Theoretical Computer Science 447 (2012) 74–84

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Finding strong bridges and strong articulation points in linear time[☆]

Giuseppe F. Italiano [a], Luigi Laura [b,*], Federico Santaroni [a]

[a] Dipartimento di Informatica, Sistemi e Produzione, Università di Roma "Tor Vergata", via del Politecnico 1, 00133 Roma, Italy

[b] Dipartimento di Informatica e Sistemistica "Antonio Ruberti", "Sapienza" University of Rome, via Ariosto 25, I-00185 Roma, Italy

ARTICLE INFO

Keywords:
Graph algorithms
Strong connectivity
Strong articulation points
Strong bridges

ABSTRACT

Given a directed graph G, an edge is a strong bridge if its removal increases the number of strongly connected components of G. Similarly, we say that a vertex is a strong articulation point if its removal increases the number of strongly connected components of G. In this paper, we present linear-time algorithms for computing all the strong bridges and all the strong articulation points of directed graphs, solving an open problem posed in Beldiceanu et al. (2005) [2].

1. Introduction

We assume that the reader is familiar with the standard graph terminology, as contained for instance in [6]. Let $G = (V, E)$ be a directed graph, with m edges and n vertices. A *directed path* in G is a sequence of vertices $v_1, v_2, \ldots, v_k$, such that edge $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k - 1$. A directed graph G is *strongly connected* if there is a directed path from each vertex in the graph to every other vertex. The *strongly connected components* of G are its maximal strongly connected subgraphs. Given a directed graph G, an edge is a strong bridge if its removal increases the number of strongly connected components of G. Similarly, we say that a vertex is a strong articulation point if its removal increases the number of strongly connected components of G.

Strong articulation points and strong bridges are related to the notion of 2-vertex and 2-edge connectivity of directed graphs. We recall that a strongly connected graph G is said to be 2-vertex-connected if the removal of any vertex leaves G strongly connected; similarly, a strongly connected graph G is said to be 2-edge-connected if the removal of any edge leaves G strongly connected. It is not difficult to see that the strong articulation points are exactly the vertex cuts for 2-vertex connectivity, while the strong bridges are exactly the edge cuts for 2-edge connectivity: G is 2-vertex-connected (respectively 2-edge-connected) if and only if G does not contain any strong articulation point (respectively strong bridge). The 2-vertex and 2-edge connectivity of a directed graph can be tested in linear time. For 2-vertex connectivity, there is a very recent algorithm of Georgiadis [14]. Although there is no specific linear-time algorithm in the literature, the 2-edge connectivity of a directed graph can be tested in $O(m + n)$ as observed somewhat indirectly in [12]: one can test whether a directed graph is 2-edge-connected by using Tarjan's algorithm [20] to compute two edge-disjoint spanning trees in combination with the disjoint set-union algorithm of Gabow and Tarjan [13]. However, both the algorithm of Georgiadis and the combination of Tarjan's algorithm with the set-union data structure of Gabow and Tarjan do not find *all* the strong articulation points or *all* the strong bridges of a directed graph.

[☆] This work has been partially supported by the 7th Framework Programme of the EU (Network of Excellence "EuroNF: Anticipating the Network of the Future - From Theory to Design") and by MIUR, the Italian Ministry of Education, University and Research, under Project AlgoDEEP. A preliminary version of this paper was presented at the *4th International Conference on Combinatorial Optimization and Applications* (COCOA 2010) (Italiano et al., 2010) [15].

[*] Corresponding author.
E-mail addresses: italiano@disp.uniroma2.it (G.F. Italiano), laura@dis.uniroma1.it (L. Laura), federico.santaroni@gmail.com (F. Santaroni).
URLs: http://www.disp.uniroma2.it/users/italiano/ (G.F. Italiano), http://www.dis.uniroma1.it/~laura (L. Laura).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.11.011

In this paper, we show how to find *all* the strong bridges and *all* the strong articulation points of a directed graph in $O(m + n)$ worst-case time. Besides being natural concepts in graph theory, strong articulation points and strong bridges find applications in other areas. For instance, the computation of strong articulation points arise in constraint programming, and in particular in the tree constraint defined by Beldiceanu et al. [2]. The detection of the strong articulation points in a directed graph is indeed a crucial step in their filtering algorithm for the tree constraint, and Beldiceanu et al. [2] leaves as an open problem the design of a linear-time algorithm for computing all the strong articulation points of a directed graph. The algorithm presented in this paper solves exactly this problem.

Besides being interesting on their own, the algorithms presented in this paper have further applications. A first application is verifying the restricted edge connectivity of strongly connected graphs, as defined by Volkman in [21]: our algorithm is able to improve the bounds in [21] from $O(m(m + n))$ to $O(n(m + n))$. As a further application, we cite that our method is able to simplify substantially the approach of Georgiadis [14] to test the 2-vertex connectivity of a strongly connected graph and it also provides a more practical linear-time algorithm for this problem. Our algorithms can also be used to speed up the computation of matrix determinants: indeed, in the case of the adjacency matrix of a directed graph G, the computation of its determinant can be sped up if G has strong articulation points or strong bridges [3,17].

The remainder of this paper is organized as follows. Section 2 introduces few preliminary definitions and proves some basic properties on strong articulation points and strong bridges. In Section 3 we analyze the relationship between strong articulation points, strong bridges and dominators in flowgraphs. In Sections 4 and 5 we show how to exploit effectively this relationship and present, respectively, our linear-time algorithms for computing the strong bridges and the strong articulation points of a directed graph, whilst in Section 6 we consider some applications. Finally, Section 7 contains some concluding remarks.

2. Preliminaries

In this section we prove some properties about strong bridges and strong articulation points. Throughout the paper, we assume that we need to compute strong bridges and strong articulation points only in the case of strongly connected graphs. This is without loss of generality, since the strong bridges (respectively strong articulation points) of a directed graph G are given by the union of the strong bridges (respectively strong articulation points) of the strongly connected components of G.

We start with some technical lemmas. Let $G = (V, E)$ be a directed graph. Let $S \subseteq E$ be a subset of edges of G: we denote by $G \setminus S$ the graph obtained after deleting from G all the edges in S. Let $X \subseteq V$ be a subset of vertices of G: we denote by $G \setminus X$ the graph obtained after deleting from G all the vertices in X together with their incident edges.

