Approximating the Smallest 2-Vertex-Connected Spanning Subgraph via Low-High Orders

Loukas Georgiadis[1], Giuseppe F. Italiano[2], and Aikaterini Karanasiou[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**
`Aikaterini.Karanasiou.@uniroma2.it`

Abstract

Let $G = (V, E)$ be a 2-vertex-connected directed graph with m edges and n vertices. We consider the problem of approximating the smallest 2-vertex connected spanning subgraph (2VCSS) of G, and provide new efficient algorithms for this problem based on a clever use of low-high orders. The best previously known algorithms were able to compute a 3/2-approximation in $O(m\sqrt{n}+n^2)$ time, or a 3-approximation faster in linear time. In this paper, we present a linear-time algorithm that achieves a better approximation ratio of 2, and another algorithm that matches the previous 3/2-approximation in $O(m\sqrt{n} + n^2)$ time. We conducted a thorough experimental evaluation of all the above algorithms on a variety of input graphs. The experimental results show that both our two new algorithms perform well in practice. In particular, in our experiments the new 3/2-approximation algorithm was always faster than the previous 3/2-approximation algorithm, while their two approximation ratios were close. On the other side, our new linear-time algorithm yielded consistently better approximation ratios than the previously known linear-time algorithm, at the price of a small overhead in the running time.

1998 ACM Subject Classification E.1 [Data Structures] Graphs and Networks, Trees, G.2.2 [Graph Theory] Graph Algorithms

Keywords and phrases 2-vertex connectivity, approximation algorithms, directed graphs

Digital Object Identifier 10.4230/LIPIcs.SEA.2017.9

1 Introduction

The problem of approximating subgraphs that satisfy certain connectivity requirements has received a lot of attention (see, e.g., [9], and the survey [21]). In general, computing efficiently small spanning subgraphs that retain some desirable properties of an input graph is of particular importance when dealing with large-scale networks (e.g., networks with hundreds of million to billion edges), which arise often in today's applications. In this framework, designing practically efficient algorithms is also of the utmost importance. In particular, one of the biggest challenge is to design fast linear-time algorithms, since algorithms with higher running times might be practically infeasible on large-scale networks.

Before defining formally our problems, we need some preliminary definitions. Let $G = (V, E)$ be a strongly connected directed graph (digraph) with m edges and n vertices. A vertex x of G is a *strong articulation point* if $G \setminus x$ is not strongly connected, i.e., the removal of x destroys the strong connectivity of G. A strongly connected digraph G is *2-vertex-connected* if it has at least three vertices and no strong articulation points. More generally, a strongly connected digraph is *k-vertex connected* if it has at least $k+1$ vertices and the removal of any set of at most $k-1$ vertices leaves the graph strongly connected. The computation of a smallest (i.e., with minimum number of edges) k-vertex-connected spanning subgraph (kVCSS) of a given k-vertex-connected graph is a fundamental problem in network design.

In this paper, we consider the problem of approximating the smallest 2-vertex connected spanning subgraph (2VCSS) of a 2-vertex connected digraph G. The current best approximation ratio for this problem is 3/2, and it was achieved first by the algorithm by Cheriyan and Thurimella [5], which runs in $O(m^2)$ time. Georgiadis [11] presented a faster linear-time algorithm which achieves a 3-approximation. He then combined his algorithm with the 3/2-approximation algorithm of Cheriyan and Thurimella [5] to achieve a new 3/2-approximation algorithm which runs in faster $O(m\sqrt{n}+n^2)$ time. As explicitly mentioned in [11], the previous experimental study on approximation algorithms for the 2VCSS problem by Georgiadis [11] focused mainly on the solution quality achieved in practice, and not much effort was put into optimizing the running time of the algorithms considered.

The main contributions of this paper are two new efficient algorithms for this problem which exploit in a novel fashion the low-high order of a digraph [14]. Specifically, we first provide a linear-time algorithm that achieves a better approximation ratio of 2, thus improving significantly the best previous approximation ratio achievable in linear time for this problem [11]. Next, we show how to combine our new linear-time algorithm with the 3/2-approximation algorithms of Cheriyan and Thurimella [5] for 2VCSS and of Zhao et al. [27] for approximating the smallest strongly connected spanning subgraph (SCSS), so as to obtain an algorithm that achieves a 3/2-approximation in $O(m\sqrt{n}+n^2)$ time for 2VCSS. Hence, our new algorithm matches the previously known best bounds of [11].

To assess their practical value, we conducted a thorough experimental evaluation of all the above algorithms on a variety of input graphs. In order to make a fair comparison, in addition to the efficient implementations of our new algorithms, we also provide newly engineered and faster implementations of the algorithms by Georgiadis [11], which have better running times in practice while still achieving the same approximation ratios.

Our experimental results show that both our two new algorithms perform well in practice. In particular, in our experiments the new 3/2-approximation algorithm kept essentially the same approximation ratio as the previous algorithm, but it was significantly faster. On the other side, our new linear-time algorithm yielded consistently better approximation ratios than the previously known linear-time algorithm, at the price of a small overhead in the running time.

