2-Connectivity in Directed Graphs

Loukas Georgiadis[1], Giuseppe F. Italiano*[2], and Nikos Parotsidis[3]

1 **University of Ioannina, Ioannina, Greece**
loukas@cs.uoi.gr
2 **University of Rome Tor Vergata, Rome, Italy**
giuseppe.italiano@uniroma2.it
3 **University of Rome Tor Vergata, Rome, Italy**
nikos.parotsidis@uniroma2.it

Abstract

We survey some recent results on 2-edge and 2-vertex connectivity problems in directed graphs. Despite being complete analogs of the corresponding notions on undirected graphs, in digraphs 2-vertex and 2-edge connectivity have a much richer and more complicated structure. It is thus not surprising that 2-connectivity problems on directed graphs appear to be more difficult than on undirected graphs. For undirected graphs it has been known for over 40 years how to compute all bridges, articulation points, 2-edge- and 2-vertex-connected components in linear time, by simply using depth-first search. In the case of digraphs, however, the very same problems have been much more challenging and required the development of new tools and techniques.

1998 ACM Subject Classification E.1 [Graphs and Networks] Trees, F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, G.2.2 [Graph Algorithms] Trees

Keywords and phrases 2-edge and 2-vertex connectivity on directed graphs, graph algorithms, dominator trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.1

Category Invited Talk

1 Preliminaries

Let $G = (V, E)$ be an *undirected* (resp., *directed*) graph, with m edges and n vertices. Throughout the paper, we use interchangeably the term directed graph and digraph. Edge and vertex connectivity are fundamental concepts in graph theory with numerous practical applications [2, 32]. As an example, we mention the computation of disjoint paths in routing and reliable communication, both in undirected and directed graphs [21, 24].

We assume that the reader is familiar with the standard graph terminology, as contained for instance in [7]. An *undirected path* (resp., *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 $i = 1, 2, \ldots, k-1$. An undirected graph G is *connected* if there is an undirected path from each vertex to every other vertex. The *connected components* of an undirected graph are its maximal connected subgraphs. A directed graph G is *strongly connected* if there is a directed path from each vertex to every other vertex. The *strongly connected components* of a directed graph are its maximal connected subgraphs.

* Partially supported by MIUR, the Italian Ministry of Education, University and Research, under Project AMANDA (Algorithmics for MAssive and Networked DAta).

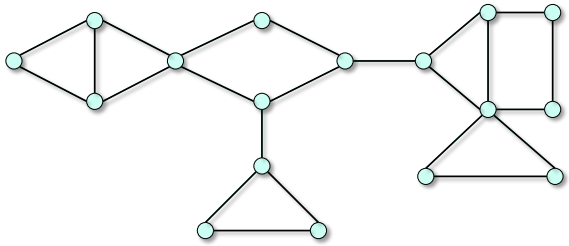


Figure 1 An undirected graph G.

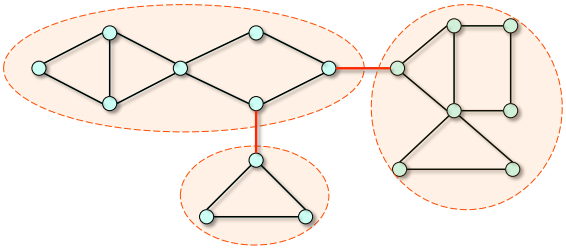


Figure 2 The bridges and 2-edge-connected components of the graph G in Figure 1. (Better viewed in color).

1.1 2-Connectivity in Undirected Graphs

Given an undirected graph $G = (V, E)$, an edge is a *bridge* if its removal increases the number of connected components of G. A graph G is 2-edge-connected if it has no bridges. The 2-*edge-connected components* of G are its maximal 2-edge-connected subgraphs. Figure 1 shows an undirected graph, and Figure 2 highlights its bridges and 2-edge-connected components.

Two vertices v and w are 2-edge-connected if there are two edge-disjoint paths between v and w: we denote this relation by $v \leftrightarrow_{2e} w$. Equivalently, by Menger's Theorem [31], v and w are 2-edge-connected if the removal of any edge leaves them in the same connected component.

Analogous definitions can be given for 2-vertex connectivity. In particular, a vertex is an *articulation point* if its removal increases the number of connected components of G. Figure 3 shows the articulation points of the graph in Figure 1. A graph G is 2-vertex-connected if it has at least three vertices and no articulation points. The 2-*vertex-connected components* of G are its maximal 2-vertex-connected subgraphs. Note that the condition on the minimum number of vertices in a 2-vertex-connected graph disallows degenerate 2-vertex-connected components consisting of one single edge. Figure 4 shows the 2-vertex-connected components of the graph in Figure 1.

Two vertices v and w are 2-vertex-connected if there are two internally vertex-disjoint paths between v and w: we denote this relation by $v \leftrightarrow_{2v} w$. If v and w are 2-vertex-connected then Menger's Theorem implies that the removal of any vertex different from v and w leaves them in the same connected component. The converse does not necessarily hold, since v and w may be adjacent but not 2-vertex-connected. It is easy to show that $v \leftrightarrow_{2e} w$ (resp., $v \leftrightarrow_{2v} w$) if and only if v and w are in a same 2-edge-connected (resp., 2-vertex-connected) component.