Lemma 2.1. *Let $G = (V, E)$ be a strongly connected graph, and let $v \in V$ be a vertex of G. Then v is a strong articulation point in G if and only if there exist vertices x and y in G, $x \neq v$, $y \neq v$, such that all the paths from x to y in G contain vertex v.*

Proof. Assume that v is a strong articulation point: then $G \setminus \{v\}$ is not strongly connected. This implies that there must be two vertices x and y, $x \neq v$, $y \neq v$, such that there is no path from x to y in $G \setminus \{v\}$, which is equivalent to saying that all the paths from x to y in G must contain vertex v. Conversely, assume that there exist vertices x and y in G, $x \neq v$, $y \neq v$, such that all the paths from x to y in G contain vertex v. Removing v from G leaves no path from x to y, and thus $G \setminus \{v\}$ is no longer strongly connected. This implies that v must be an articulation point of G. □

Lemma 2.2. *Let $G = (V, E)$ be a strongly connected graph, and let $(u, v) \in E$ be an edge of G. Then (u, v) is a strong bridge in G if and only if there exist vertices x and y in G such that all the paths from x to y in G contain edge (u, v).*

Proof. Similar to the proof of Lemma 2.1: assume that (u, v) is a strong bridge, then $G \setminus \{(u, v)\}$ is not strongly connected; therefore there must be two vertices x and y, such that there is no path between x and y in $G \setminus \{(u, v)\}$, that is equivalent to saying that all the paths from x to y in G must contain edge (u, v). Conversely, assume that there are two vertices x and y such that all the paths from x to y in G contain the edge (u, v); thus, removing edge (u, v) from G leaves no path from x to y, and therefore $G \setminus \{(u, v)\}$ is no longer strongly connected, i.e. (u, v) is a strong bridge of G. □

Note that as a special case of Lemma 2.2 we can have $x = u$ and $y = v$. We now need the following definition.

Definition 2.1. Given a directed graph $G = (V, E)$, we say that an edge (u, v) is *redundant* if there is an alternative path from vertex u to vertex v avoiding edge (u, v). Otherwise, we say that (u, v) is *non-redundant*.

The following corollary is an immediate consequence of Lemma 2.2.

Corollary 2.3. *Let $G = (V, E)$ be a strongly connected graph. Then the edge $(u, v) \in E$ is a strong bridge if and only if (u, v) is non-redundant in G.*

By Corollary 2.3, in a strongly connected graph the problem of finding strong bridges is equivalent to the problem of finding redundant edges. We remark that finding all the redundant edges in a directed acyclic graph is essentially the transitive reduction problem. This is equivalent to the problem of computing the transitive closure [1], which is known to be equivalent to Boolean matrix multiplication [10,11,18]. Thus, while for directed acyclic graphs the best known bound for computing redundant edges is $O(n^{\omega})$, where ω is the exponent of the fastest matrix multiplication algorithm (currently

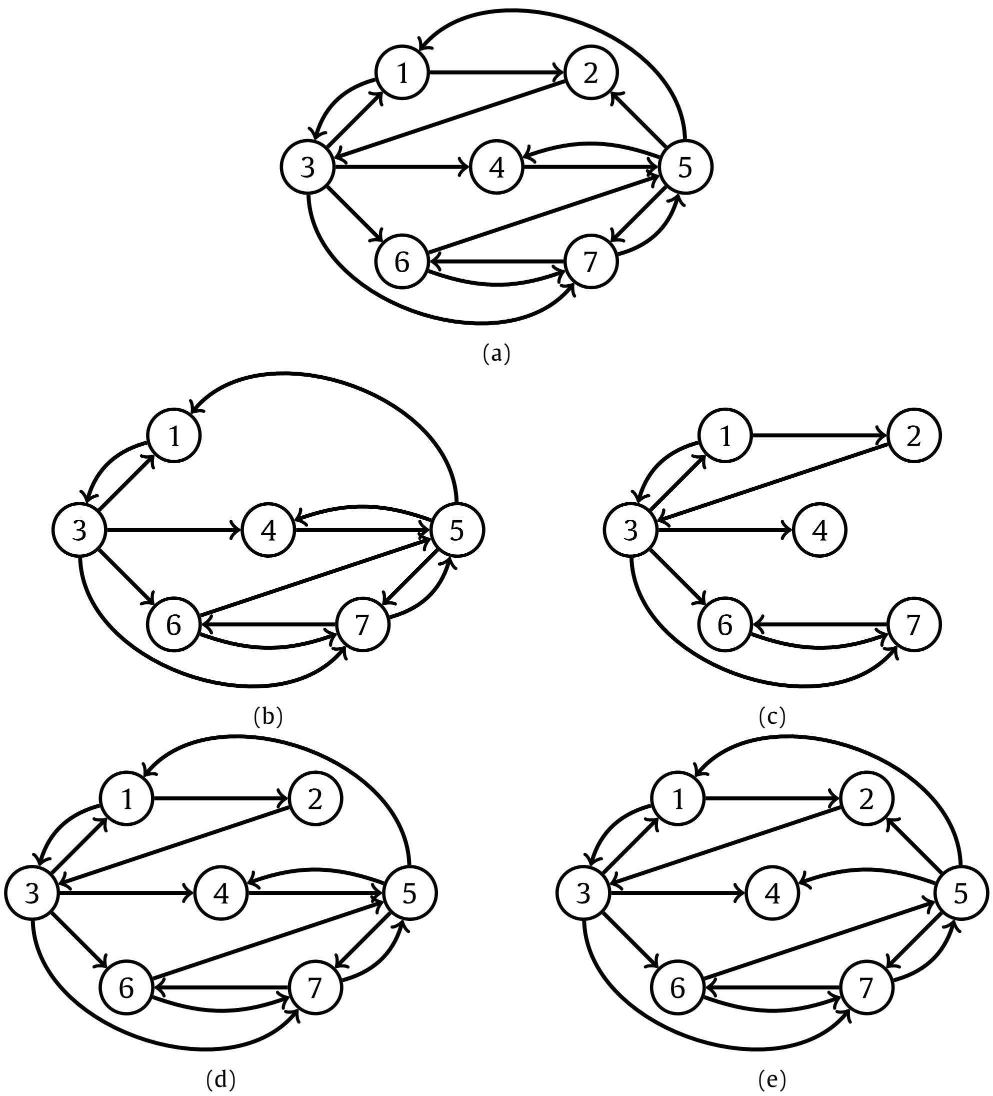

Fig. 1. (a) A strongly connected graph $G = (V, E)$. (b) After removing vertex 2, G is still strongly connected, so vertex 2 is **not** a strong articulation point, while (c) after removing vertex 5, G is no longer strongly connected, so vertex 5 is a strong articulation point. (d) After removing edge (5, 2), G is still strongly connected, so edge (5, 2) is **not** a strong bridge, while (e) after removing edge (4, 5), G is no longer strongly connected, so edge (4, 5) is a strong bridge. In summary, the strong articulation points in G are vertices 3 and 5, and the strong bridges in G are edges (2, 3) and (4, 5).