We observe that recent work [12, 13] considered also slightly more general problems than the one considered in this paper, such as approximating the smallest strongly connected spanning subgraph that maintains 2-connectivity relations of a strongly connected digraph G (where G is not necessarily 2-vertex-connected). Some of the results in this paper extend directly to this setting as well. For instance, our new linear-time 2-approximation algorithm for 2VCSS immediately implies a linear-time 2-approximation algorithm for computing the smallest strongly connected spanning subgraph of G that maintains the maximal 2-vertex-connected subgraphs of G.

2 Preliminaries

In this section, we review some basic notions and results used in our algorithms. A *flow graph* $G=(V,E,s)$ is a directed graph (digraph) with a distinguished start vertex $s \in V$

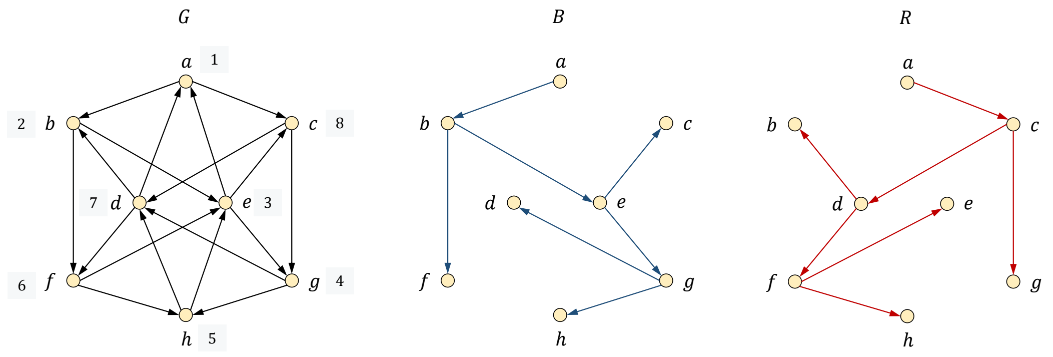


Figure 1 A 2-vertex-connected digraph G with vertices numbered in a low-high order (left); two divergent spanning trees B and R of G rooted at vertex a (right).

such that all vertices are reachable from s. The *dominator relation* in G is defined as follows. A vertex v is a *dominator* of a vertex w (v *dominates* w) if every path from s to w contains v; v is a *proper dominator* of w if v dominates w and $v \neq w$. The dominator relation in G can be represented by a tree rooted at s, the *dominator tree* D, such that v dominates w if and only if v is an ancestor of w in D. Throughout the paper, for each vertex $v \neq s$ we let $d(v)$ denote the parent of v in D. The dominator tree is a central tool in program optimization and code generation [6], and it has applications in other diverse areas [16]. The dominator tree of a flow graph can be computed in linear time [1, 4].

A spanning tree T of a flow graph G is a tree with root s that contains a path from s to v for all vertices v.

Given a rooted tree T, we denote by $T(v)$ the subtree of T rooted at v (we also view $T(v)$ as the set of descendants of v).

Let T be a tree rooted at s with vertex set V, and let $t(v)$ denote the parent of a vertex v in T. If v is an ancestor of w, we denote by $T[v, w]$ the path from v to w in T. In particular, $D[s, v]$ consists of the vertices that dominate v. If v is a proper ancestor of w, $T(v, w]$ is the path to w from the child of v that is an ancestor of w. A tree T is *flat* if its root is the parent of every other vertex.

A *preorder* of T is a total order of the vertices of T such that, for every vertex v, the descendants of v are ordered consecutively, with v first.

A *low-high order* δ *of* G [14] is a preorder of the dominator tree D with the following property: for all vertices $v \neq s$, either $(d(v), v) \in E$ or there are two edges $(u, v) \in E$, $(w, v) \in E$ such that u is less than v ($u <_\delta v$), v is less than w ($v <_\delta w$), and w is not a descendant of v in D. . Note that if D is flat, then the above definition of a low-high order δ is simplified as follows: for all vertices $v \neq s$, either $(s, v) \in E$ or there are two edges $(u, v) \in E$, $(w, v) \in E$ such that $u <_\delta v$ and $v <_\delta w$. See Figure 1. Every flow graph G has a low-high order, computable in linear-time [14]. Low-high orders provide a correctness certificate for dominator trees that is straightforward to verify [26], and also have applications in path-determination problems [14, 25] and in fault-tolerant network design [2, 3, 15].

Let $G = (V, E, s)$ be a flow graph, and let D be a dominator tree of G. Fix a low-high order δ of G and let $E' \subseteq E$ be a subset of edges of G. We say that E' *satisfies* δ if for any vertex $v \neq s$ we have that either $(d(v), v) \in E'$ or there are two edges $(u, v) \in E'$, $(w, v) \in E'$ such that $u <_\delta v$ and $v <_\delta w$, and w is not a descendant of v in D. If $E' \subseteq E$ satisfies δ, then $G' = (V, E', s)$ is a flow graph with the same dominator tree as G.