All bridges, articulation points, 2-edge- and 2-vertex-connected components of undirected graphs can be computed in linear time essentially by the same algorithm, which is simply based on depth-first search [34].

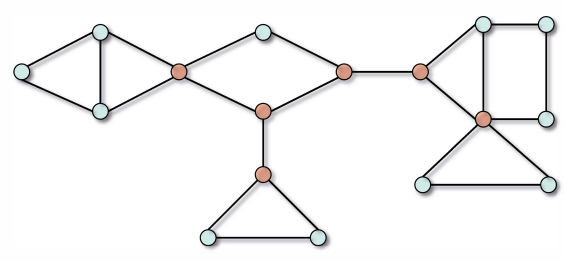


Figure 3 The articulation points of the graph G in Figure 1. (Better viewed in color).

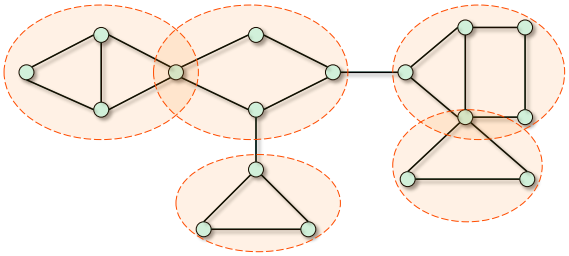
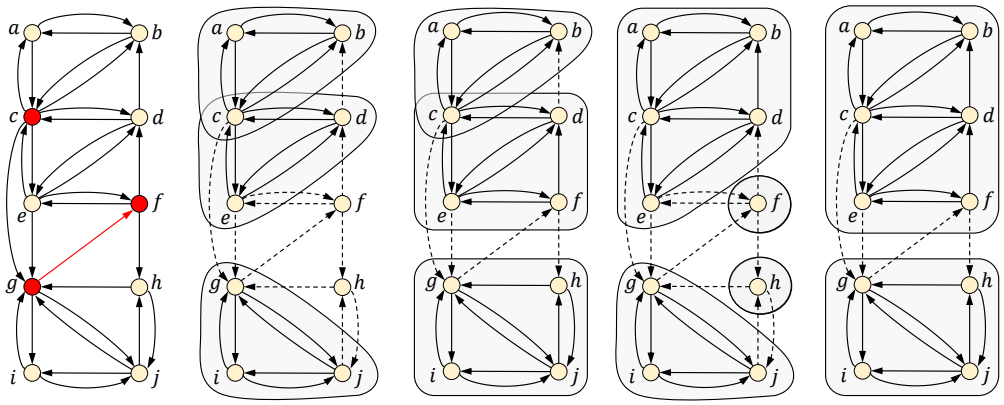


Figure 4 The 2-vertex-connected components of the graph G in Figure 1. (Better viewed in color).

1.2 2-Connectivity in Directed Graphs

The notions of 2-edge and 2-vertex connectivity can be naturally extended to directed graphs. The main idea is that now the role of connected components is played by strongly connected components. Given a digraph G, an edge (resp., a vertex) is a *strong bridge* (resp., a *strong articulation point*) if its removal increases the number of strongly connected components of G. A digraph G is 2-edge-connected if it has no strong bridges; G is 2-vertex-connected if it has at least three vertices and no strong articulation points. The 2-edge-connected (resp., 2-vertex-connected) components of G are its maximal 2-edge-connected (resp., 2-vertex-connected) subgraphs. Again, the condition on the minimum number of verti ces disallows for degenerate 2-vertex-connected components consisting of two mutually adjacent vertices (i.e., two vertices v and w and the two edges (v, w) and (w, v)).

Similarly to the undirected case, we say that two vertices v and w are 2-edge-connected (resp., 2-vertex-connected), and we denote again this relation by $v \leftrightarrow_{2e} w$ (resp., $v \leftrightarrow_{2v} w$), if there are two edge-disjoint (resp., internally vertex-disjoint) directed paths from v to w and two edge-disjoint (resp., internally vertex-disjoint) directed paths from w to v. (Note that a path from v to w and a path from w to v need not be edge-disjoint or vertex-disjoint). It is easy to see that $v \leftrightarrow_{2e} w$ if and only if the removal of any edge leaves v and w in the same strongly connected component. Similarly, $v \leftrightarrow_{2v} w$ implies that the removal of any vertex different from v and w leaves v and w in the same strongly connected component. We define a 2-*edge-connected block* (resp., 2-*vertex-connected block*) of a digraph $G = (V, E)$ as a maximal subset $B \subseteq V$ such that $u \leftrightarrow_{2e} v$ (resp., $u \leftrightarrow_{2v} v$) for all $u, v \in B$. Figure 5 illustrates (a) a strongly connected digraph G together with its strong articulation points and strong bridges, (b) the 2-vertex-connected components of G, (c) the 2-vertex-connected blocks of G, (d) the 2-edge-connected components of G, and (e) the 2-edge-connected blocks of G.



(a) G (b) $2VCC(G)$ (c) $2VCB(G)$ (d) $2ECC(G)$ (e) $2ECB(G)$

Figure 5 (a) A strongly connected digraph G, with strong articulation points and strong bridges shown in red (better viewed in color). (b) The 2-vertex-connected components of G. (c) The 2-vertex-connected blocks of G. (d) The 2-edge-connected components of G. (e) The 2-edge-connected blocks of G. Note that vertices e and f are in the same 2-vertex- (resp., 2-edge-) connected block of G since there are two internally vertex-disjoint (resp., edge-disjoint) paths from e to f and from f to e. However, e and f are not in the same 2-vertex (resp., 2-edge-) connected component of G. (Better viewed in color).