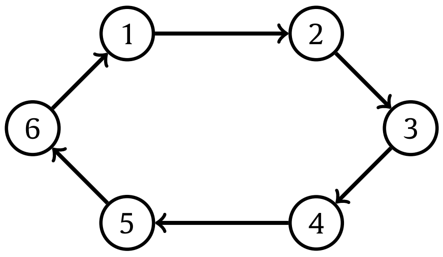

Fig. 2. A graph with n vertices and n strong articulation points.

$\omega < 2.376$), we show in this paper that for strongly connected graphs all the redundant edges can be computed faster, in optimal $O(m + n)$ time.

A directed graph can have at most n strong articulation points. This bound is realized by the graph consisting of a simple cycle: indeed in this graph each vertex is a strong articulation point (see Fig. 2).

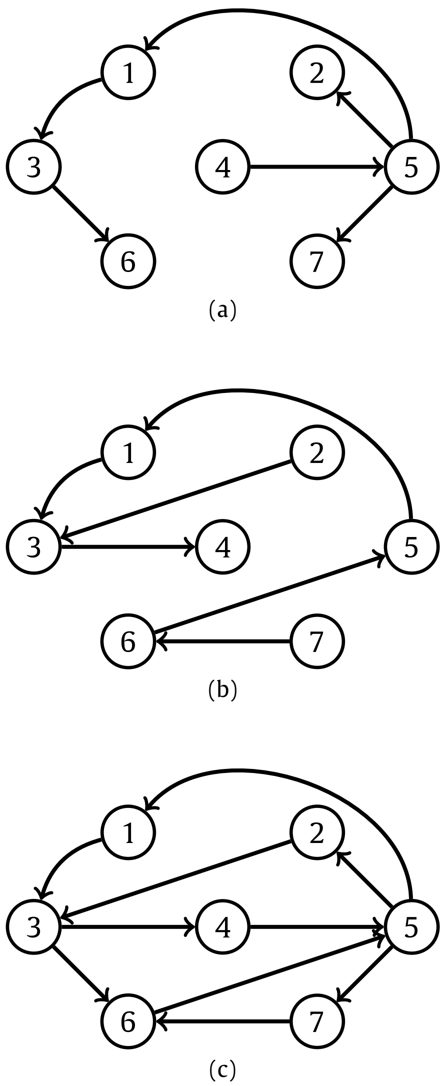

Fig. 3. Let G be the graph shown in Fig. 1a. (a) An out-branching $T^+(4)$ of G. (b) An in-branching $T^-(4)$ of G. (c) The graph $G' = T^+(4) \cup T^-(4)$; note that G' is strongly connected, and thus it includes all the strong bridges of G.

To bound the number of strong bridges in a directed graph, we need a different argument. Let $G = (V, E)$ be a strongly connected graph, and fix any vertex v of G. Let $T^+(v)$ to be an out-branching rooted at v, i.e., a directed spanning tree rooted at v with all edges directed away from v. Similarly, let $T^-(v)$ to be an in-branching rooted at v, i.e., a directed spanning tree rooted at v with all edges directed toward v.

Lemma 2.4. *The graph $G' = T^+(v) \cup T^-(v)$ contains at most $(2n - 2)$ edges, can be computed in $O(m + n)$ time and includes all the strong bridges of G.*

Proof. G' is the union of two branchings, each having exactly $(n - 1)$ edges: this gives immediately the bound on the size of G'. G' can be computed in linear time using either depth-first or breadth-first search. We now show that all the strong bridges of G must be contained in G'. For any given pair of vertices x, y in V, there is a directed path from x to y in G': the in-branching $T^-(v)$ contains a path from x to v, and the out-branching $T^+(v)$ contains a path from v to y. Thus, G' is a strongly connected subgraph of G, and therefore all the edges not in G' are redundant and cannot be strong bridges. □

An example of a graph $G' = T^+(v) \cup T^-(v)$, that refers to the graph G shown in Fig. 1a, is depicted in Fig. 3. The following corollary is an immediate consequence of Lemma 2.4:

Corollary 2.5. *A directed graph G can have at most $(2n - 2)$ strong bridges.*

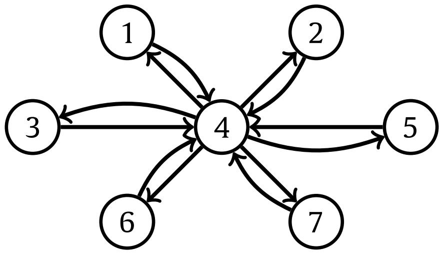

Fig. 4. A graph with n vertices and $2(n-1)$ strong bridges.

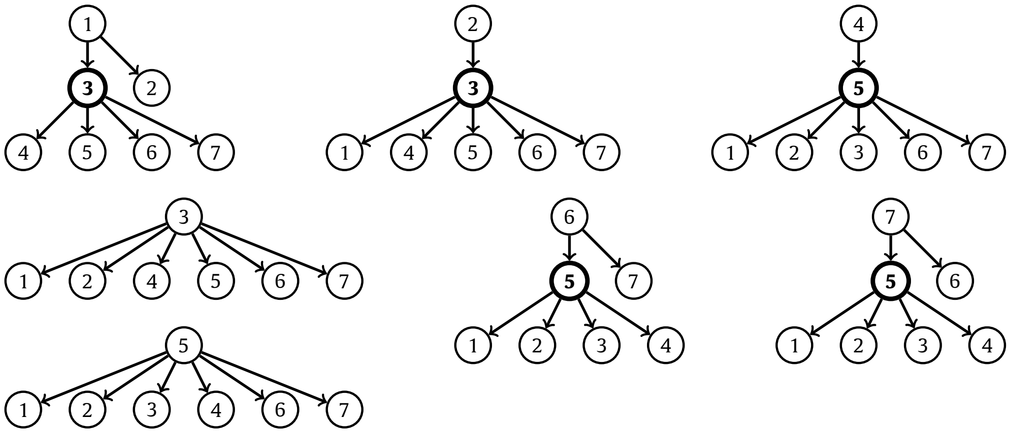

Fig. 5. Dominator trees of the flowgraphs relative to the graph of Fig. 1a. Non-trivial dominators are shown in bold.

The bound given in Corollary 2.5 is tight, as it is easy to construct graphs having exactly $(2n-2)$ strong bridges (see Fig. 4). Lemma 2.4 gives immediately a simple $O(n(m+n))$ algorithm to compute strong bridges: first compute the subgraph $G' = T^+(v) \cup T^-(v)$ in $O(m+n)$ time; since all the strong bridges of G are contained in G', for each edge (u, v) in G' test whether (u, v) is a strong bridge by computing the strongly connected components of $G \setminus \{(u, v)\}$.