A notion closely related to low-high orders is that of divergent spanning trees [14].

Let $G = (V, E, s)$ be a flow graph. Two spanning trees B and R of G, rooted at s, are *divergent* if for all v, the paths from s to v in B and R share only the dominators of v, i.e., $B[s, v] \cap R[s, v] = D[s, v]$.

Every flow graph has a pair of divergent spanning trees. Given a low-high order of G, two divergent spanning trees of G can be computed in time $O(m + n)$ [14].

Let $G = (V, E)$ be a strongly connected graph. Note that, for any arbitrarily selected start vertex s in V, $G_s = (V, E, s)$ is a flow graph. Since there is no danger of ambiguity, in the following we will denote by G both the original strongly connected graph G and the flow graph G_s. We denote by $G^R = (V, E^R)$ the *reverse digraph* of G that results from G after reversing all edge directions. Arbitrarily fix a start vertex s in V: similarly to before, we also denote by G^R the flow graph with start vertex s, and by D^R the dominator tree of the flow graph G^R. As proved in [19], a vertex $v \neq s$ is a strong articulation point of G if and only if v is not a leaf in D or not a leaf in D^R. This implies the following property, which will be used throughout the paper:

▶ **Property 1.** *A strongly connected graph G is 2-vertex-connected if and only if:*
(a) *Both D and D^R are flat, and*
(b) *$G \setminus s$ is strongly connected.*

3 A linear-time 2-approximation algorithm

In this section, we present our new linear-time algorithm that computes a 2-approximation to the smallest 2-vertex-connected subgraph (2VCSS) of a 2-vertex-connected digraph G. The algorithm, which we call LH-Z, exploits the properties of low-high orders and uses the algorithm of Zhao et al. [27] for computing approximate smallest strongly connected spanning subgraphs (SCSS). LH-Z, described in Algorithm 1 as pseudocode, works as follows. We first choose arbitrarily a vertex s in G and start with an approximate smallest strongly connected spanning subgraph H of $G \setminus s$, which can be computed with the algorithm of Zhao et al. [27] (lines 1–3). We then compute a low-high order of the flow graph G with start vertex s (line 4); next, we add edges to H so as to ensure that the edge set of H satisfies δ, that is, δ is also a low-high order for all vertices $v \neq s$ in H (lines 5–17). This step is repeated also for the reverse flow graph G^R, with the same start vertex s (line 18). We start by proving that the spanning subgraph computed by Algorithm LH-Z is 2-vertex-connected.

▶ **Lemma 2.** *Algorithm LH-Z computes a 2-vertex-connected spanning subgraph of G.*

Proof. Let H be the subgraph computed by LH-Z. To show that H is 2-vertex connected, we prove that Property 1 holds. Note that (b) trivially holds because H is initially a strongly connected spanning subgraph of $G \setminus s$ (line 2), and it remains so after adding edges. It thus remains to show that both H and H^R have flat dominator trees. We only prove this for H, since the same argument applies to H^R.

Let δ be the low-high order of G computed in line 4. We argue that after the execution of the for loop in lines 5–17, δ must be also a low-high order in H (i.e., the edges of H satisfy δ). Consider an arbitrary vertex $v \neq s$. Let (x, v) be an edge entering v in the initial strongly connected spanning subgraph of G computed in line 2. If $x >_\delta v$, then, by the definition of δ, there is at least one edge $(y, v) \in E$ such that $y <_\delta v$. (Note that we can have $y = s$ since $s <_\delta v$ for all $v \neq s$.) Hence, after the execution of the for loop for v, the edge set E_H will contain at least two edges (u, v) and (w, v) such that $u <_\delta v <_\delta w$. On the other hand, if $x <_\delta v$, then the definition of δ implies that there is an edge $(y, v) \in E$ such that $y >_\delta v$ or $y = s$. Notice that in either case $y \neq x$ because (x, v) is an edge of $G \setminus s$. So, again, after the

Algorithm 1: LH-Z(G)

```
   Input: 2-vertex-connected digraph $G = (V, E)$
   Output: 2-approximation of a smallest 2-vertex-connected spanning subgraph
           $H = (V, E_H)$ of $G$
1  Choose an arbitrary vertex $s$ of $G$ as start vertex.
2  Compute a strongly connected spanning subgraph $H = (V \setminus s, E_H)$ of $G \setminus s$.
3  Set $H \leftarrow (V, E_H)$.
4  Compute a low-high order $\delta$ of flow graph $G$ with start vertex $s$.
5  foreach vertex $v \neq s$ do
6      if there are two edges $(u, v)$ and $(w, v)$ in $E_H$ such that $u <_\delta v$ and $v <_\delta w$ then
7          do nothing
8      end
9      else if there is no edge $(u, v) \in E_H$ such that $u <_\delta v$ then
10         find an edge $e = (u, v) \in E$ with $u <_\delta v$
11         set $E_H \leftarrow E_H \cup \{e\}$
12     end
13     else if there is no edge $(w, v) \in E_H$ such that $v <_\delta w$ then
14         find an edge $e = (w, v) \in E$ with $v <_\delta w$ or $w = s$
15         set $E_H \leftarrow E_H \cup \{e\}$
16     end
17 end
18 Execute the analogous steps of lines 4–17 for the reverse flow graph $G^R$ with start
     vertex $s$.
19 return $H = (V, E_H)$
```

execution of the for loop for v, the edge set E_H will contain at least two edges (u, v) and (w, v) such that either $u <_\delta v <_\delta w$, or $u <_\delta v$ and $w = s$. It follows that δ is a low-high order for all vertices $v \neq s$ in H.

