairs of vertices where one is represented by $r_C(w_1)$ and the other by $r_C(w_2)$; that is, $\{w_1, w_2\}$ is emitted exactly once during the execution of Traverse$(C, r_C(s, t))$. Finally, assume that the pair $\{w_1, w_2\}$ is emitted during processing of another component C'. Note that this can only occur in lines 10, 16, or 21 of the EPE algorithm. According to the algorithm description, $r_{C'}(w_1)$, $r_{C'}(w_2)$, $r_{C'}((s, t))$ must be distinct edges. Therefore, C' is

another central component w.r.t. $w_1, w_2, (s,t)$, in a contradiction to Lemma 19.

By examination of the pre-order traversal of the components of $\mathcal{T}$, the $O(|E|)$ time complexity for *implicit output* follows from each edge in components of $\mathcal{T}$ being examined at most a constant number of times and the $O(|E|)$ size of the entire SPQR tree data structure. For *explicit output*, the required list of vertex pairs can be generated in $O(\max\{|E|, N\})$ using the following lemma. Recall that $B(C', e)$ denotes the sub-tree of $\mathcal{T}$ hanging from C' on the structural edge that a virtual edge e in C' is associated with.

Lemma 20. *For any $e \in E^{virt}$, the number of nodes in $B(C', e)$ is at most twice $|V(C', e)|$.*

Proof: Consider the scan of $B(C', e)$ from C'. Each descendant component $\tilde{C}$ contains at least one vertex not counted before visiting $\tilde{C}$, except for P components. Note that any P component $\tilde{C}$ has at least one child and does not have children of type P. Hence, we can choose any of its children and charge it for $\tilde{C}$. The lemma follows. □

Due to the lemma, assuming the list of vertices in each component $\tilde{C}$ is explicitly stored, the complexity of creating the list of vertices in $V(C', e)$ via the scan of $B(C', e)$ is thus linear in the list size, $O(|V(C', e)|)$. Therefore, even using the straightforward scheme of creating the lists of vertices in $V(C', e)$ for all $e \in E'$ from scratch whenever EmitPairs(C', E') is called, and then simply iterating over the pairs of lists to produce the explicit output, takes time linear in the number of vertex pairs emitted. □

Note that Algorithm 2 can easily be modified to emit all *exclusion-pairs of edges*, rather than vertices. Simply omit the conditions checking whether the edges in E_{EX} are virtual, allowing real edges to be emitted as well. The semantics of EmitPairs is then changed to emit the edge pairs using the set of edges in each $Exp(C', e)$ instead of the set of vertices in $V(C', e)$. Additionally, when computing the set of edges for the second argument (E_{EX}) of EmitPairs, real edges are also added: whenever Algorithm 2 indicates E^{virt} this is replaced by *all* edges of C^-. Then, EmitPairs interprets a real edge e in its E_{EX} argument as an exclusion involving e itself, rather than the reference to the edges in $Exp(C', e)$. Generalizing to all *exclusion-pairs of elements* (vertices and edges) can be done similarly.

6 Conclusion

This paper closes the gap in the study on existence of a simple cycle containing three given elements (vertices or edges) in an undirected triconnected graph. For the last unsolved case of three given edges, we prove that two simple necessary conditions are also sufficient. Based on this result and on the properties of SPQR trees, we extend this result to cycles and paths that must include three given elements of a general undirected graph. These theoretical results enable linear-time algorithms answering the existence question and finding the required path / cycle.

Further theoretical research in this direction can be done in several directions: that of existence of paths and cycles which must include more than three given graph elements, the same question restricted to k-connected graphs, $k = 2, 3, 4, \ldots$, the reduction of these questions in general graphs to their k-connected subgraphs, etc.

The results of this paper were already used to compute a heuristic to solve certain combinatorial search problems [4], in a way that considerably speeds up the computation of the heuristic. Future work could further speed up computation of the heuristic by extending our EPE algorithm to cover incremental computing of the heuristic.

Acknowledgements

The authors are indebted to an anonymous reviewer for valuable comments, especially for the idea to make the proof of Proposition 16 more structural using SPQR trees; this idea resulted in the Decremental Lemma that made this proof indeed much better structured and more concise.