3. Strong articulation points, strong bridges and dominators

Our linear-time algorithms for computing strong articulation points and strong bridges exploits a connection between strong articulation points, strong bridges and dominators in flowgraphs. We start with few definitions, and next we show how those notions are related.

A flowgraph $G(s) = (V, E, s)$ is a directed graph with a *start vertex* $s \in V$ such that every vertex in V is reachable from s. The *dominance relation* in $G(s)$ is defined as follows: a vertex u is a *dominator* of vertex v if every path from vertex s to vertex v contains vertex u. Let $dom(v)$ be the set of dominators of v. Clearly, $dom(s) = \{s\}$ and for any $v \neq s$ we have that $\{s, v\} \subseteq dom(v)$: we say that s and v are the *trivial dominators* of v in the flowgraph $G(s)$. The dominance relation is transitive and its transitive reduction is referred to as the *dominator tree* $DT(s)$. Note that the dominator tree $DT(s)$ is rooted at vertex s. Furthermore, vertex u dominates vertex v if and only if u is an ancestor of v in $DT(s)$. We say that u is an *immediate dominator* of v if u is a dominator of v, and every other non-trivial dominator of v also dominates u. It is known that if a vertex v has any dominators, then v has a unique immediate dominator: the immediate dominator of v is the parent of v in the dominator tree $DT(s)$.

Let $G = (V, E)$ be a strongly connected graph, and let s be any vertex in G. Since G is strongly connected, every vertex of V is reachable from s: thus for every vertex $s \in V$, $G(s) = (V, E, s)$ is a flowgraph. Note that there are n flowgraphs for each strongly connected graph. As an example, Fig. 5 shows the dominator trees of the flowgraphs relative to the graph of Fig. 1a. The following lemmas show a close relationship between strong articulation points in strongly connected graphs and non-trivial dominators in flow graphs.

Lemma 3.1. *Let $G = (V, E)$ be a strongly connected graph, and let s be any vertex in G. Let $G(s) = (V, E, s)$ be the flowgraph with start vertex s. If u is a non-trivial dominator of a vertex v in $G(s)$, then u is a strong articulation point in G.*

Proof. If u is a non-trivial dominator of v in the flowgraph $G(s) = (V, E, s)$, then $u \neq s$, $u \neq v$ and all the paths in G from s to v must include u. Consequently, $G \setminus \{u\}$ is not strongly connected and thus u must be a strong articulation point in G. □

Lemma 3.2. *Let $G = (V, E)$ be a strongly connected graph. If u is a strong articulation point in G, then there must be a vertex $s \in V$ such that u is a non-trivial dominator of a vertex v in the flowgraph $G(s) = (V, E, s)$.*

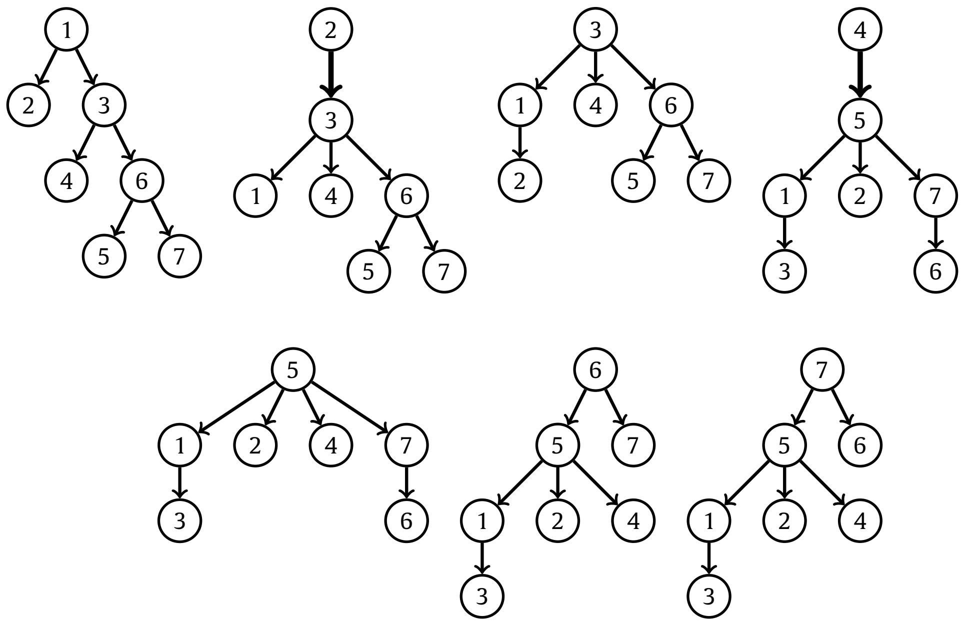

Fig. 6. Out-branchings of the flowgraphs relative to the graph of Fig. 1a. Edge dominators are shown in bold.

Proof. If u is a strong articulation point of G, then by Lemma 2.1 there must be two vertices s and v in G, $s \neq u$, $v \neq u$, such that every path from s to v contains vertex u. This implies that u must be a non-trivial dominator of vertex v in the flowgraph $G(s)$. □

We observe that Lemmas 3.1 and 3.2 are still not sufficient to achieve a linear-time algorithm for our problem: indeed, to compute all the strong articulation points of a strongly connected graph G, we need to compute all the non-trivial dominators in the flowgraphs $G(s)$, for each vertex s in V. Since the dominators of a flowgraph can be computed in $O(m + n)$ time [5] and there are exactly n flowgraphs to be considered, the running time of this algorithm is $O(n(m+n))$. In the next sections, we will show how a more careful exploitation of the relationship between strong articulation points and dominators yields a linear-time algorithm for computing the strong articulation points of a directed graph.

We now show how to exploit similar properties for strong bridges. We say that an edge (u, v) is an *edge dominator* of vertex w if every path from vertex s to vertex w contains edge (u, v). Furthermore, if (u, v) is an edge dominator of w, and every other edge dominator of w also dominates u, we say that (u, v) is an *immediate edge dominator* of w. Similar to the notion of dominators, if a vertex has any edge dominators, then it has a unique immediate edge dominator. As an example, Fig. 6 shows the out-branching trees of the flowgraphs relative to the graph of Fig. 1a. The following lemmas are completely analogous to Lemmas 3.1 and 3.2.