1.3 Differences between 2-Connectivity in Undirected and Directed Graphs

Connectivity-related problems for digraphs are notoriously harder than for undirected graphs, and indeed many notions for undirected connectivity do not translate to the directed case. Differently from undirected graphs, in digraphs 2-edge- and 2-vertex-connected blocks do not correspond to 2-edge- and 2-vertex-connected components, as it is clearly illustrated in Figure 5. Namely, two vertices may be 2-edge-connected (resp., 2-vertex-connected) but lie in different 2-edge-connected (resp., 2-vertex-connected) components. Furthermore, these notions seem to have a much richer and more complicated structure in digraphs, as depicted in Figure 6. Just to give an example, we observe that while in the case of undirected connected graphs the 2-edge-connected components (which correspond to the 2-edge-connected blocks) are exactly the connected components left after the removal of all bridges, for directed strongly connected graphs the 2-edge-connected components, the 2-edge-connected blocks, and the strongly connected components left after the removal of all strong bridges are not necessarily the same (see Figure 7).

Finally, we observe that an undirected graph is naturally decomposed by bridges (resp., articulation points) into a tree of 2-edge- (resp., 2-vertex-) connected components, known as the bridge-block (resp., block) tree (see, e.g., [36]). In digraphs, the decomposition induced by strong bridges and strong articulation points becomes much more complicated (see Figure 8): in general, it was shown by Benczúr that in digraphs there can be no "cut" tree for various connectivity concepts [3].

It is thus not surprising that, despite being complete analogs of the corresponding notions on undirected graphs, 2-edge and 2-vertex connectivity problems appear to be much more difficult on digraphs.

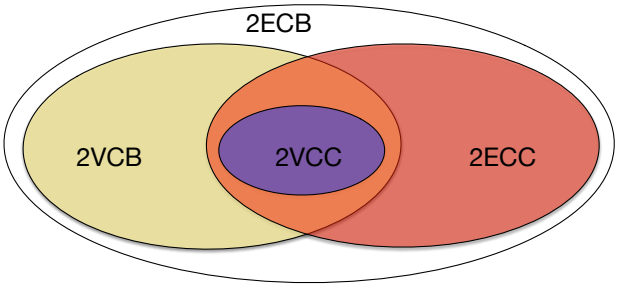
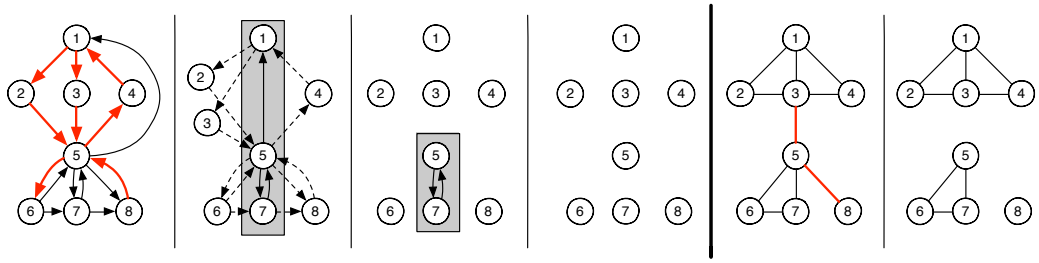


Figure 6 The relation among various notions of 2-connectivity in directed graphs. Two vertices that are 2-edge-connected (resp., 2-vertex-connected) are in the same 2-edge-connected (resp., 2-vertex-connected) block but not necessarily in the same 2-edge-connected (resp., 2-vertex-connected) component. Also, a 2-vertex-connected component is included in a 2-edge-connected component. (Better viewed in color).



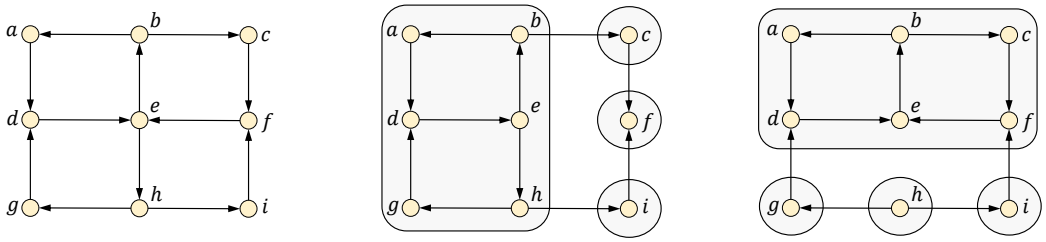
(a) G (b) $2ECB(G)$ (c) $G/SB(G)$ (d) $2ECC(G)$ (e) U (f) $2ECC(U)$

Figure 7 (a) A digraph G with strong bridges shown in red; (b) The 2-edge-connected blocks of G; (c) The strongly connected components left after removing all the strong bridges from G; (d) The 2-edge-connected components of G. (e) An undirected graph U with bridges shown in red; (f) The 2-edge-connected components of U, corresponding to the 2-edge-connected blocks and to the connected components left after the removal of all bridges of U. (Better viewed in color).

