Survey on 2-connectivity in Directed Graphs

T.Manohar Reddy*1, Dr.P.Chandra Sekhar
PG Student, Department of CSE, JNT University, Ananthapur, AP
Department of CSE, SK University, Ananthapur, AP

ABSTRACT

In this paper**,** the Survey of some recent results on graph connectivity problems like 2-vertex connectivity and 2-edge connectivity problems in directed graphs have been discussed. 2-vertex and 2-edge connectivity problems in directed graphs are more difficult than on undirected graphs. By using depth first search 2-edge and 2-vertex connectivity, bridges, articulation points can be computed in linear time for undirected graphs. In the case of a directed graph, the same problems have been much more challenging and required the development of new ideas and techniques.

I.INTRODUCTION

Connectivity is a fundamental concept in graph theory. A lot of work done in the case of a undirected graph, but not much has been studied for directed graphs. In this survey, we are summarizing the 2-vertex and 2-edge connectivity for directed graphs, the concepts of graph theory. The definition of connectivity given by Douglas West [1] as a **separating set** or **vertex cut** of a graph is a set S is a subset of *V(G)*such that G – S has more than one component. The **Connectivity** of G, written k(G), is minimum size set S such that G – S is disconnected or has only one vertex. A graph is k-**connected** if it has connectivity is at least *k*. A graph with more than two vertices has connectivity 1 if and only if it is connected and has a cutvertex. A graph with more than one vertex has connectivity 0 if and only if it is disconnected.

Douglas West [1,4.1.7] defined, A **disconnecting set**of edges is a set F ⊆ E(G) such that G – F has more than one component. A graph is *k*-**edge connected** if every disconnecting set has at least *k* edges, The **edge-connectivity** of G is a minimum size of a disconnecting set. A maximal connected subgraph that has no cut-vertex is a **block.**

II.FUNDAMENTAL CONCEPTS

2.1 Connectivity in Undirected Graphs

In an undirected graph G, if we remove any edge, then the number of connected components will increase, such that the removal edge known as a *bridge.*In a graph, if no bridges exist then the graph is 2-edge connected.

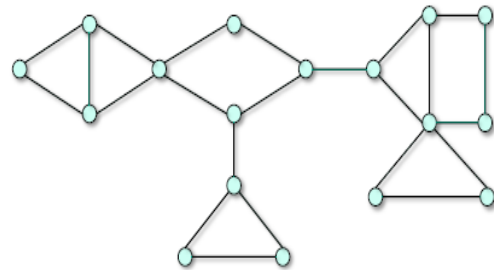

Figure 1 undirected graph

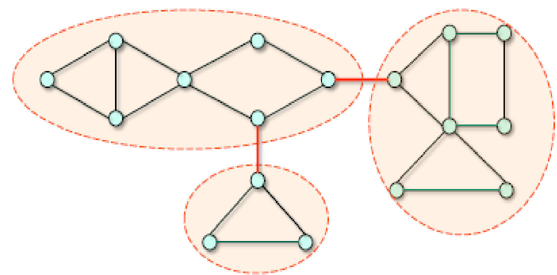

Figure 2 2-edge connected components

In figure1and Figure 2, we have shown bridges and 2-edge connected components for an undirected graph. By Menger's Theorem[2], any two vertices are 2-edge connected if and only if a number of connected components will not increase if we remove an edge from the graph G.

In an undirected graph G, if we remove any vertex, then the number of connected components will increase, then removal vertex is known as

an*articulation point*.In a graph no articulation point present then the graph is 2-vertex connected.