Lemma 3.3. *Let $G = (V, E)$ be a strongly connected graph, and let s be any vertex in G. Let $G(s) = (V, E, s)$ be the flowgraph with start vertex s. If (u, v) is an edge dominator in $G(s)$, then (u, v) is a strong bridge in G.*

Lemma 3.4. *Let $G = (V, E)$ be a strongly connected graph. If (u, v) is a strong bridge in G, then there must be a vertex $s \in V$ such that (u, v) is an edge dominator in the flowgraph $G(s) = (V, E, s)$.*

Similar to dominators, also the edge dominators of a flow graph $G(s) = (V, E, s)$ can be computed in linear time. This can be done by finding two out-branchings rooted at s having only the edge dominators in common.[1] To find two such out-branchings in linear time one can use the algorithm of Tarjan [20] with the linear-time disjoint-set union algorithm of Gabow and Tarjan [13].

As previously explained for the case of dominators and strong articulation points, Lemmas 3.3 and 3.4 are still not sufficient to obtain a linear-time algorithm for computing all the strong bridges of a strongly connected graph G, since we would need to compute the edge dominators for all n flowgraphs of a given strongly connected graph. We will show in the next section how to obtain this goal by exploiting more carefully the relationship between strong bridges and edge dominators.

[1] This is related to Edmonds' Theorem [7], which states that a k-connected directed graph admits k edge-disjoint out-branchings rooted at a fixed vertex r.

4. Finding all strong bridges

In this section we present a linear-time algorithm for computing all the strong bridges of a directed graph G. We recall from Section 2 that it is enough to restrict our attention to the case where G is strongly connected. Given a directed graph $G = (V, E)$, define its *reversal graph* $G^R = (V, E^R)$ by reversing all edges of G: namely, G^R has the same vertex set as G and for each edge (u, v) in G there is an edge (v, u) in G^R. We say that the edge (v, u) in G^R is the *reversal* of edge (u, v) in G. The following lemma is immediate.

Lemma 4.1. *Let $G = (V, E)$ be a strongly connected graph and $G^R = (V, E^R)$ be its reversal graph. Then G^R is strongly connected. Furthermore, edge (u, v) is a strong bridge in G if and only if its reversal (v, u) is a strong bridge in G^R.*

Let $G = (V, E)$ be a strongly connected graph, let $s \in V$ be any vertex in G, and let $G(s) = (V, E, s)$ be the flowgraph with start vertex s. We denote by $DE(s)$ the set of edge dominators in $G(s)$. Similarly, let $G^R = (V, E^R)$ be the reversal graph of G, and let $G^R(s) = (V, E^R, s)$ be the flowgraph with start vertex s. We denote by $DE^R(s)$ the set of edge dominators in $G^R(s)$. The following theorem provides a characterization of strong bridges in terms of the edge dominators in the two flowgraphs $G(s) = (V, E, s)$ and $G^R(s) = (V, E^R, s)$.

Theorem 4.2. *Let $G = (V, E)$ be a strongly connected graph, and let $s \in V$ be any vertex in G. Then edge (u, v) is a strong bridge in G if and only if $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$.*

Proof. We first prove that if (u, v) is a strong bridge in G, then we must have $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$. Assume not: namely, assume that (u, v) is a strong bridge in G, $v \neq s$, but $(u, v) \notin DE(s)$ and $(v, u) \notin DE^R(s)$. Since v is a strong bridge in G, then $G \setminus \{(u, v)\}$ is not strongly connected. As a consequence, there must be a vertex w in G, $w \neq s$, such that the following is true: w is in the same strongly connected component as s in G, but w is not the same strongly connected component as s in $G \setminus \{(u, v)\}$. Namely, either (a) there is a path from s to w in G, but there is no path from s to w in $G \setminus \{(u, v)\}$, or (b) there is a path from w to s in G, but there is no path from w to s in $G \setminus \{(u, v)\}$. If we are in case (a), then all the paths from s to w in G must contain edge (u, v). This is equivalent to saying that (u, v) is an edge dominator of w in the flowgraph $G(s) = (V, E, s)$, i.e., $(u, v) \in DE(s)$, which clearly contradicts the assumption $(u, v) \notin DE(s)$ and $(v, u) \notin DE^R(s)$. If we are in case (b), then all the paths from w to s in G must contain edge (u, v). This is equivalent to saying that (v, u) is an edge dominator of w in the flowgraph $G^R(s) = (V, E^R, s)$, i.e., $(v, u) \in DE^R(s)$, which again contradicts the assumption $(u, v) \notin DE(s)$ and $(v, u) \notin DE^R(s)$. This shows that if (u, v) is a strong bridge in G, then (u, v) must be in $DE(s)$ or (v, u) must be in $DE^R(s)$.

To prove the converse, let (u, v) be any edge such that $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$. If $(u, v) \in DE(s)$, (u, v) is an edge dominator in $G(s)$, and thus (u, v) must be a strong bridge in G by Lemma 3.3. Analogously, if $(v, u) \in DE^R(s)$, again by Lemma 3.3 (v, u) must be a strong bridge in G^R: Lemma 4.1 now ensures that (u, v) must be a strong bridge in G as well. This completes the proof of the theorem. □

We now present our algorithm for computing the strong bridges of a strongly connected graph.

Algorithm StrongBridges(G)

Input : A strongly connected graph $G = (V, E)$, with n vertices and m edges.

Output : The strong bridges of G.

```
1. Choose arbitrarily a vertex $s \in V$ in $G$.
2. Compute $DE(s)$, the set of edge dominators in the flowgraph $G(s) = (V, E, s)$.
3. Compute the reversal graph $G^R = (V, E^R)$.
4. Compute $DE^R(s)$, the set of edge dominators in the flowgraph $G^R(s) = (V, E^R, s)$.
5. Output the union of the edges in $DE(s)$ and the reversal of the edges in $DE^R(s)$.
```

Theorem 4.3. *Algorithm* `StrongBridges` *computes the strong bridges of a strongly connected graph G in time $O(m + n)$.*

Proof. The correctness of the algorithm hinges on Theorem 4.2. We now analyze its running time. Since the edge dominators of a flowgraph can be computed in linear time, Steps 2 and 4 can be implemented in time $O(m + n)$. Finally, the reversal graph G^R in Step 3 can be computed in linear time, and thus the theorem follows. □

Theorem 4.3 can be generalized to general directed graphs.

Theorem 4.4. *The strong bridges of a directed graph G can be computed in time $O(m + n)$.*

Proof. To compute the strong bridges of G it is enough to find the strongly connected components of G in $O(m+n)$ time [19], and then to return the strong bridges of each connected component of G. By Theorem 4.3, this require overall $O(m + n)$ time. □

5. Finding all strong articulation points

In the previous section we presented an algorithm to compute effectively all the strong bridges of a directed graph. Now we show how to adapt the same approach to compute all the strong articulation points. The following are the analogs of Lemma 4.1 and of Theorem 4.2.