References

[1] G. D. Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996. doi:10.1007/BF01961541.

[2] K. Bérczi and Y. Kobayashi. The directed disjoint shortest paths problem. In K. Pruhs and C. Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7824, doi:10.4230/LIPIcs.ESA.2017.13.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[4] G. Dahan, I. Tabib, S. E. Shimony, and Y. Dinitz. Generalized longest simple path problems: Speeding up search using SPQR trees. In A. Felner and J. Li, editors, *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, pages 28–36. AAAI Press, 2024. URL: https://doi.org/10.1609/socs.v17i1.31539, doi:10.1609/SOCS.V17I1.31539.

[5] G. Dahan, I. Tabib, S. E. Shimony, and A. Felner. Generalized longest path problems. In L. Chrpa and A. Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 56–64. AAAI Press, 2022. URL: https://doi.org/10.1609/socs.v15i1.21752, doi:10.1609/SOCS.V15I1.21752.

[6] Y. Dinitz and S. E. Shimony. On existence of must-include paths and cycles in undirected graphs. *CoRR*, abs/2302.09614, 2023. arXiv:2302.09614, doi:10.48550/ARXIV.2302.09614.

[7] Y. Dinitz and A. Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In F. T. Leighton and M. T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 716–725. ACM, 1994. doi:10.1145/195058.195442.

[8] Y. Dinitz and A. Vainshtein. The general structure of edge-connectivity of a vertex subset in a graph and its incremental maintenance. odd case. *SIAM J. Comput.*, 30(3):753–808, 2000. doi:10.1137/S0097539797330045.

[9] G. A. Dirac. In abstrakten graphen vorhandene vollständige 4-graphen und ihre unterteilungen. *Mathematische Nachrichten*, 22(1-2):61–85, 1960. doi:10.1002/mana.19600220107.

[10] T. Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998. URL: https://www.sciencedirect.com/science/article/pii/S0166218X97001212, doi:10.1016/S0166-218X(97)00121-2.

[11] D. Eppstein and E. Mumford. Steinitz theorems for orthogonal polyhedra. *CoRR*, abs/0912.0537, 2009. URL: http://arxiv.org/abs/0912.0537, arXiv:0912.0537.

[12] S. Even. *Graph Algorithms*. Cambridge University Press, 2 edition, 2011. URL: https://books.google.it/books?id=m3QTSMYm5rkC, doi:10.1017/CBO9781139015165.

[13] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, USA, 2010.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979. URL: http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455.

[15] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Proceedings of the 8th International Symposium on Graph Drawing*, September 2000. doi:10.1007/3-540-44541-2_8.

[16] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[17] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973. arXiv:https://doi.org/10.1137/0202012, doi:10.1137/0202012.

[18] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

[19] M. Lavrov. k-connected graph, there exists a cycle that contains any 2 edges and any k-2 vertices. https://math.stackexchange.com/questions/3599622/k-connected-graph-there-exists-a-cycle-that-contains-any-2-edges-and-any-k-2?noredirect=1, 2020.

[20] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *ACM Sigda Newsletter*, 5(3):31–36, sep 1975. doi:10.1145/1061425.1061430.

[21] N. Matteo. G a k-connected graph. show that a set of k-2 vertices and a set of two edges lie on a common cycle. https://math.stackexchange.com/questions/3600673/g-a-k-connected-graph-show-that-a-set-of-k-2-vertices-and-a-set-of-two-edges-li?rq=1, 2020.

[22] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. URL: http://eudml.org/doc/211191.

[23] B. A. Reed, N. Robertson, A. Schrijver, and P. D. Seymour. Finding disjoint trees in planar graphs in linear time. *Contemporary Mathematics*, 147:295–301, 1993.

[24] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

[25] P. D. Seymour. Four-terminus flows. *Networks*, 10(1):79–86, 1980. doi:10.1002/net.3230100108.

[26] H. Vardhan, S. Billenahalli, W. Huang, M. Razo, A. Sivasankaran, L. Tang, P. Monti, M. Tacca, and A. Fumagalli. Finding a simple path with multiple must-include nodes. In *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 1–3, 2009. doi:10.1109/MASCOT.2009.5366808.