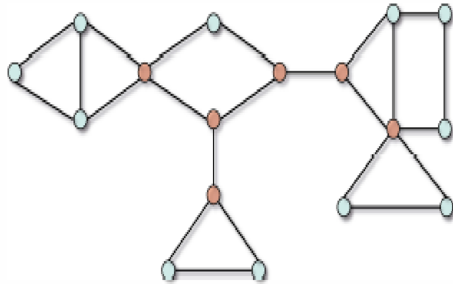

Figure 3 The articulation points of Graph

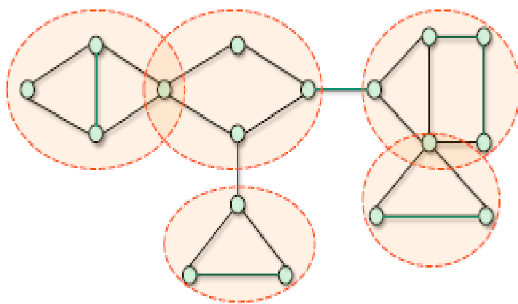

Figure 4 2-vertex connected components

By using depth-first search [4] we can compute bridges, articulation points, 2-edge connected components and 2-vertex connected components in linear time.

2.2 Connectivity in Directed Graphs

In a directed graph G, if we remove an edge(resp.. a vertex), thena number of *strongly connected components,* the removal edge known as a*strong bridge(a strong articulation point)*[10].In a directed graph if no strong bridge(strong articulation point) exists, then graph the graph is 2-edgeconnected(2-vertex connected). Two vertices, let us say v and u, are 2-edge connected(resp..2-vertex connected) if it has two edge-disjoint (two vertex-disjoint) paths from v to u as well as u to v should be present.

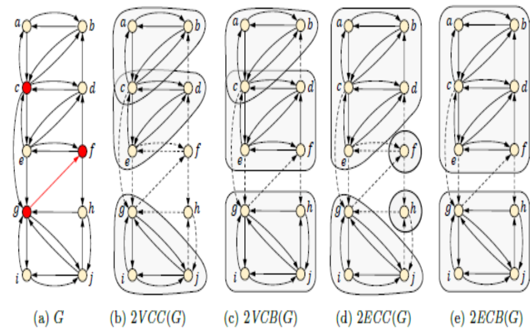

Figure 5 (a) strongly connected directed graph.(b) 2-vertex connected components. (c) 2-vertex connected blocks. (d) 2-edge connected components. (e) 2-edge connected blocks.

III.LITERTURE SURVEY

ALGORITHMS FOR CONNECTIVITY IN DIRECTED GRAPHS

3.1 Authors: Erusalimskii, Svetlov [6]

In this algorithm, authors proposed that it reduces 2-vertex connected components problem of thedirected graph to the 2-vertex connected components of the undirected graph. But they did not analyze the time complexity of theproblem.

METHOD:

1. Repeat
2. Compute the connected components
3. Remove the strong bridges
4. For every vertex do
5. Compute the connected components
6. Remove the strong bridges
7. Until no edge to remove on step 6
8. Compute 2-connected components

The authors did not analyze the time complexity of above algorithm, but Jaberi[5]showed that the algorithm will take $O(nm^2)$ time which has a huge gap between directed and undirected graph.the authors provided an algorithm for computing all *biblocks* of a directed graph, where the biblocks of a directed graph are its maximal strongly connectedsubgraphs that do not contain any strong articulation point. A biblock is either a 2-vcc, a single vertex or two vertices which are connected by two

antiparallel edges.In this paper, we are only interested in computing the 2-vccs of a directed graph. LetG = (V,E) be a strongly connected graph, and let v be a strong articulation pointin G. Then the vertex v does not necessarily occur in two or more 2-vccs of G.

Theorem:*The running time of thealgorithm is* $O(nm^2)$.

Proof. The number of iterations of the repeat-loop is at most m since at least oneedge is removed in each iteration. The strongly connected components of a directedgraph can be found in linear time using Tarjan's algorithm. In each iteration ofthe repeat-loop, steps 4–7 require O(n(n + m)) time. The 2-connected componentsof an undirected graph can be computed in linear time using Tarjan's algorithm.Thus, the total running time of Algorithm is $O(m(n(m + n))) = O(nm^2)$.