2 Simple-minded Algorithms for 2-edge and 2-vertex Connectivity in Directed Graphs

A simple algorithm for computing the 2-edge-connected components can be obtained by repeatedly removing all the strong bridges in the graph (and repeating this process until no strong bridges are left). At each round all the strong bridges can be computed in $O(m+n)$ time [26] and since there can be at most $O(n)$ rounds, the total time taken by this algorithm is $O(mn)$. The same bound was previously achieved by Nagamochi and Watanabe [33]. As for 2-vertex connectivity, Erusalimskii and Svetlov [9] proposed an algorithm that reduces the problem of computing the 2-vertex-connected components of a digraph to the computation of the 2-vertex-connected components in an undirected graph, but did not analyze the running time of their algorithm. Jaberi [28] showed that the algorithm of Erusalimskii and Svetlov has $O(nm^2)$ running time, and proposed two different algorithms with running time $O(mn)$. Both algorithms follow substantially the same high-level approach as the simple algorithm for computing the 2-edge-connected components of a digraph sketched before.

A simple algorithm for computing the 2-edge- or 2-vertex-connected blocks of a digraph G takes $O(mn)$ time: given a vertex v, one can find in linear time all the vertices that are



(a) G (b) $G \setminus (f, e)$ (c) $G \setminus (e, h)$

Figure 8 An example illustrating the complicated structure of 1-edge cuts in digraphs. (a) A strongly connected digraph G. (b) The strongly connected components in $G \setminus (f, e)$. (c) The strongly connected components in $G \setminus (e, h)$. Note that a strongly connected component in $G \setminus (f, e)$ and a strongly connected component in $G \setminus (e, h)$ are neither disjoint nor nested. In fact, all edges are strong bridges, and the deletion of each edge creates many non-disjoint and non-nested sets in the resulting partitions.

2-edge- or 2-vertex-connected with v with the help of dominator trees [11]. Since in the worst case this step must be repeated for all vertices v in G, the total time required by this simple algorithm is $O(mn)$. Very recently, Jaberi [27] presented algorithms for computing the 2-vertex-connected and 2-edge-connected blocks. His algorithms require $O(n \cdot \min\{m, b^*n\})$ time for computing the 2-edge-connected blocks and $O(n \cdot \min\{m, (a^* + b^*)n\})$ time for computing the 2-vertex-connected blocks, where a^* and b^* are respectively the number of strong articulation points and strong bridges in the digraph G. Since both a^* and b^* can be as large as $O(n)$, both bounds are $O(mn)$ in the worst case.

3 Flow Graphs and Dominators

In this section, we introduce some of the main tools that provided to be useful for solving 2-connectivity problems. Let $G = (V, E)$ be a strongly connected graph. Throughout, we denote by $G^R = (V, E^R)$ the *reverse digraph* of G, i.e., the digraph obtained by reversing the direction of all edges.

A *flow graph* is a digraph with a distinguished *start vertex* s such that every vertex is reachable from s. Let s be a fixed but arbitrary start vertex of a strongly connected digraph G. Since G is strongly connected, all vertices are reachable from s and reach s, so we can view both G and G^R as flow graphs with start vertex s. To avoid ambiguities, throughout the paper we will denote those flow graphs respectively by G_s and G_s^R. Vertex u is a *dominator* of vertex v (u *dominates* v) in G_s if every path from s to v in G_s contains u. Vertex u is a *proper dominator* of v if u dominates v and $u \neq v$. 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 flow graph G_s. The dominator relation is reflexive and transitive. Its transitive reduction is a rooted tree, the *dominator tree* D: u dominates v if and only if u is an ancestor of v in D. For any $v \neq s$, we denote by $d(v)$ the parent of v in D. Similarly, we can define the dominator relation in the flow graph G_s^R, and let D^R denote the dominator tree of G_s^R, and $d^R(v)$ the parent of v in D^R. Throughout the paper, we let N (resp., N^R) denote the set of nontrivial dominators of G_s (resp., G_s^R). Lengauer and Tarjan [30] presented an algorithm for computing dominators in $O(m\alpha(m, n))$ time for a flow graph with n vertices and m edges, where α is a functional inverse of Ackermann's function [35]. Subsequently, several linear-time algorithms were discovered [1, 4, 11, 12].