Lemma 5.1. *Let $G = (V, E)$ be a strongly connected graph and $G^R = (V, E^R)$ be its reversal graph. Vertex v is a strong articulation point in G if and only if v is a strong articulation point in G^R.*

Theorem 5.2. *Let $G = (V, E)$ be a strongly connected graph, and let $s \in V$ be any vertex in G. Then vertex $v \neq s$ is a strong articulation point in G if and only if $v \in D(s) \cup D^R(s)$.*

Proof. This proof follows the line of the proof of Theorem 4.2. We first prove that if v is a strong articulation point in G, $v \neq s$, then we must have $v \in D(s) \cup D^R(s)$. Assume not: namely, assume that v is a strong articulation point in G, $v \neq s$, but $v \notin D(s)$ and $v \notin D^R(s)$. Since v is a strong articulation point in G, then $G \setminus \{v\}$ is not strongly connected. As a consequence, there must be a vertex w in G, $w \neq s$, such that the following is true: w is in the same strongly connected component as s in G, but w is not the same strongly connected component as s in $G \setminus \{v\}$. Namely, either (a) there is a path from s to w in G, but there is no path from s to w in $G \setminus \{v\}$, or (b) there is a path from w to s in G, but there is no path from w to s in $G \setminus \{v\}$. If we are in case (a), then all the paths from s to w in G must contain vertex v. This is equivalent to saying that v is a dominator of w in the flowgraph $G(s) = (V, E, s)$, i.e., $v \in D(s)$, which clearly contradicts the assumption $v \notin D(s)$ and $v \notin D^R(s)$. If we are in case (b), then all the paths from w to s in G must contain vertex v. This is equivalent to saying that v is a dominator of w in the flowgraph $G^R(s) = (V, E^R, s)$, i.e., $v \in D^R(s)$, which again contradicts the assumption $v \notin D(s)$ and $v \notin D^R(s)$. This shows that if v is a strong articulation point in G, then v must be in $D(s)$ or in $D^R(s)$.

To prove the converse, let v be any vertex such that $v \in D(s)$ or $v \in D^R(s)$. If $v \in D(s)$, v is a dominator in $G(s)$, and thus v must be a strong articulation point in G by Lemma 3.1. Analogously, if $v \in D^R(s)$, again by Lemma 3.1 v must be a strong articulation point in G^R: Lemma 5.1 now ensures that v must be a strong articulation point in G as well. This completes the proof of the theorem. □

Note that Theorem 5.2 provides no information on whether vertex s is a strong articulation point. However, this can be easily checked as shown in the following algorithm.

Algorithm StrongArticulationPoints(G)

Input : A strongly connected graph $G = (V, E)$, with n vertices and m edges.

Output : The strong articulation points of G.

1. Choose arbitrarily a vertex $s \in V$ in G, and test whether s is a strong articulation point in G. If s is an articulation point, output s.
2. Compute $D(s)$, the set of non-trivial dominators in the flowgraph $G(s) = (V, E, s)$.
3. Compute the reversal graph $G^R = (V, E^R)$.
4. Compute $D^R(s)$, the set of non-trivial dominators in the flowgraph $G^R(s) = (V, E^R, s)$.
5. Output $D(s) \cup D^R(s)$.

Theorem 5.3. *Algorithm* `StrongArticulationPoints` *computes the strong articulation points of a strongly connected graph G in time $O(m + n)$.*

Proof. The correctness of the algorithm hinges on Theorem 5.2. We now analyze its running time. Step 1 can be implemented in time $O(m + n)$ by simply computing the strongly connected components of $G \setminus \{s\}$ [19]. Since computing the dominators of a flowgraph requires linear time [4], also Steps 2 and 4 can be implemented in time $O(m + n)$. Finally, the reversal graph G^R in Step 3 can be computed in linear time, and thus the theorem follows. □

Theorem 5.3 can be generalized to general directed graphs:

Theorem 5.4. *The strong articulation points of a directed graph G can be computed in time $O(m + n)$.*

Proof. To compute the strong articulation points of G it is enough to find the strongly connected components of G in $O(m+n)$ time [19], and then to return the strong articulation points of each connected component of G. By Theorem 5.3, this require overall $O(m + n)$ time. □

We conclude this section by observing that it is possible to reduce the problem of computing strong bridges to the problem of computing strong articulation points.

Lemma 5.5. *If there is an algorithm to compute the strong articulation points of a strongly connected graph in time $T(m, n)$, then there is an algorithm to compute the strong bridges of a strongly connected graph in time $O(m + n + T(2m, n + m))$.*

Proof. Let $G = (V, E)$ be a strongly connected graph with m edges and n vertices. We define a new graph G' as follows. G' contains all the vertices of G; furthermore, for each edge $e = (u, v)$ in G, we introduce a new vertex $\varphi(e)$ in G'. The edge set of G' is defined as follows: for each edge $e = (u, v)$ in G, there are edges $(u, \varphi(e))$ and $(\varphi(e), v)$ in G'. Note that G' has exactly $n + m$ vertices and $2m$ edges, it is still strongly connected and can be computed in time $O(m + n)$. The lemma now follows from the observation that an edge e is a strong bridge in G if and only if the corresponding vertex $\varphi(e)$ is a strong articulation point in G'. □

By Lemma 5.5, a linear-time algorithm for computing the strong articulation points implies immediately a linear-time algorithm for computing the strong bridges. This provides an alternative algorithm to the one described in Section 4.

6. Applications of strong articulation points and strong bridges

In this section, we list a few other applications of our linear-time algorithms for computing strong articulation points and strong bridges.

Verifying restricted edge connectivity. A notion related to edge connectivity is *restricted edge connectivity*, originally introduced for undirected graphs by Esfahanian and Hakimi [9] in 1988. The restricted edge connectivity of an undirected graph G is the minimum cardinality over all edge-cuts S of G such that there are no isolated vertices in $G \setminus S$. Given a strongly connected graph G, we say that an edge set S is a restricted edge-cut of G if $G \setminus S$ has a non-trivial strongly connected component D such that $G \setminus V(D)$ contains (at least) one edge. The restricted edge connectivity $\lambda'(G)$ is the minimum cardinality over all restricted edge-cuts S. A strongly connected graph G is called λ'-connected if $\lambda'(G)$ exists. The restricted edge connectivity of undirected and directed graphs is an interesting measure of network reliability. In [21] Volkmann presented an $O(m(m + n))$ algorithm to verify the λ'-connectedness of a strongly connected graph G. The algorithm by Volkmann needs to carry out m distinct computations of strongly connected components, one for each edge of the graph. It is possible to reduce to $O(n)$ the total number of computations of strongly connected components required, i.e., one for each strong bridge of G (recall that the strong bridges of a directed graph are $O(n)$ by Corollary 2.5). This improves to $O(n(m+n))$ the overall time needed to verify the λ'-connectedness of a strongly connected graph.