3.2 Nagamochi, Watanabe[9]

Nagamochi and Watanabe used max-flow algorithm to determine the edge weights in the auxiliary graph.

METHOD:

In theprocedure, the algorithm picks the random vertex from the input graph G, then runs the max-flow algorithm to find out the max-flow from the sourcevertex to randomly picked vertex in the first step. Then determine the min-cut from all max-flow, which is nothing but minimum max-flow among all max-flows. The min –cut edge added to the auxiliary graph. This procedure calls recursively.After the auxiliary graph constructed k-edge connected components can be constructed by removing all edges whose max-flow value less than k.The algorithm time complexity is O(Fmn), where F is the time taken for computing max-flow and m,n arenumber of edges and vertices in the input graph G. This algorithm not only for 2-connectivity and also computes the k-connected components, where k might be any value greater than or equal to one.

3.3 Authors: Jaberi [6]

In this method, Author investigates the relationship between 2-vertex connected components and dominator trees and also proposed an algorithm for 3-vertex connected components. And it takes $O(nm^2)$in theworst case, of graph G which contain m edges and n vertices.

Method:

1. If G is 2-vertex connected then output v
2. else
3. Compute the strong articulation points
4. Chose a vertex such that is a strong articulation point
5. Compute dominator trees
6. Chose a dominator tree of that contains more non-trivial dominators
7. for each vertex V do
8. if $|M(v)| \geq 2$ then
9. compute the strongly connected components
10. for each strongly connected component do
11. if $|C| \geq 3$ then
12. Recursively compute the 2-vertex connected components of G[C]
13. Else
14. Recursively compute 2-VCC of G[w]U[w] and output them.

Theorem 3,1. The algorithm runs in O(nm) time.

Proof. The dominators of a flowgraph can be found in linear time [15,14]. The strongarticulation points of a directed graph can also be computed in linear time using thealgorithm of Italiano et al. [12]. Furthermore, the strongly connected components of a directed graph can be computed in linear time using Tarjan's algorithm [13]. Ateach level of the recursion, atleast one vertex must be removed in lines 4–5 or the setof vertices must be split in lines 6–10. Hence, the recursion depth is at most n. Fixsome recursion level. We consider the cost of the calls of the procedure exceptingthe recursion. For one call, the cost is linear in the size of the current subgraph. LetG′[C1 ∪ {v}] = (V1,E1),G′[C2 ∪ {v}] = (V2,E2), . . . ,G′[Ct ∪ {v}] = (Vt,Et) be thesubgraphs of the directed graph G′ = (V ′,E′) considered on this level in all calls.Then P1≤i≤t |Ei| ≤ |E′| since the strongly connected components of G′ are disjoint.The total cost at each level of the recursion is, therefore, O(nm).we have abase condition such that a graph 2-vertex connected, it should have atleast 3 vertices. Finding strong articulation points we use simple breadth first search algorithm which takes only *O(m+n)* linear time. To find dominator tree we

should find all dominator of the subgraph. Vertex *u* is said to be dominator of vertex *v* in graph G if every path from the source vertex to vertex v in graph G contains vertex*u*. Lengauer and Tarjan [7] presented an algorithm for computing dominators in$O(m\alpha(m,n))$ time for graph G which has m edges and n vertices, α is an Ackermann's[8] inverse function. Overall time for above method is $O(nm)$ in the worst case. If a strong bridge separates pairs of vertices in successive recursive calls which cause the worst behavior of this method, then it will appear as the strong bridge entering the root of a subtree of a dominator tree

3.4 Authors:M.Henzinger, S.Krinninge,and V.Loitzenbauer

The authors present algorithms that compute the 2eSCCs and the 2vSCCs of a directed graph in $O(n^2)$ time. For 2eSCCs we additionally provide an algorithm that runs in O(m2/ log n) time, which is faster than $O(n^2)$ if m = O(n).Thus they significantly improve upon the previous O(mn)-time algorithms for both 2eSCCs and 2vSCCs. For 2eSCCs the previous upper bound stood for 20 years. This approach immediately generalizes to computing the k-edge strongly connected components (keSCCs) and the k-vertex strongly connected components (kvSCCs). We give algorithms that, for any integral constant k > 2, compute (1) the keSCCs in time O(n2log n) improving upon the previous upper bound of O(mn) and (2) the kvSCCs in time O(n3) improving upon the previous upper bound of $O(mn^2)$.