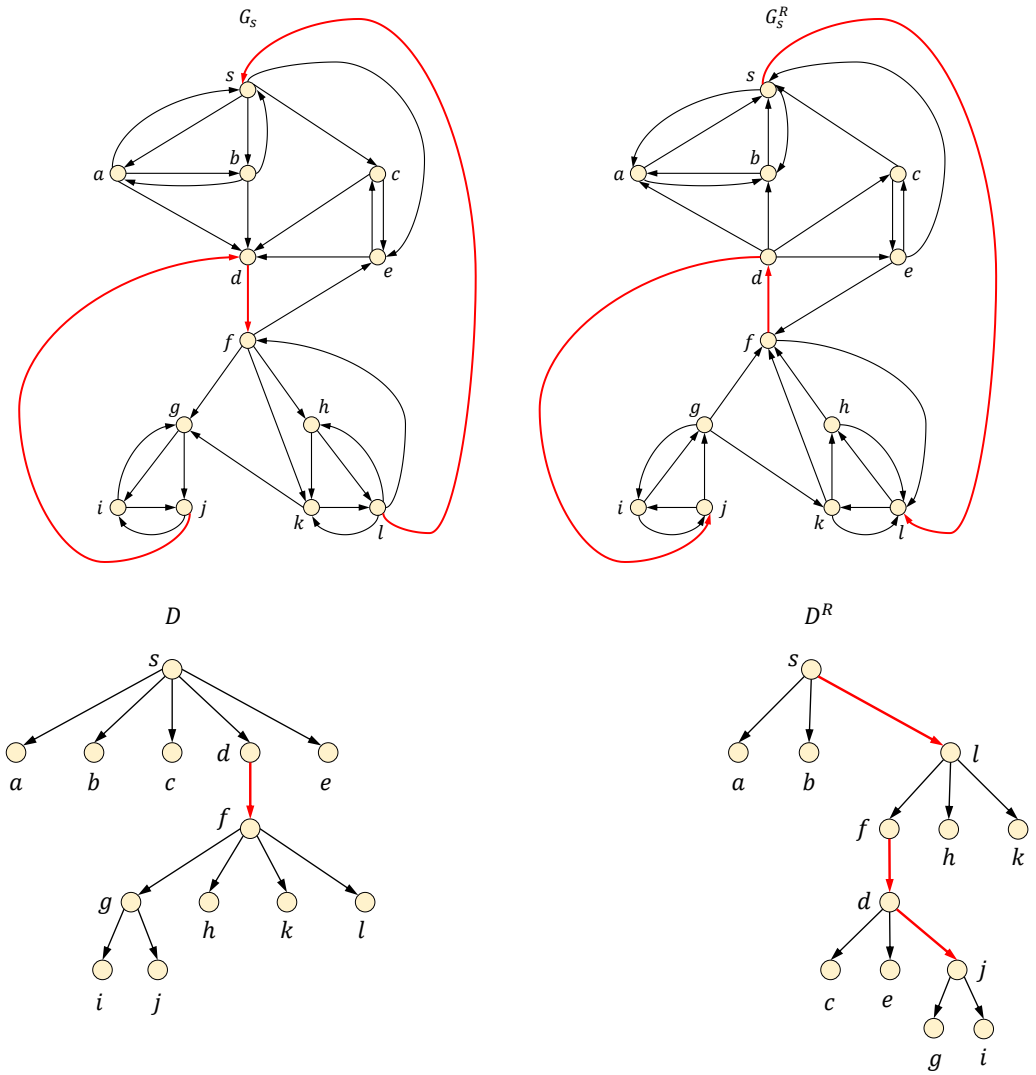


Figure 9 A flow graph G_s and its reverse G_s^R, and their dominator trees D and D^R. The corresponding digraph G is strongly connected. Strong bridges of G and G^R and bridges of G_s and G_s^R in D and D^R are shown red. (Better viewed in color.)

Figure 9 shows a flow graph G_s, its reverse G_s^R, and their dominator trees D and D^R.

An edge (u, v) is a *bridge* of a flow graph G_s if all paths from s to v include (u, v).[1] Let s be an arbitrary start vertex of G. As shown in [26], an edge $e = (u, v)$ is strong bridge of G if and only if it is either a bridge of G_s or a bridge of G_s^R. As a consequence, all the strong bridges of G can be obtained from the bridges of the flow graphs G_s and G_s^R, and thus there can be at most $2(n - 1)$ strong bridges overall.

[1] Throughout the paper, to avoid confusion we use consistently the term *bridge* to refer to a bridge of a flow graph and the term *strong bridge* to refer to a strong bridge in the original graph.

4 Efficient Algorithms for 2-Connectivity in Directed Graphs

In this section, we show how to exploit the basic tools described in Section 3 to obtain fast algorithms for 2-connectivity in digraphs. We start by showing a connection between strong articulation points and dominators in flowgraphs. Consider the problem of finding all strong articulation points of a strongly connected digraph $G = (V, E)$. Let s be any vertex in G. Since G is strongly connected, every vertex in G is reachable from s: thus for every vertex $s \in V$, G_s is a flowgraph. Note that there can be n flowgraphs for each strongly connected graph. The following lemmas show a close relationship between strong articulation points in strongly connected graphs and non-trivial dominators in flow graphs.

▶ **Lemma 1** ([26]). *Let $G = (V, E)$ be a strongly connected graph, and let s be any vertex in G. Let G_s be the flowgraph with start vertex s. If a vertex 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, 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 2** ([26]). *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.*

Proof. If u is a strong articulation point of G, then there must exist 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 note that Lemmas 1 and 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 [1, 4, 11, 12] and there are exactly n flowgraphs to be considered, the running time of this algorithm is $O(n(m + n))$. We show next 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.

▶ **Theorem 3** ([26]). *Let $G = (V, E)$ be a strongly connected graph, and let $s \in V$ be any vertex in G. Let G_s and G_s^R be respectively the flowgraphs with start vertex s, D and D^R their dominator trees, and N and N^R the non-trivial dominators in D and D^R. Then vertex $v \neq s$ is a strong articulation point in G if and only if $v \in N \cup N^R$.*

Proof. We first prove that if v is a strong articulation point in G, $v \neq s$, then v must be a non-trivial dominator either in D or in D^R. Assume not: namely, assume that v is a strong articulation point in G, $v \neq s$, but $v \notin N \cup N^R$. 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, $w \neq s$, $w \neq v$, such that the following is true: w is in the same strongly connected component as s in G, but w is not in the same strongly connected component as s in $G \setminus \{v\}$. Namely, $v \neq s$, $v \neq w$, and 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 non-trivial dominator of w in the flowgraph G_s, which clearly contradicts our assumption that $v \notin N$. 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 non-trivial dominator of w in the flowgraph G_s^R, which contradicts our assumption that $v \notin N^R$. This shows that if v is a strong articulation point in G, then v must be in N or in N^R.