Testing 2-vertex connectivity. Very recently Georgiadis [14] has given a linear-time algorithm for testing the 2-vertex connectivity of a strongly connected graph G. The algorithm in [14] work as follows: first, choose two distinct vertices a and b in G, and next check whether the four flowgraphs $G(a)$, $G(b)$, $G^R(a)$, and $G^R(b)$ have only trivial dominators. The proof of correctness given in [14] is rather involved and lengthy. Theorem 5.2 in this paper provides a much simpler proof: if $G(a)$ and $G^R(a)$ have only trivial dominators, then only vertex a could be an articulation point in G; if also the flowgraphs $G(b)$ and $G^R(b)$ have only trivial dominators, then a cannot be a strong articulation point, and hence G must be 2-vertex connected. Conversely, if G is 2-vertex connected, then G has no strong articulation points, and therefore, for any choice of the start vertex s, the flowgraphs $G(s)$ and $G^R(s)$ have only trivial dominators. We also observe that the algorithm presented in this paper is likely to be faster in practice than the algorithm in [14]. Indeed, while the algorithm in [14] needs to work with four different flowgraphs, the algorithm presented here needs only two different flowgraphs, plus one simple computation of the strongly connected components. We recall that, besides simplifying the correctness proofs and providing algorithms that are likely to be faster in practice, our algorithm is able to report all the cut vertices, i.e., all the strong articulation points in G.

The Tree Constraint. We conclude this section by mentioning the tree constraint defined by Beldiceanu et al. [2]: here the problem is to partition a directed graph into a set of vertex-disjoint anti-arborescences, where a directed graph A is an *anti-arborescence* with *anti-root* r if and only if there exists a path from all vertices of A to r and the undirected graph associated with A is a tree. One of the bottlenecks of this problem is the computation of the strong articulation points of a directed graph, which is exactly the problem solved in this paper.

7. Conclusions and open problems

Strong articulation points and strong bridges in directed graphs are a natural generalization of the notions of articulation points and bridges in undirected graphs, and appear to be interesting on their own. Surprisingly, they have not received much attention in the literature, despite the fact that they seem to arise in several applications (see, e.g., [2,17,21]). In this paper, we have presented linear-time algorithms for computing all the strong articulation points and all the strong bridges of a directed graph. Our algorithms are simple, and thus appear to be amenable to practical implementations.

Our work raises some additional and perhaps intriguing questions. Let G be a strongly connected directed graph. As in the case of undirected graphs, one could define the *2-vertex-connected components* of G as the maximal 2-vertex-connected subgraphs of G, i.e., the maximal strongly connected subgraphs of G that contain no strong articulation points (see Fig. 7). Note that this is completely analogous to the definition of 2-vertex-connected (or biconnected) components of undirected graphs. Similarly, one could define the *2-edge-connected components* of a strongly connected G as the maximal 2-edge-connected subgraphs of G, i.e., the maximal strongly connected subgraphs of G that contain no strong bridges (see Fig. 8). Can the 2-vertex- and the 2-edge-connected components of a strongly connected graph G be computed efficiently (i.e., in linear time) like their undirected counterparts? To the best of our knowledge, the best bound known in the literature for these problems is $\Omega(mn)$ [8,16].

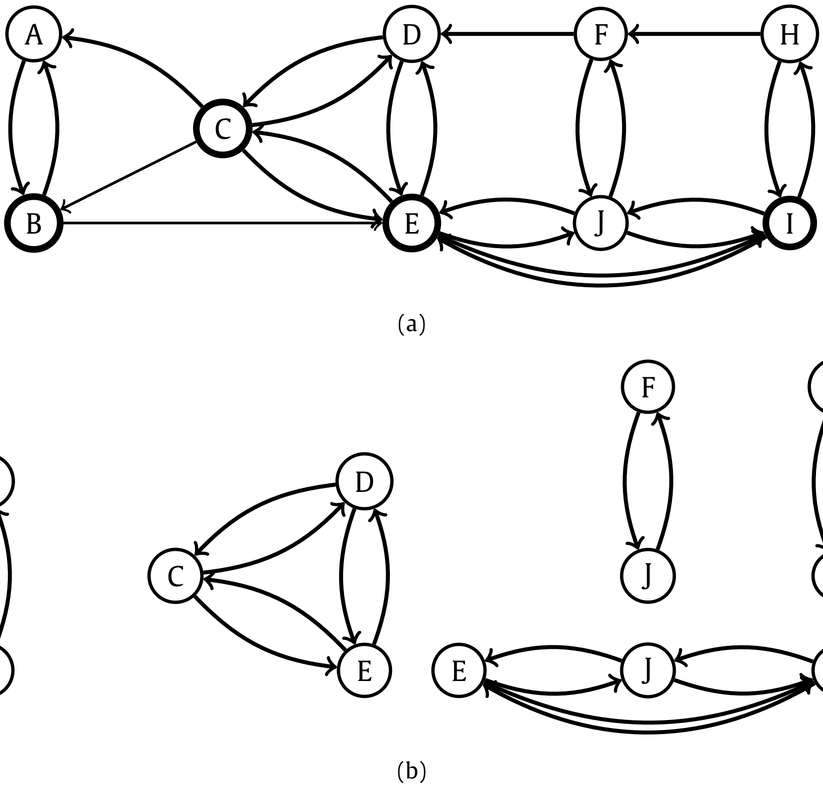

Fig. 7. (a) A strongly connected directed graph *G*, with strong articulation points shown in bold. (b) The 2-vertex-connected components of *G*.