1. for i ← 1 to [log γ] − 1 do
2. 2 (S, Z) ← 2IsolatedSetLevel(G, i) Z contains v if G[S] is almost top or bottom SCC w.r.t. v
3. if S ≠∅ then
4. return 2vSCC(G[S ∪ Z]) ∪ 2vSCC(G[V \ S]).
5. (S, Z) ← 2IsolatedSet(G) /* Z contains v if G[S] is almost top SCC w.r.t. v */
6. if S ≠∅ then
7. return 2vSCC(G[S ∪ Z]) ∪ 2vSCC(G[V \ S])
8. else
9. return {G}

Let S be a set of at most 2 i vertices that induces a strongly connected subgraph G[S] of G such that G[S] is a top SCC or an almost top SCC with respect to some vertex v. Since the only edges from vertices of V \ S to S are from v, the in-degree of each vertex in S can be at most 2^i . By applying the results from the previous section, we show that we can detect such a set S by searching for SCCs and vertex-dominators in the graphs $F_{G,i}$ constructed from Gi with the artificial root G.

Lemma 3.4.1. If a set of vertices S with $|S| \leq 2^i$ induces a tSCC or an almost tSCC in G with respect to some vertex v, then $S \subseteq V \setminus B_{G,i}$.

To find bSCCs and almost bSCCs we also search for top SCCs in Rev(G). The search for both top and bottom SCCs ensures that whenever an (almost) tSCC and a disjoint (almost) bSCC exist in G, we only spend time proportional to the smaller one. This search is performed in Procedure 2Isolated Set Level, which fulfills the following guarantee.

Lemma3.4.2. If for some integer 1 ≤ i < log γ and G ∈ {G, Rev(G)} there exists a set of vertices T ⊆ V \ BG,i that induces in G a tSCC or an almost tSCC with respect to some vertex v with T (V \ {v}, then 2IsolatedSetLevel(G, i) returns a non-empty set S.

In Procedure 2vSCC we start the search for (almost) top SCCs at i = 1. Whenever the search is not successful, we increase i by one, until we have Gi = G or Rev(G) i = Rev(G). For the search the Procedure 2IsolatedSetLevel is used as long as 2 i< γ, i.e., both $B_{G,i}$and Brev(G),i are non-empty, and the Procedure 2IsolatedSetafterwards. Procedure 2IsolatedSet identifies an (almost) top SCC in G if one exists by using the known procedures for finding SCCs and articulation points. In this way, we can show that whenever we had to go up to i or had to use Procedure 2IsolatedSet to identify an (almost) top or bottom SCC in G, the identified subgraph contains $\Omega(2^i)$ vertices, where i = [log γ] for Procedure 2Isolated Set This will imply that the search in Gi and Rev(G) i for i up to i* takes time $O(n^{2i_*})$ which is O(n · min{|S|, |V \ S|}). This will allow us to bound the total running time by $O(n^2)$.Whenever the algorithm identifies an (almost) top or bottom SCC induced by a set of vertices S, it recursively calls itself on G[S ∪ Z] and G[V \ S] for Z = ∅ or Z = {v}, respectively. Every 2vSCC of G is completely contained in either G[S ∪ Z] or G[V \ S], which will imply the correctness of the algorithm.

