Finding Strong Bridges and Strong Articulation Points in Linear Time*

Giuseppe F. Italiano[1], Luigi Laura[2], and Federico Santaroni[3]

[1] Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata", via del Politecnico 1, 00133 Roma, Italy
italiano@disp.uniroma2.it

[2] Dipartimento di Informatica e Sistemistica
"Sapienza" Università di Roma, via Ariosto 25, 00185, Roma, Italy
laura@dis.uniroma1.it

[3] Università di Roma "Tor Vergata", 00133 Roma, Italy
federico.santaroni@gmail.com

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 [2].

1 Introduction

We assume that the reader is familiar with the standard graph terminology, as contained for instance in [5]. 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, ..., v_k, such that edge $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \dots, k-1$. 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, a vertex $v \in V$ is a *strong articulation point* if its removal increases the number of strongly connected components of G. Similarly, we say that an edge $e \in E$ is a *strong bridge* if its removal increases the number of strongly connected components of G. Figure 1 illustrates the notion of strong articulation points and strong bridges of a directed graph.

The notions of strong articolation 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

* 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.

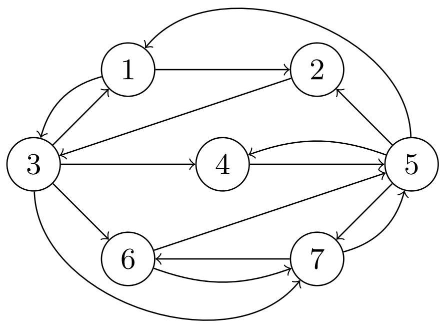

Fig. 1. A strongly connected graph G. Vertices 3 and 5 are strong articulation points in G, while edges $(2, 3)$ and $(4, 5)$ are strong bridges in G

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 [11]. 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 [9]: one can test whether a directed graph is 2-edge-connected by using Tarjan's algorithm [14] to compute two edge-disjoint spanning trees in combination with the disjoint set-union algorithm of Gabow and Tarjan [10]. 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.

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 algoritms 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 [15]: our algorithm is able to improve the bounds in [15] 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 [11] to test the 2-vertex connectivity of a strongly connected graph and it also provides a more practical linear-time algorithm for this problem.

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 Section 4 and Section 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. Some details are omitted from this extended abstract for lack of space.

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 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. *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 1. □

Note that as a special case of Lemma 2 we can have $x = u$ and $y = v$. We now need the following definition.

Definition 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 lemma is a consequence of Lemma 2.

Lemma 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 Lemma 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 [7,8,12]. 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 $\omega < 2.376$), 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. 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 towards v.

Lemma 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 at most $(n - 1)$ 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. □

The following corollary is an immediate consequence of Lemma 4:

Corollary 1. *A directed graph G can have at most $(2n-2)$ strong bridges.*

The bound given in Corollary 1 is tight, as it is easy to construct graphs having exactly $(2n-2)$ strong bridges. Lemma 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 $DT(s)$. If u is a dominator of v, and every other dominator of u also dominates v, we say that u is an *immediate dominator* of v. 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, Figure 2 shows the dominator trees of the flowgraphs relative to the graph of Figure 1. The following lemmas show a close relationship between strong articulation points in strongly connected graphs and non-trivial dominators in flow graphs.

Lemma 5. *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. □

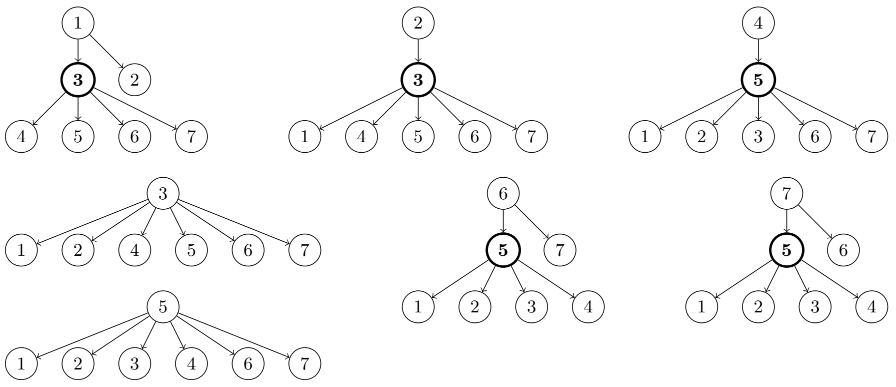

Fig. 2. Dominator trees of the flowgraphs relative to the graph of Figure 1. Non-trivial dominators are shown in bold.