As proved in [14], this implies that H contains two divergent spanning trees B and R of G. Since G is 2-vertex-connected, it has a flat dominator tree, and thus we have that $B[s, v] \cap R[s, v] = \{s, v\}$ for all $v \in V \setminus s$. Hence, since H contains B and R, the dominator tree of H is also flat. ◀

We remark that the construction of H in algorithm LH-Z guarantees that s will have in-degree and out-degree at least 2 in H. (This fact is implicit in the proof of Lemma 2.) Indeed, H will contain the edges from s to the vertices in $V \setminus s$ with minimum and maximum order in δ, and the edges entering s from the vertices in $V \setminus s$ with minimum and maximum order in δ^R.

▶ **Theorem 3.** *Algorithm LH-Z computes a 2-approximation for 2VCSS in linear time.*

Proof. We first establish the approximation ratio of LH-Z by showing that $|E_H| \leq 4n$. The approximation ratio of 2 follows from the fact that any vertex in a 2-vertex-connected digraph must have in-degree at least two. In line 2 we can compute an approximate smallest strongly connected spanning subgraph H of $G \setminus s$ [20]. For this, we can use the linear-time algorithm of Zhao et al. [27], which selects at most $2(n-1)$ edges. Now consider the edges selected in the for loop of lines 5–17. Since after line 2, $H \setminus s$ is strongly connected, each vertex $v \in V \setminus s$ has at least one entering edge (x, v). If $x <_\delta v$ then lines 10–11 will not be executed;

otherwise, $v <_\delta x$ and lines 14–15 will not be executed. Thus, the for loop of lines 5–17 adds at most one edge entering each vertex $v \neq s$. The same argument implies that the analogous steps executed for G^R add at most one edge leaving each vertex $v \neq s$. Hence, at the end of the execution E_H contains at most $4(n-1)$ edges.

Note that the algorithm by Zhao et al. [27] runs in linear time, and a low-high order can also be computed in linear-time [14]. Furthermore, all other steps of Algorithm LH-Z can be implemented in linear time. This yields the lemma. ◀

▶ Remark. In line 2 of algorithm LH-Z, we can alternatively set H to be the union of two spanning trees as follows. We choose an arbitrary vertex $s' \neq s$ as the start vertex of $G \setminus s$ and compute two spanning trees T and T^R of the flow graphs $G \setminus s$ and $(G \setminus s)^R$, respectively, rooted at s'. Then, we let H consist of the edges $\{(u,v) : (u,v) \in T\} \cup \{(u,v) : (v,u) \in T^R\}$, which are at most $2(n-1)$ as required by the proof of Theorem 3. In our implementation, however, we use the algorithm of Zhao et al. [27] instead. This way, we obtained better results in practice.

4 A 3/2-approximation algorithm

In this section we present a new algorithm, called LH-Z-CT, that combines our linear-time algorithm LH-Z described in Section 3 with the 3/2-approximation algorithm of Cheriyan and Thurimella [5]. We first describe a simple filtering algorithm that computes a minimal 2VCSS, and then give an overview of the Cheriyan-Thurimella algorithm.

Let $G = (V, E)$ be the input 2-vertex-connected digraph. A simple $O(m^2)$-time algorithm that gives a 2-approximation $G' = (V, E')$ of the smallest 2VCSS of G filters out redundant edges as follows: Initially, we set $G' = G$. Then, we process the edges of E in an arbitrary order: when we process an edge (x, y) we test if $G' \setminus (x, y)$ contains at least two vertex-disjoint paths from x to y. If this is the case, then we remove the edge (x, y) from E'; otherwise, we keep the edge (x, y) in E' and proceed with the next edge. Clearly, at the end of this procedure G' is a minimal 2VCSS of G, i.e., for any edge $(x, y) \in E'$, $G' \setminus (x, y)$ is not 2-vertex-connected. We refer to this algorithm as MINIMAL.

Testing if a digraph G has two vertex-disjoint paths from x to y can be done in $O(m)$ time by using two iterations of the Ford-Fulkerson flow-augmenting method [10]. The Ford-Fulkerson method actually finds edge-disjoint paths, but we can also compute vertex-disjoint paths after applying vertex-splitting. Specifically, we create a modified graph $\overline{G} = (\overline{V}, \overline{E})$ that results from G as follows. The vertex set $\overline{V}$ contains a pair of vertices v^- and v^+ for each vertex $v \in V$. The edge set $\overline{E}$ contains the edges (v^-, v^+) corresponding to all $v \in V$. Also, for each edge $(v, w) \in E$, we include the edge (v^+, w^-) in $\overline{E}$. It is easy to see that there are k vertex-disjoint paths from x to y in G if and only if there are k edge-disjoint paths from x^+ to y^- in $\overline{G}$. Note that $\overline{G}$ has $2n$ vertices and $m + n$ edges.