Procedure:

1. foreach G ∈ {G, Rev(G)} do
2. construct G_i = (V, E_i) with $E_i = \cup_{v \in V}$ {first 2^i edges in $In_G(v)$}
3. $B_{G,i}$ = {v | $Indeg_G$ (v) > 2^i}
4. S ← TopSCCWithout(G_i ,$B_{G,i}$)
5. if S ≠∅ then
6. return (S, ∅)
7. construct flow graph $F_{G,i}(r_{G,i})$
8. if exists vertex-dominator v in $F_{G,i}(r_{G,i})$ then
9. S ← TopSCC Without(G_i \ {v}, $B_{G,i}$)
10. return (S, {v})
11. else if |$B_{G,i}$| = 1 and ∃ tSCC (V \ {$r_{G,i}$} in Gi \ {$r_{G,i}$} then
12. S ← TopSCC(Gi \ {$r_{G,i}$})
13. return (S, {$r_{G,i}$})
14. return (∅, ∅).

Theorem 3.4.3.1 Let G be a simple directed graph. The algorithm computes the 2vSCCs of G in O(n^2).

By stopping the recursion when the number of vertices is a small constant and distinguishingbetween the number of vertices n, at the current level of the recursion and the total number of vertices n, we can show that the runtime of O(n, min{|S|, |V \ S|}) without recursion leads to a total runtime of O(n^2).

3.5. Authors: T Wang, Y Zong[16]

Previous works focused on finding the *k*-edge-connected components when *k* is given in advance, especially for some small *k*, e.g., $k = 2$, $k = 3$, or the input graph is an undirected graph. Our algorithm can give answers for all possible values of *k*, and for both directed and undirected, simple graph or multiple graphs.If the capacity of each edge is regarded as one, computing *k*-edge-connected component can be solved by executing an algorithm for max-flow (or min-cut, by the max-flow min-cut). Since the cardinality of the minimum edge-cut separating vertices *a* and *b* is the number of edge-disjoint paths between *a* and *b*, a naive idea is to run an *s*-*t* min-cut algorithm for each pair of vertices on the graph. If we use an $O(n^3)$ time algorithm proposed by Goldberg and Tarjan [19], we can achieve an $O(n^5)$ time algorithm to get the min-cut of any two vertices. The other method is to use the global min-cut algorithm proposed by Stoer and Wagner [17], which finds the global min-cut in $O(mn + n^2 \log n)$ time. If the min-cut capacity is more than *k*, the graph is a *k*-edge-connected component; otherwise, any two vertices separated by the cut cannot be in the same *k*-edge-connected component. In the worst case, the global min-cut algorithm can be executed $n - 1$ rounds, leading to an $O(n^2m + n^3 \log n)$ time algorithm.In this paper, we give a simple algorithm to find the *k*-edge-connected components for all *k* in a directed or undirected, simple or multiple graph. We use an *s*-*t* max-flow algorithm as the basic procedure which is executed $2n - 2$ rounds to construct an auxiliary graph to store information concerning the edge-connectivity between all vertex pairs of the input graph. The time complexity to construct the auxiliary graph is $O(Fn)$, where *F* is the time required to compute the maximal flow between two vertices in graph *G*, e.g., for the maximal flow algorithm of Ford and Fulkerson [18], $F = O(fm)$, where *f* is the maximal value of all pairs of maximal flows, and for the algorithm by Goldberg and Tarjan [19], $F = O(n^3)$.Furthermore, any improvement made on *F* automatically implies improvement on the time complexity of our algorithm.After the auxiliary graph is constructed, for any value of *k*, the *k*-edge-connected components can then be determined by traversing the auxiliary graph in $O(n)$ time by a simple scan over the auxiliary graph.

Algorithm : Construction(G(V, E), s, N)

If $N = \{s\}$

Return.

Randomly pick a vertex *t* from $N - \{s\}$.

(x, S, T) ≔ s-t max-flow(G, s, t).

(x', T', S') ≔ s-t max-flow(G, t, s).

If $x' < x$

$x := x', S := S', T := T'$

Add edge (s, t) with weight x to A

Construction(G, s, $N \cap S$)

Construction(G, t, $N \cap T$)

Lemma 3.5.1. *There are n − 1 calls of procedure* Construction.