Lemma 6. *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 $G(s) = (V, E, s)$.*

Proof. If u is a strong articulation point of G, then by Lemma 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 5 and 6 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 [4] 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 a *dominator edge* of vertex w if all 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 u also dominates w, 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, Figure 3 shows the out-branching trees of the flowgraphs relative to the graph of Figure 1. The following lemmas are completely analogous to Lemmas 5 and 6.

Lemma 7. *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 a dominator edge in $G(s)$, then (u, v) is a strong bridge in G.*

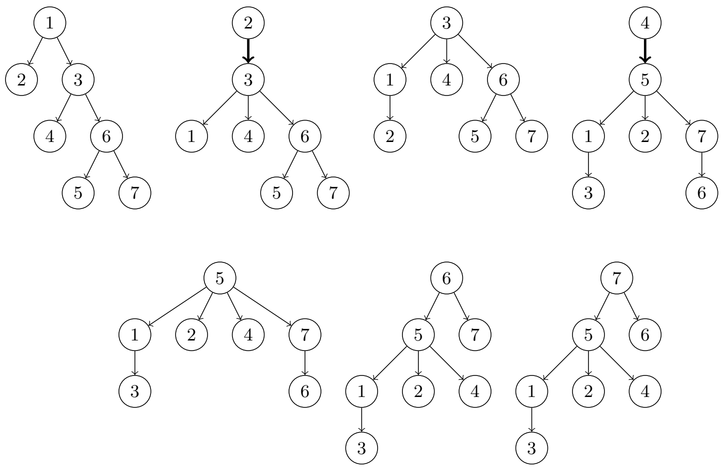

Fig. 3. Out-branchings of the flowgraphs relative to the graph of Figure 1. Dominator edges are shown in bold.

Lemma 8. *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 a dominator edge in the flowgraph $G(s) = (V, E, s)$.*

Similar to dominators, also the dominator edges of a flow graph can be computed in linear time. To do this, we compute two edge-disjoint out-branchings rooted at a fixed vertex s with the algorithm of Tarjan [14]: by definition, the dominator edges of the flowgraph $G(s)$ are all the edges that appear in both out-branchings[1]. Tarjan's algorithm is able to identify the dominator edges before building the out-branchings, and can be made to run in time $O(m + n)$ with the help of the linear time disjoint-set union algorithm of Gabow and Tarjan [10].

As previously explained for the case of dominators and strong articulation points, Lemmas 7 and 8 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 dominator edges.

4 Finding Strong Bridges

In this section we present a linear-time algorithm for computing the strong bridges of a directed graph G. We recall from Section 2 that it is enough to

[1] This result can be seen also as a consequence of Edmond's Theorem [6], that states that a k-connected directed graph admits k edge-disjoint out-branchings rooted at a fixed vertex r.

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 9. *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 dominator edges 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 dominator edges in $G^R(s)$. The following theorem provides a characterization of strong bridges in terms of the dominator edges in the two flowgraphs $G(s) = (V, E, s)$ and $G^R(s) = (V, E^R, s)$.