To prove the converse, let v be any vertex such that $v \in N$ or $v \in N^R$. If $v \in N$, v is a non-trivial dominator in G_s, and thus v must be a strong articulation point in G by Lemma 1. Analogously, if $v \in N^R$, again by Lemma 1 v must be a strong articulation point in G^R, and thus in G. This completes the proof of the theorem. ◀

Note that Theorem 3 provides no information on whether vertex s is a strong articulation point. However, this can be easily checked in linear time, yielding the following theorem.

▶ **Theorem 4** ([26]). *All the strong articulation points of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

The strong bridges of a directed graph can be found in an analogous fashion, giving rise to the following theorem:

▶ **Theorem 5** ([26]). *All the strong bridges of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

A more sophisticated usage of dominator trees, combined with other properties, gives rise to efficient algorithms for computing the 2-edge-connected and 2-vertex-connected blocks and components of a directed graph, as stated in the remainder of this section.

In particular, Georgiadis et al. [16] gave a linear-time algorithms for computing the 2-edge-connected blocks of a digraph. Their approach hinges on two different algorithms. The first is a simple iterative algorithm that builds the 2-edge-connected blocks by removing one strong bridge at a time. The second algorithm is more involved and recursive: the main idea is to consider simultaneously how different strong bridges partition vertices with the help of dominator trees. Although both algorithms run in $O(mn)$ time in the worst case, Georgiadis et al. [16] showed that a sophisticated combination of the iterative and the recursive method is able to achieve a linear-time bound, as shown in the following theorem.

▶ **Theorem 6** ([16]). *The 2-edge-connected blocks of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

Using the linear-time algorithm for computing the 2-edge-connected blocks, one can preprocess a digraph in linear time, and then can answer in constant time queries on whether any two vertices are 2-edge-connected. Additionally, when two query vertices v and w are not 2-edge-connected, one can produce in constant time a "witness" of this property, by exhibiting an edge that is contained in all paths from v to w or in all paths from w to v. As a consequence of the linear-time algorithm of Theorem 6, one can also compute in linear time a sparse certificate for 2-edge-connected blocks, i.e., a subgraph of the input graph that has $O(n)$ edges and maintains the same 2-edge-connected blocks as the input graph. The interested reader is referred to [16] for all the details.

Following the high-level approach of [16] for finding the 2-edge-connected blocks, Georgiadis et al. [17] were able to prove that also the 2-vertex-connected blocks of a digraph can be computed in linear time. The algorithm for computing the 2-vertex-connected blocks is much more involved than the 2-edge connectivity algorithm required several novel ideas and more sophisticated techniques to achieve the claimed bounds. Moreover, differently from 2-edge connectivity, 2-vertex connectivity in digraphs is plagued with several degenerate special cases, which are not only more tedious but also more cumbersome to deal with. For

instance, the algorithm in [16] exploits implicitly the property that two vertices v and w are 2-edge-connected if and only if the removal of any edge leaves v and w in the same strongly connected component. Unfortunately, this property no longer holds for 2-vertex connectivity, as for instance two mutually adjacent vertices are always left in the same strongly connected component by the removal of any other vertex, but they are not necessarily 2-vertex-connected. This is summarized in the following theorem.

▶ **Theorem 7** ([17]). *The 2-vertex-connected blocks of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

Similary to the case of 2-edge connectivity, other side results can be obtained as an application of this algorithm. In particular, one can construct an $O(n)$-space data structure that reports in constant time if two vertices are 2-vertex-connected. by exhibiting a vertex (i.e., a strong articulation point) or an edge (i.e., a strong bridge) that separates them. Once again, one can also compute in linear time a sparse certificate for 2-vertex connectivity, i.e., a subgraph of the input graph that has $O(n)$ edges and maintains the same 2-vertex connectivity properties.

We now turn to the problem of computing the 2-edge- and 2-vertex-connected components of a digraph. In this case, Henzinger et al. [23], presented fast algorithms for computing the 2-edge- and 2-vertex-connected components of a directed graph. The main idea behind their algorithm is a hierarchical graph sparsification that was introduced by Henzinger et al. [22] for undirected graphs and extended to directed graphs in [5]. Roughly speaking, this sparsification technique allows one to replace the m in the $O(mn)$ running times by an n, yielding $O(n^2)$ running times in place of $O(mn)$. Henzinger et al. [23] were able to find structural properties of 2-edge and 2-vertex connectivity in directed graphs that allow one to apply this technique starting from the simple-minded $O(mn)$ algorithms and a clever use of dominators. Those bounds are summarized in the following theorem.