The algorithm by Cheriyan and Thurimella [5] (CT) uses matchings in order to improve the approximation guarantee of MINIMAL. Let $M \subseteq E$ be a set of edges such that every vertex has indegree and outdegree at least one in the subgraph having vertex set V and edge set M.

We call a minimum such set M a 1-*matching* of G. This can be computed in time $O(m\sqrt{n})$ via a reduction to maximum bipartite matching [18]. After computing M, the CT algorithm executes a slightly modified filtering phase, which applies the two vertex-disjoint paths test to all edges in $E \setminus M$. Hence, CT computes a subgraph $G' = (V, E')$ of G, where $E' = M \cup F$ and F is a minimal set of edges of G such that G' is 2-vertex-connected. Algorithm CT also runs in $O(m^2)$ time.

Algorithm 2: LH-Z-CT(G)

```
Input: 2-vertex-connected digraph $G = (V, E)$
Output: 3/2-approximation of a smallest 2-vertex-connected spanning subgraph
        $H = (V, E_H)$ of $G$
1  Compute a 1-matching $M$ of $G$.
2  Choose an arbitrary vertex $s$ of $G$ as start vertex.
3  Let $G'$ be the subgraph of $G \setminus s$, for arbitrary start vertex $s$, that contains only the
     edges in $M$.
4  Compute the strongly connected components $C_1, \dots, C_k$ in $G'$.
5  Form a contracted version $\breve{G}$ of $G \setminus s$ as follows. For each strongly connected
     component $C_i$ of $G'$, we contract all vertices in $C_i$ into a representative vertex
     $u_i \in C_i$.
6  Compute a strongly connected spanning subgraph $\breve{G}'$ of $\breve{G}$. Let $Z$ be the original
     edges of $G$ that correspond to the edges of $\breve{G}$ selected in $\breve{G}'$.
7  Set $H \leftarrow (V, E_H = M \cup Z)$.
8  Compute a low-high order $\delta$ of flow graph $G$ with start vertex $s$.
9  foreach vertex $v \neq s$ do
10     if there are two edges $(u, v)$ and $(w, v)$ in $E_H$ such that $u <_\delta v$ and $v <_\delta w$ then
11         do nothing
12     end
13     else if there is no edge $(u, v) \in E_H$ such that $u <_\delta v$ then
14         find an edge $e = (u, v) \in E$ with $u <_\delta v$
15         set $E_H \leftarrow E_H \cup \{e\}$
16     end
17     else if there is no edge $(w, v) \in E_H$ such that $v <_\delta w$ then
18         find an edge $e = (w, v) \in E$ with $v <_\delta w$ or $w = s$
19         set $E_H \leftarrow E_H \cup \{e\}$
20     end
21 end
22 Execute the analogous steps of lines 4–17 for the reverse flow graph $G^R$ with start
     vertex $s$.
23 foreach edge $(x, y)$ of $E_H \setminus M$ do
24     if there are two vertex-disjoint paths from $x$ to $y$ in $H \setminus (x, y)$ then
25         Set $E_H \leftarrow E_H \setminus (x, y)$.
26     end
27 end
28 return $H = (V, E_H)$
```

Our Algorithm LH-Z-CT (whose pseudocode is described below) works as follows. First, it computes a 1-matching M as CT. Let s be an arbitrary start vertex, and let G' be the subgraph of $G \setminus s$ that contains only the edges in M. We compute the strongly connected components $C_1, \dots, C_k$ in G', and form a contracted version $\breve{G}$ of $G \setminus s$ as follows. For each strongly connected component C_i of G', we contract all vertices in C_i into a representative vertex $u_i \in C_i$.

Then, we execute the linear-time algorithm of Zhao et al. [27] to compute a strongly connected spanning subgraph of $\breve{G}$, and store the original edges of G that correspond to the selected edges by the Zhao et al. algorithm. Let Z be this set of edges. Next, we compute a low-high order of G with root s, and use it in order to compute a 2-vertex-connected spanning subgraph H of G using as many edges from Z and M as possible, as in LH-Z.

Then, we run the filtering phase of Cheriyan and Thurimella, as follows. For each edge (x, y) of H that is not in M, we test if x has two vertex-disjoint paths to y in $H \setminus (x, y)$. If it does, then we set $H \leftarrow H \setminus (x, y)$.

▶ **Theorem 4.** *Algorithm LH-Z-CT computes a $3/2$-approximation for 2VCSS in $O(m\sqrt{n}+n^2)$ time.*

Proof. First, we note that the spanning subgraph computed by algorithm LH-Z-CT is 2-vertex-connected since it satisfies Property 1. Indeed, let H' be the graph computed in lines 1–22. Then H' is 2-vertex-connected, since it contains a strongly connected spanning subgraph of $G \setminus s$, and a set of edges that satisfies a low-high order of G and G^R. Also, the filtering phase preserves the 2-vertex-connectivity of H.