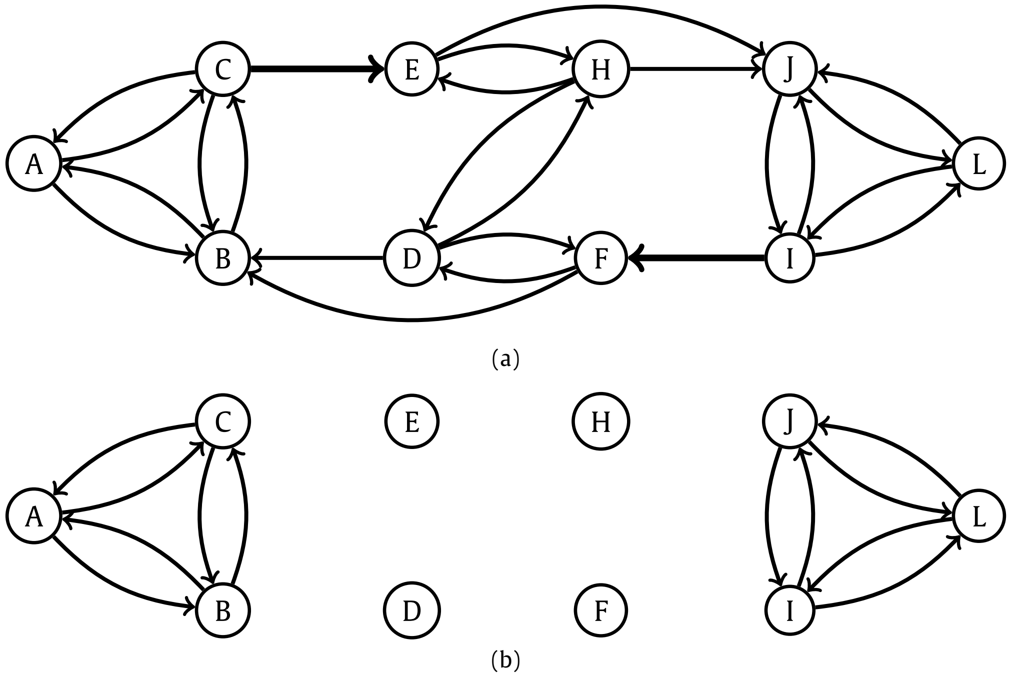

Fig. 8. (a) A strongly connected directed graph *G*, with strong bridges shown in bold. (b) The 2-edge-connected components of *G*.

We remark that 2-vertex- and 2-edge-connected components of strongly connected graphs seem to have rather different structural properties from their undirected counterparts. For the sake of argument, consider 2-vertex-connected components. First, in strongly connected graphs a strong articulation point does not necessarily occur in two or more 2-vertex-connected components (see vertex *C* in Fig. 7), while in undirected graphs an articulation point always occurs in two or more biconnected components. Second, in strongly connected graphs a vertex common to two different 2-vertex-connected components is not necessarily a strong articulation point (see vertex J in Fig. 7), while in undirected graphs a vertex common to two biconnected components is always an articulation point. Third, differently from the case of undirected graphs, in strongly connected graphs the 2-vertex-connected components do not partition the edge set.

Another problem that seems to deserve further study is the computation of higher connectivity cuts in strongly connected graphs. For instance, given a strongly 2-vertex-connected graph G (i.e., a strongly connected graph containing no strong articulation point), one could define a separation pair as a pair of vertices whose removal increases the number of strongly connected components of G. Similarly, given a strongly 2-edge-connected graph G (i.e., a strongly connected graph containing no strong bridge), one could define a separation edge pair as a pair of edges whose removal increases the number of strongly connected components of G. Can we compute efficiently all (vertex and edge) separation pairs of a strongly connected graph?

Acknowledgements

We are indebted to Nicolas Beldiceanu for suggesting this problem, and to Fabrizio Grandoni for initial discussions on this topic.

References

[1] Alfred V. Aho, Michael R. Garey, Jeffrey D. Ullman, The transitive reduction of a directed graph, SIAM J. Comput. 1 (2) (1972) 131–137.

[2] Nicolas Beldiceanu, Pierre Flener, Xavier Lorca, The tree constraint, in: Proc. 2nd Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2005, in: LNCS, vol. 3524, Springer-Verlag, 2005, pp. 64–78.

[3] Dario Bini, Victor Y. Pan, Polynomial and Matrix Computations (vol. 1): Fundamental Algorithms, Birkhauser Verlag, Basel, Switzerland, 1994.

[4] Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, Jeffery R. Westbrook, Linear-time algorithms for dominators and other path-evaluation problems, SIAM J. Comput. 38 (4) (2008) 1533–1573.

[5] Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, Jeffery R. Westbrook, A new, simpler linear-time dominators algorithm, ACM Trans. Program. Lang. Syst. 20 (6) (1998) 1265–1296.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.

[7] Jack Edmonds, Edge-disjoint branchings, in: Proceedings of the 9th Courant Computer Science Symposium, Combinatorial Algorithms, Algorithmics Press, 1972, pp. 91–96.

[8] Ya.M. Erusalimskii, G.G. Svetlov, Bijoin points, bibridges, and biblocks of directed graphs, Cybernetics and Systems Analysis 16 (1980) 41–44.

[9] Abdol-Hossein Esfahanian, S. Louis Hakimi, On computing a conditional edge-connectivity of a graph, Inf. Process. Lett. 27 (1988) 195–199.

[10] Michael J. Fischer, Albert R. Meyer, Boolean matrix multiplication and transitive closure, in: Proceedings of 12th FOCS, IEEE, 1971, pp. 129–131.

[11] M.E. Furman, Applications of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph, Soviet Math. Dokl. 11 (5) (1970) 1252.

[12] Harold N. Gabow, A matroid approach to finding edge connectivity and packing arborescences, J. Comput. Syst. Sci. 50 (2) (1995) 259–273.

[13] Harold N. Gabow, Robert E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. System Sci. 30 (2) (1985) 209–221.

[14] Loukas Georgiadis, Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs, in: ICALP 10: Proceedings of the 37th International Colloquium on Automata, Languages and Programming, 2010, pp. 433–442.

[15] Giuseppe F. Italiano, Luigi Laura, Federico Santaroni, Finding strong bridges and strong articulation points in linear time, in: Proceedings of the 4th International Conference on Combinatorial Optimization and Applications, COCOA 2010, in: Lecture Notes in Computer Science, vol. 6508, Springer, 2010, pp. 157–169.

[16] David W. Matula, Rakesh. V. Vohra, Calculating the connectivity of a directed graph, Technical Report 386, Institute for Mathematics and Application, University of Minnesota, 1988.

[17] J.S. Maybee, D.D. Olesky, P. van den Driessche, G. Wiener, Matrices, digraphs, and determinants, SIAM J. Matrix Anal. Appl. 10 (1989) 500–519.

[18] Ian Munro, Efficient determination of the transitive closure of a directed graph, Inform. Process. Lett. 1 (2) (1971) 56–58.

[19] Robert E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.

[20] Robert E. Tarjan, Edge-disjoint spanning trees and depth-first search, Acta Inf. 6 (1976) 171–185.

[21] Lutz Volkmann, Restricted arc-connectivity of digraphs, Inf. Process. Lett. 103 (6) (2007) 234–239.