▶ **Theorem 8** ([23]). *The 2-edge- and 2-vertex-connected components of a directed graph G can be computed in $O(n^2)$ time in the worst case.*

Additionally, Henzinger et al. [23] presented an $O(m^2/\log n)$ time algorithm for computing the 2-edge-connected components, which provides a small improvement for sparse graphs, i.e., $m = O(n)$. The same approach can be extended to k-edge- and k-vertex-connected components, for any constant k, with a running time of $O(n^2 \log n)$ for k-edge connectivity and $O(n^3)$ for k-vertex connectivity.

Finally, we mention that Georgiadis et al. [19] initiated the study of the dynamic maintenance of 2-edge-connectivity relationships in directed graphs. In particular, they presented an algorithm that can update the 2-edge-connected blocks of a digraph G with n vertices through a sequence of m edge insertions in a total of $O(mn)$ time. After each insertion, one can answer the following queries in asymptotically optimal time:

- Test in constant time if two query vertices v and w are 2-edge-connected. Moreover, if v and w are not 2-edge-connected, one can produce in constant time a "witness" of this property, by exhibiting an edge that is contained in all paths from v to w or in all paths from w to v.
- Report in $O(n)$ time all the 2-edge-connected blocks of G.

▶ **Theorem 9** ([19]). *The 2-edge-connected blocks of a digraph with n vertices can be maintained through a sequence of edge insertions in $O(mn)$ time, where m is the total number of edges in G after all insertions.*

We remark that this is the first known dynamic algorithm for 2-connectivity problems on digraphs, and it matches the best known bounds for simpler problems, such as incremental transitive closure [25].

5 Sparse Subgraphs Preserving 2-Connectivity in Directed Graphs

Other problems that were considered in the area of 2-connectivity for directed graphs are related to the computation of a minimum spanning subgraph (i.e., a subgraph with minimum number of edges) that maintains certain 2-connectivity requirements in addition to strong connectivity. More specifically, one problem that has been investigated is finding a smallest strongly connected spanning subgraph of a digraph G that has the same 2-edge- (respectively, 2-vertex-) connectivity properties as G. Both for 2-edge- and for 2-vertex connectivity, this problem is known to be NP-hard [13, 18]. We next review some of the algorithms proposed in the literature respectively for edge and for vertex connectivity.

Laekhanukit et al. [29] gave a randomized $(1+1/k)$-approximation algorithm for computing the smallest k-edge-connected spanning subgraph of a k-edge-connected graph. Georgiadis et al. [18] used the algorithm in [29] to compute a 3/2-approximate minimum spanning subgraph that has the same 2-edge-connected components and additionally presented a faster 2-approximation algorithm that runs in linear time. Let G be a strongly connected graph. Jaberi [27] considered the problem of computing a smallest subgraph that has the same 2-edge-connected blocks (or the same 2-vertex-connected blocks) as G. Unfortunately, the approximation ratio in Jaberi's algorithms is $O(n)$ in the worst case. Georgiadis et al. [18] improved this result by presenting a linear-time 4-approximation algorithm for computing the smallest strongly connected spanning subgraph that has the same 2-edge-connected blocks as G. Additionally, they presented a linear-time algorithm for the problem of computing the smallest subgraph that has both the same 2-edge-connected components and the same 2-edge-connected blocks as G. The algorithms in [18] that compute spanning subgraphs with the same 2-edge-connected components as G run in linear time once the 2-edge-connected components of G are available (we remark that the currently best known bound for computing the 2-edge-connected components is $O(n^2)$ [23]).

For the smallest k-vertex-connected spanning subgraph, Cheriyan and Thurimella [6], gave a $(1 + 1/k)$-approximation algorithm that runs in $O(km^2)$ time. For $k = 2$, Georgiadis [14] presented a linear time algorithm with approximation ratio 3. Based on the algorithm from [14], the running time of Cheriyan and Thurimella's algorithm was improved to $O(m\sqrt{n}+n^2)$ for $k = 2$. Let G be a strongly connected graph. Georgiadis et al. [15] presented a constant-factor approximation algorithm for the problem of computing the smallest subgraph that preserves the 2-vertex-connected blocks of G. More specifically, they gave a linear-time 6-approximation algorithm for this problems, and further extended this algorithm to compute a sparse subgraph with the same approximation gurantee that has both the same 2-vertex-connected components and the same 2-vertex-connected blocks as G. The algorithm that computes a sparse subgraph that preserves both the 2-vertex-connected blocks and the 2-vertex-connected components of the input graph G runs in linear time, once the 2-vertex-connected components of G are available (we remark that the currently best known bound for computing the 2-vertex-connected components is $O(n^2)$ [23]). Finally, in [15] Georgiadis et al. presented a 6-approximation algorithm for computing a strongly connected spanning subgraph of G that preserves all the 2-connectivity relations, i.e., both the 2-edge- and the 2-vertex-connected components and the 2-edge- and the 2-vertex-connected blocks. Once again, this algorithm runs in linear time, provided that the 2-edge- and the 2-vertex-connected components of G are available.

We remark that references [15, 18] provide efficient implementations of all those approximation algorithms that run very fast in practice. Additionally, they also present several heuristics that improve the quality (i.e., the number of edges) of the computed spanning subgraphs, and assess how all these algorithms perform in practical scenarios by conducting a thorough experimental study.