Next, we establish the $3/2$ approximation ratio of LH-Z-CT by showing that a specific execution of CT produces the same output subgraph.

Let S be the set of edges of H' (i.e., the edges of H just after the execution of lines 1–22). Note that the approximation ratio of CT does not depend on the order that edges are processed during the filtering phase. Hence, we can assume that CT processes the edges of $E \setminus S$ first. Notice that for each $(x, y) \in E \setminus S$, H' contains two vertex-disjoint paths from x to y. Hence, each such edge will not be included in the subgraph computed by CT. So, if we fix the order in which the edges in S are processed, the filtering phase in both CT and LH-Z-CT will remove exactly the same redundant edges.

Finally, we consider the running time of LH-Z-CT. Line 1 takes $O(m\sqrt{n})$ time [18], and lines 2–5 take $O(m)$ time [24]. In line 6, we can compute a SCSS of $\breve{G}$ in $O(m)$ time [27], and in line 8 we can compute a low-high order of G in $O(m)$ time [14]. Finally, the loops in lines 9–2 and 23–27 take $O(m)$ and $O(n^2)$ time, respectively. ◀

We mention that in our implementation, the bipartite matching is computed via max-flow, using an implementation of the Goldberg-Tarjan push-relabel algorithm [17] from [7], which is known to be very fast in practice.

5 Empirical Analysis

For the experimental evaluation we use the graph datasets shown in Table 1, taken from the Koblenz Network Collection [22], the Stanford Network Analysis Project [23] and the 9th DIMACS Implementation Challenge [8]. For each tested graph, we computed its largest 2-vertex-connected subgraph and used that as input to our algorithms. We wrote our implementations in `C++`, using `g++ v.4.6.4` with full optimization (flag `-O3`) to compile the code. We report the running times on a GNU/Linux machine, with Ubuntu (16.04LTS): a Dell Inspiron 64-bit machine with Intel® Core ™ i7-7500U processor's seventh-generation (4 MB of cache, up to 3.5GHz) and 16 GB of RAM memory. In our experiments we did not use any parallelization, and each algorithm ran on a single core. We report CPU times measured with the `getrusage` function, averaged over ten different runs.

Since we do not know the size of the optimal 2VCSS in each of these graphs, we measure the quality of the produced solution $G' = (V', E')$ by calculating the relative distance from the naive theoretical lower bound, i.e., $\frac{|E'|-2|V|}{2|V|} \times 100\%$. We refer to this relative distance as the *quality measure* (qm).

Table 1 Real-world graphs used in the experiments. From each original graph, we extracted its largest 2-vertex-connected subgraph. The number of vertices n and of edges m refer to each such subgraph.

Graph	**2VCCs**		**Type**
	n	m	
amazon-302	55414	241663	co-purchase [23]
amazon-601	276049	2461072	co-purchase [23]
advogato	3140	35979	social [22]
rome99	2249	6467	road network [8]
soc-Epinions	17117	395183	trust network [23]
web-BerkStan-1	1106	8206	web [23]
web-BerkStan-2	4927	28142	web [23]
web-BerkStan-3	29145	439148	web [23]
web-Google	77480	840829	web [23]
web-NotreDame-1	1462	10195	web [23]
web-NotreDame-2	1409	9663	web [23]
web-NotreDame-3	1416	13226	web [23]
web-Stanford-1	5179	129897	web [23]
web-Stanford-2	10893	162295	web [23]
wiki-signed	14895	324776	online contact [22]
wikiTalk	49430	2461072	wiki communication [23]

5.1 Implemented Algorithms

In our experimental evaluation we compared a total of six algorithms for computing the (approximated) smallest 2-vertex-connected spanning subgraph. In addition to our two new algorithms, LH-Z (Section 3) and LH-Z-CT (Section 4), we also considered the algorithms from [11]: FAST, CT, MINIMAL and FAST-CT.

Algorithm FAST computes a 3-approximation in linear-time by using divergent spanning trees. Specifically, it computes the edges of two divergent spanning trees of G and of two divergent spanning trees of G^R so that it satisfies Property 1(a). Then it tests if these edges also satisfy Property 1(b), and if not it adds the edges of a SCSS of $G \setminus s$ by running the algorithm of Zhao et al. [27]. MINIMAL computes a 2-approximation in $O(m^2)$ time by applying the two vertex-disjoint paths test (see Section 4) and FAST-CT combines FAST with the 3/2-approximation algorithm of Cheriyan and Thurimella [5], which gives a 3/2-approximation in $O(m\sqrt{n} + n^2)$ time. In the experiments reported in [11], FAST achieved the best running times, while FAST-CT achieved the best solution quality.