Theorem 1. *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, $v \neq s$, 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 a dominator edge of w in the flowgraph $G(s) = (V, E, s)$, i.e., $(u, v) \in DE(s)$, which clearly contradicts the assumption $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$ 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 a dominator edge 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 $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$. 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 a dominator edge in $G(s)$, and thus (u, v) must be a strong bridge in G by Lemma 7. Analogously, if $(v, u) \in DE^R(s)$, again by Lemma 7 (v, u) must be a strong bridge in G^R: Lemma 9 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 dominator edges 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 dominator edges 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 2. *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 1. 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 2 can be generalized to general directed graphs:

Theorem 3. *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 [13], and then to return the strong bridges of each connected component of G. By Theorem 2, this require overall $O(m + n)$ time. □

5 Finding Strong Articulation Points

In the previous section we have seen how to compute the strong bridges of a directed graph. Now we show that to adapt the same approach to compute the strong articulation points. The following are the analogs of Lemma 9 and of Theorem 1.

Lemma 10. *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 4. *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 $v \in D(s) \cup D^R(s)$.*

Note that Theorem 4 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. *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 4. We now analyze its running time. Step 1 can be implemented in time $O(m + n)$ by simply computing the strongly connected components of $G \setminus \{s\}$ [13]. Since computing the dominators of a flowgraph requires linear time [3], 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 can be generalized to general directed graphs:

Theorem 6. *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 [13], and then to return the strong articulation points of each connected component of G. By Theorem 5, this require overall $O(m + n)$ time. □

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

Lemma 11. *If there is an algorithm to compute the strong articulation points of a strongly connected graph in time $T(m, n)$, then there is 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 11, 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

Strong articulation points and strong bridges are natural concepts in graph theory, and appear to be interesting on their own. 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* [15]. Let G be a strongly connected graph: 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. In [15] 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 1). 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 [11] gave a linear-time algorithm for testing the 2-vertex connectivity of a strongly connected graph G. The algorithm in [11] 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 [11] is rather involved and lengthy. Theorem 4 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 [11]. Indeed, while the algorithm in [11] 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.

7 Conclusions

Strong articulation points and strong bridges in directed graphs are a natural generalization of the notions of articulation points and bridges in undirected graphs. Surprisingly, they have not received much attention in the literature, despite the fact that they seem to arise in several applications (see, e.g., [2,15]). 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.

Acknowledgments

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

References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. 1(2), 131–137 (1972)
2. Beldiceanu, N., Flener, P., Lorca, X.: The tree constraint. In: Barták, R., Milano, M. (eds.) CPAIOR 2005. LNCS, vol. 3524, pp. 64–78. Springer, Heidelberg (2005)
3. Buchsbaum, A.L., Georgiadis, L., Kaplan, H., Rogers, A., Tarjan, R.E., Westbrook, J.R.: Linear-time algorithms for dominators and other path-evaluation problems. SIAM Journal on Computing 38(4), 1533–1573 (2008)
4. Buchsbaum, A.L., Kaplan, H., Rogers, A., Westbrook, J.R.: A new, simpler linear-time dominators algorithm. ACM Trans. Program. Lang. Syst. 20(6), 1265–1296 (1998)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
6. Edmonds, J.: Edge-disjoint branchings. In: Proceedings of the 9th Courant Computer Science Symposium, Combinatorial Algorithms, pp. 91–96. Algorithmics Press (1972)
7. Fischer, M.J., Meyer, A.R.: Boolean matrix multiplication and transitive closure. In: Proceedings of 12th FOCS, pp. 129–131. IEEE, Los Alamitos (1971)

8. Furman, M.E.: 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), 1252 (1970)
9. Gabow, H.N.: A matroid approach to finding edge connectivity and packing arborescences. J. Comput. Syst. Sci. 50(2), 259–273 (1995)
10. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. Journal of Computer and System Sciences 30(2), 209–221 (1985)
11. Georgiadis, L.: Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. In: ICALP 2010: Proceedings of the 37th International Colloquium on Automata, Languages and Programming, pp. 433–442 (2010)
12. Munro, I.: Efficient determination of the transitive closure of a directed graph. Inform. Process. Lett. 1(2), 56–58 (1971)
13. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
14. Tarjan, R.E.: Edge-disjoint spanning trees and depth-first search. Acta Inf. 6, 171–185 (1976)
15. Volkmann, L.: Restricted arc-connectivity of digraphs. Inf. Process. Lett. 103(6), 234–239 (2007)