6 Conclusions and Open Problems

We have surveyed some very recent results on 2-edge and 2-vertex connectivity problems in directed graphs, which revealed to be harder than their counterparts on undirected graphs. Experimental studies for algorithms that compute dominators, strong bridges, strong articulation points, 2-edge- and 2-vertex-connected blocks are presented in [8, 10, 20]. Those experimental results are very promising, as they show that the corresponding fast algorithms given in [11, 16, 17, 26] perform very well in practice even on very large graphs.

This recent bulk of work has raised some interesting and perhaps intriguing questions. In particular, the main open problem is whether the 2-edge-connected or the 2-vertex-connected components of a digraph can be computed in linear time. Moreover, the dynamic maintenance of 2-edge and 2-vertex connectivity in directed graphs deserves further investigation. Finally, we have described in Section 5 linear-time constant-factor approximation algorithms for computing minimum spanning subgraphs that preserve the 2-edge- and 2-vertex-connected blocks of a graph [15, 18]. The trade-offs between running times and approximation guarantees need further study. In particular, can the approximation guarantees in [15, 18] be improved while still maintaining linear running times? Can they match the corresponding approximation ratios for computing the 2-edge- and 2-vertex-connected spanning subgraphs of 2-edge- and 2-vertex-connected graphs [6, 29], respectively?

References

1 S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–32, 1999.

2 J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications (Springer Monographs in Mathematics)*. Springer, 1st ed. 2001. 3rd printing edition, 2002.

3 A. A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.*, 24:505–510, 1995.

4 A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.

5 K. Chatterjee and M. Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014. doi:10.1145/2597631.

6 J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. Comput.*, 30(2):528–560, 2000.

7 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

8 W. Di Luigi, L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-connectivity in directed graphs: An experimental study. In *Proc. 17th Workshop on Algorithm Engineering and Experiments*, pages 173–187, 2015. doi:10.1137/1.9781611973754.15.

9 Ya. M. Erusalimskii and G. G. Svetlov. Bijoin points, bibridges, and biblocks of directed graphs. *Cybernetics*, 16(1):41–44, 1980. doi:10.1007/BF01099359.

10 D. Firmani, G. F. Italiano, L. Laura, A. Orlandi, and F. Santaroni. Computing strong articulation points and strong bridges in large scale graphs. In *Proc. 10th Int'l. Symp. on Experimental Algorithms*, pages 195–207, 2012.

11 W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013. doi:10.1016/j.jda.2013.10.003.

12 H. N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Transactions on Algorithms*, 12(2):24:1–24:73, February 2016. doi:10.1145/2764909.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

14 L. Georgiadis. Approximating the smallest 2-vertex connected spanning subgraph of a directed graph. In *Proc. 19th European Symposium on Algorithms*, pages 13–24, 2011.

15 L. Georgiadis, G. F. Italiano, A. Karanasiou, C. Papadopoulos, and N. Parotsidis. Sparse subgraphs for 2-connectivity in directed graphs. In *Proc. 15th Int'l. Symp. on Experimental Algorithms*, pages 150–166, 2016. doi:10.1007/978-3-319-38851-9_11.

16 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-edge connectivity in directed graphs. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms*, pages 1988–2005, 2015.

17 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-vertex connectivity in directed graphs. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 605–616, 2015.

18 L. Georgiadis, G. F. Italiano, C. Papadopoulos, and N. Parotsidis. Approximating the smallest spanning subgraph for 2-edge-connectivity in directed graphs. In *ESA 2015*, pages 582–594, 2015.

19 L. Georgiadis, G. F. Italiano, and N. Parotsidis. Incremental 2-edge connectivity in directed graphs. In *Proc. 43rd Int'l. Coll. on Automata, Languages, and Programming*, 2016. To appear.

20 L. Georgiadis, L. Laura, N. Parotsidis, and R. E. Tarjan. Loop nesting forests, dominators, and applications. In *Proc. 13th Int'l. Symp. on Experimental Algorithms*, pages 174–186, 2014.

21 Y. Guo, F. Kuipers, and P. Van Mieghem. Link-disjoint paths for reliable QoS routing. *International Journal of Communication Systems*, 16(9):779–798, 2003. doi:10.1002/dac.612.

22 M. Henzinger, V. King, and T. J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999. doi:10.1007/PL00009268.

23 M. Henzinger, S. Krinninger, and V. Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 713–724, 2015.

24 A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79(1):43–59, 1988.

25 G. F. Italiano. Amortized efficiency of a path retrieval data structure. *Theor. Comput. Sci.*, 48(3):273–281, 1986. doi:10.1016/0304-3975(86)90098-8.

26 G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012. doi:10.1016/j.tcs.2011.11.011.

27 R. Jaberi. Computing the 2-blocks of directed graphs. *RAIRO-Theor. Inf. Appl.*, 49(2):93–119, 2015. doi:10.1051/ita/2015001.

28 R. Jaberi. On computing the 2-vertex-connected components of directed graphs. *Discrete Applied Mathematics*, 204:164–172, 2016. doi:10.1016/j.dam.2015.10.001.

29 B. Laekhanukit, S. O. Gharan, and M. Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *ICALP 2012*, pages 606–616, 2012.

30 T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.

31 K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

32 H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition.

33 H. Nagamochi and T. Watanabe. Computing k-edge-connected components of a multigraph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76–A(4):513–517, 1993.

34 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

35 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

36 J. Westbrook and R. E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(5&6):433–464, 1992. doi:10.1007/BF01758773.