Here, we also provide a new and faster implementation of CT, MINIMAL and FAST-CT, that we refer to as CT+, MINIMAL+ and FAST-CT+. The main improvement is in the implementation of the filtering phase. In the original implementations in [11], the filtering phase is performed by computing dominators in order to avoid computing the modified graph

$\overline{G}$ (which is obtained by applying vertex-splitting). Specifically, to test if $G' \setminus (x, y)$ has two vertex-disjoint paths from x to y, FAST-CT sets x as the start vertex of G' and computes the immediate dominator $d(y)$ of y. Such two paths exist if and only if $d(y) = x$. Our new implementations apply two iterations of the flow-augmentation method on the modified graph $\overline{G}$, as described in Section 4.

(We used the same implementation of the filtering phase in LH-Z-CT as well.)

We only report the comparison of the running times for FAST-CT and its improved implementation FAST-CT+ in Figure 2 and Table 2. (Both implementations produce the same solutions.) As it is evident, our improved implementation is faster by one order of magnitude. Similar speedups are obtained for CT+ and MINIMAL-CT+

5.2 Experimental Results

The running time and quality measure of LH-Z, FAST, LH-Z-CT, FAST-CT+, CT+, and MINIMAL+ are given in Tables 3 and 4, and plotted in Figures 3 and 4. Recall that the quality measure of each of the algorithms FAST-CT, CT, and MINIMAL from [11] is identical to the quality measure achieved by the corresponding improved implementations FAST-CT+, CT+, and MINIMAL+, respectively. Hence, we do not report the results of the former implementations. It is easy to observe that the algorithms belong to two distinct classes, with FAST and LH-Z being faster than the rest by approximately four to five orders of magnitude. One the other hand, on average they produce a 2VCSS with about 10–20% more edges.

Since for large scale graphs it is important to be able to compute a good solution very fast, it is interesting to compare the performance of the linear-time algorithms FAST and LH-Z. We observe that in all test cases LH-Z was able to compute a 2VCSS with 6–25% fewer edges at the price of a small overhead in the running time.

In our next experiment, we compare algorithms FAST-CT+ and LH-Z-CT which produce the best solutions overall. Observe that LH-Z-CT is always faster, mainly due to the fact that it has to process fewer edges during the filtering phase. Moreover, in some instances LH-Z-CT is significantly faster; e.g., its more than 45% times faster for wiki-signed and amazon-601. FAST-CT+ on the other hand, produced better solutions in 10 out the 14 test graphs, but the difference is marginal (at most 6.7% fewer edges).

Finally, we consider the performance of CT+ and MINIMAL+. Notice that although MINIMAL+, rather surprisingly, computes better solutions in 3 test graphs, its performance is rather unstable. Observe, for instance, that for 3 graphs (web-Stanford-2, web-BerkStan-3 and web-Google) it computes a worse solution than LH-Z. CT+ seems more robust in that sense, but with the exception of 4 graphs it computed an inferior solution compared to FAST-CT+, while being significantly slower.

Hence, our main conclusions are the following:

- Our new linear-time algorithm, LH-Z, computes a 2VCSS of reasonable quality very fast. Hence, if one wants a fast and good solution LH-Z is the right choice.
- Executing the filtering phase on a sparse subgraph of the input digraph, produced by either FAST or LH-Z, not only decreases the running time drastically, but also helps to compute a smaller 2VCSS in the end.

Acknowledgments. We are grateful to the authors of [7] for providing the implementation of their algorithm.

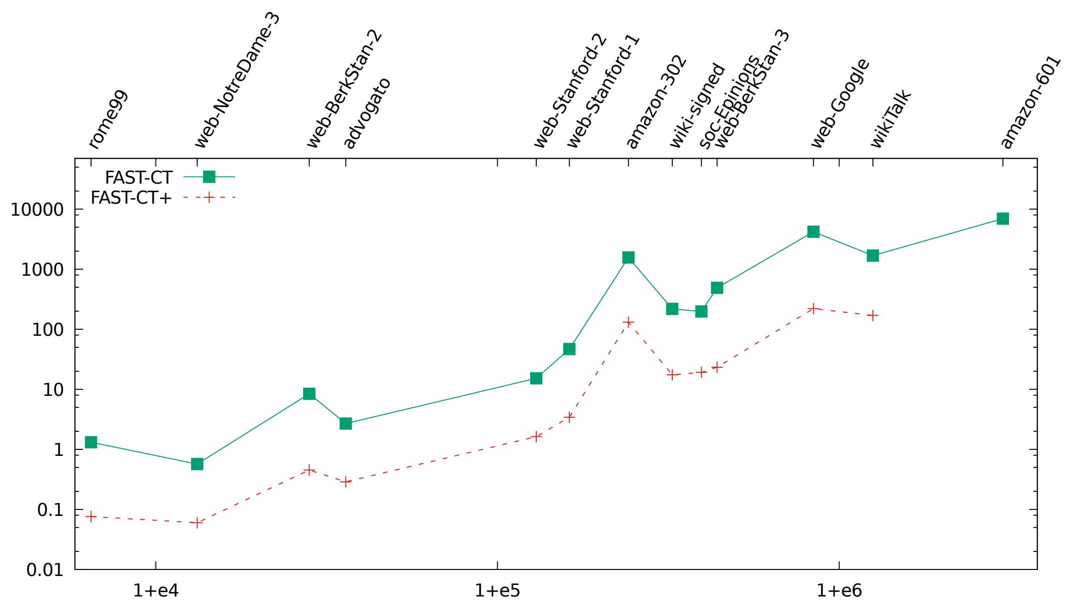


Figure 2 Running times (in seconds) of FAST-CT vs our improved implementation FAST-CT+. (The data on the experiment with amazon-601 is not reported, because FAST-CT took too long to finish.)