Proof. Since each call of procedure Construction adds an edge to A, and there are $n - 1$ edges in the finished A, there are $n - 1$ calls of procedure Construction.

Theorem 3. 5.1. *The preprocessing phase takes O(Fn) time and the query phase takes O(n) time per query, where F is the time to compute the maximal flow for two vertices in graph G.*

Proof. Procedure Construction is called $n - 1$ times. In procedure Construction, the basic algorithm for finding the maximal flow and runs in $O(F)$ time is executed for $n - 1$ times. Therefore, the preprocessing phase takes $O(Fn)$ time. (Note: if we use the Ford-Fulkerson algorithm [18] to compute the max-flow, the total time complexity is $O(fmn)$, where f is the maximal value of all pair of maximal flows.Since each query initiates a DFS traversal over A, the query time is thus $O(m + n)$. Since the vertex set of $A$is V and $|V| = n$, and $m = |E_A| = n - 1$ the query time is $O(n)$.

3.6.Authors:L.GEORGIADIS,G. ITALIANO, L. LAURA, N. PAROSTSIDIS

Although Algorithmsrun in $O(mn)$ time, they show that a careful combination of them gives linear-time algorithm.The critical observationis that if a strong bridge separates different pairs of verticesin successive recursive calls which cause the worst-case behavior of Algorithm

Algorithm Fast2ECB

1. Chose an arbitrary vertex s € V as a start vertex. Compute the dominator tree D(s) and the bridges of the flow graph G(s).
2. Partition D(s) into subtrees T(r) and compute the corresponding auxiliary graphs G_r.
3. For each auxiliary graph H = $G_r$do:
4. Compute the dominator tree $D^R_H(r)$ and bridges of $H^R(r)$. Let $d_H{}^R(q)$ be the parent of $q \neq r$ in D^R_H (r).
5. Partition D^R_H (r) into the subtrees $T^H(q)$. Compute the corresponding auxiliary graphs H_q with $q \neq r$.
6. Set [r] to consist of the ordinary vertices $T^H(r)$.
7. For each auxiliary graph Hq with $q \neq r$ do
8. Compute the strongly connected components S_1, S_2,….S_k.
9. Partition the ordinary vertices of Hq into blocks according to each sj, j = 1,….k; For each ordinary vertex v, $[v]_{2e}$ contains the ordinary vertices in the strongly connected component of v.

LEMMA 3. 6.1. If G(s) has b bridges, then all the auxiliary graphs Gr have at most n+2bvertices and m+ 2b edges in total.

PROOF. Every vertex appears as an ordinary vertex only in one auxiliary graph. Amarked vertex in D(s) corresponds to a bridge in G(s), so there are $b \leq n - 1$ markedvertices. Since we have one auxiliary graph for each marked vertex, the total numberof the auxiliary vertices d(r) is b. Each marked vertex v can also appear in at most oneother auxiliary graph as a child of a boundary vertex. So, the total number of verticesis at most n + 2b. Next, we bound the total number of edges. Excluding bridges, thetotal number of edges between two ordinary vertices or between an ordinary vertexand an auxiliary vertex in all auxiliary graphs is at most m−b. Each bridge can appearin at most two auxiliary graphs. It remains to count the number of edges between twoauxiliary vertices. Each such edge is of the form (w, d(r)), where w and r are roots inthe canonical decomposition, and moreover, w is a marked child of a boundary vertexin the auxiliary graph Gr. Hence, there is at most one such edge leaving each markedvertex, so at most b overall. The total number of edges in all auxiliary graphs is then bounded by $m-b+ 2b+ b = m+ 2b$.

LEMMA 3.6.2.*Any digraph with n vertices has at most 2n− 2 strong bridges.*

Experimental studies for algorithms that compute dominators, strong bridges, andstrong articulation points are presented in Firmani et al. [2012] and Georgiadis et al.[2014]. The experimental results show that the corresponding fast algorithms given inFraczak et al. [2013], Italiano et al. [2012], Lengauer and Tarjan [1979], and Tarjan[1976] perform very well in practice, even on very large graphs.

LEMMA 3.6.3. *Algorithm* Fast2ECB *runs in* $O(m)$ *time.*

PROOF. We analyze the total time spent on each step that Algorithm Fast 2ECB executes. Step 1 takes $O(m)$ time by Buchsbaum et al. [2008], and Step 2 takes $O(m)$ time by Lemma 3.13. From Lemma 6.1, we have that the total number of verticesand the total number of edges in all auxiliary graphs H of G are $O(n)$ and $O(m)$,respectively. Therefore, the total number of strong bridges in these auxiliary graphs is $O(n)$ by Lemma 2.1. Then, by Lemma 3.9, the total size (number of vertices and edges)of all auxiliary graphs HRq for all H, computed in Step 5, is still $O(m)$, and they arealso computed in $O(m)$ total time by Lemma 3.13. So, Steps 5 and 7 take $O(m)$ time in total, as well.

IV.CONCLUSION

In this paper, we have surveyed important algorithms for finding out 2-connectivity in directed graphs, which are harder than their counterparts on the undirected graph.This recent lot of work has raised some interesting questions, whether the 2-edge connected or 2-vertex connected components can be computed in linear time?. Moreover, the dynamic maintenance of 2-edge and 2-vertex connectivity in directed graph deserves further investigation.

V.REFERENCES

1 Douglas West. Introduction to Graph Theory second edition.

2 K. Menger. ZurAllgemeinekurventheorie. *Fund. Math.*, 10:96–115, 1927.

3 http://www.geeksforgeeks.org/tag/graph-connectivity

4 https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial

5 R. Jaberi. Computing the 2-blocks of directed graphs. RAIRO-Theor. Inf. Appl.,49(2):93119,2015.doi:10.1051/ita/2015001.

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

7 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

8 http://www.dcc.fc.up.pt/~acm/PRinv.pdf

9 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

10 G. F. Italiano, L. Laura, F. Santaroni, Finding strong bridges and strong articulation points in linear time, Theoretical Computer Science. 447 (2012) 74–84.

11 R. Diestel, Graph Theory, 2nd ed., Springer, New York, 2000, pp. 43–44.

12 G. F. Italiano, L. Laura, F. Santaroni, Finding strong bridges and strong articulation points in linear time, Theoretical Computer Science. 447 (2012) 74–84.

13 R. E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput.1(2) (1972) 146–160.

14 S. Alstrup, D. Harel, P.W. Lauridsen, M. Thorup, Dominators in linear time.SIAM J. Comput. 28(6) (1999) 2117–2132.

15 A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, J. R.Westbrook, Linear-time algorithms for dominators and other path-evaluationproblems,SIAM J.Comput. 38(4) (2008) 1533–1573.

16 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4569431/

17 Stoer M. and Wagner F., A simple min-cut algorithm, *J. ACM* 44(4), 1997. doi: 10.1145/263867.263872

18 Ford L. R. and Fulkerson D. R., Flows in Networks, *Princeton University Press* Princeton, New Jersey, 1962

19 Goldberg A. V. and Tarjan R. E., A new approach to maximum flow problem, *J. ACM* 35(4), 1988. doi: 10.1145/48014.61051

20 D. Firmani, G. F. Italiano, L. Laura, A. Orlandi, and F. Santaroni. 2012. Computing strong articulationpoints and strong bridges in large scale graphs. In Proceedings of the 10th International Symposium on Experimental Algorithms. 195–207.

21 W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan. 2013. Finding Dominators via disjoint set union.Journal of Discrete Algorithms 23,DOI:http://dx.doi.org/10.1016/j.jda.2013.10.003

22 A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. 2008. Lineartimealgorithms for dominators and other path-evaluation problems. SIAM J. Comput. 38, 4 (2008)