Table 2 Running times (in seconds) of the plot shown in Figure 2.

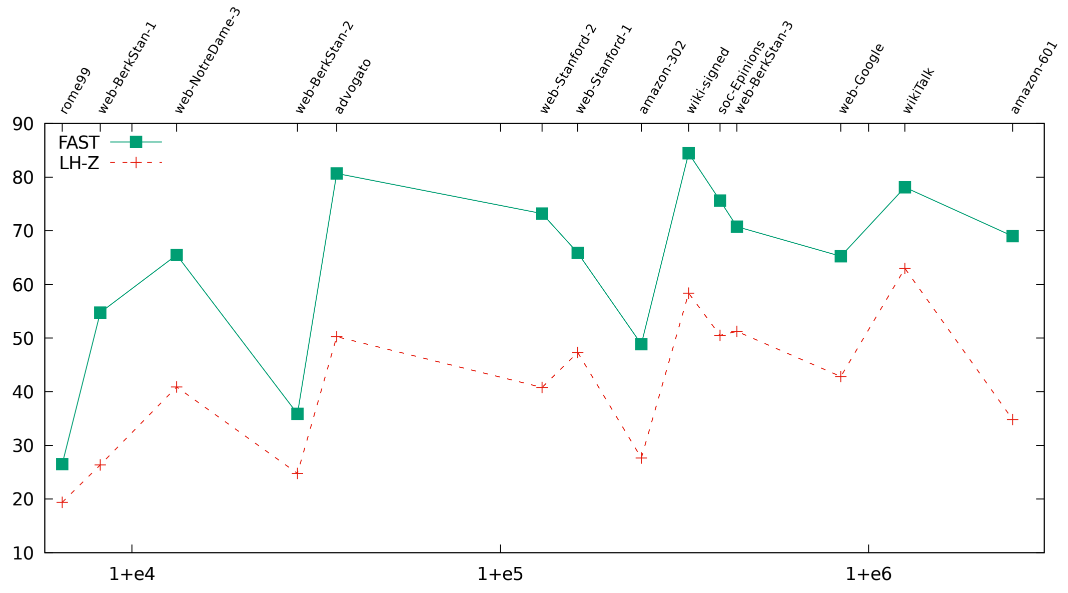
Graph	FAST-CT+	FAST-CT
rome99	0.076	1.312
web-BerkStan-1	0.044	0.220
web-NotreDame-3	0.060	0.568
web-BerkStan-2	0.452	8.404
advogato	0.352	2.692
web-Stanford-2	1.628	15.228
web-Stanford-1	3.424	46.800
amazon-302	131.376	1569.784
wiki-signed	17.436	217.672
soc-Epinions	19.132	197.940
web-BerkStan-3	23.264	488.604
web-Google	220.480	4200.888
wikiTalk	169.580	1689.104
amazon-601	6959.168	>24h

Table 3 Solution Quality (Quality Measure) of all algorithms that we considered.

Graph	LH-Z	FAST	LH-Z-CT	FAST-CT+	CT+	MINIMAL+
rome99	19.39	26.52	7.65	7.16	6.85	6.56
web-BerkStan-1	26.36	54.75	9.13	6.69	21.75	4.66
web-NotreDame-3	40.89	65.50	24.61	26.20	39.83	33.47
web-BerkStan-2	24.78	35.89	9.18	7.66	17.16	20.25
advogato	50.26	80.69	20.84	16.83	16.57	20.95
web-Stanford-2	40.79	73.20	14.94	14.36	32.02	47.78
web-Stanford-1	53.74	65.88	27.70	28.95	41.18	51.81
amazon-302	27.65	48.86	10.59	11.46	13.68	9.61
wiki-signed	58.37	84.44	36.61	28.02	27.50	29.80
soc-Epinions	50.50	75.63	21.98	19.86	20.59	27.56
web-BerkStan-3	51.25	70.76	35.38	32.45	41.23	52.25
web-Google	42.83	65.24	24.88	24.40	35.60	44.95
wikiTalk	62.99	78.11	40.91	38.91	37.38	38.33
amazon-601	34.82	68.99	10.16	12.74	24.64	17.08

Table 4 Running times (in seconds) of all algorithms that we considered.

Graph	LH-Z	FAST	LH-Z-CT	FAST-CT+	CT+	MINIMAL+
rome99	0.001	0.001	0.062	0.076	0.092	0.144
web-BerkStan-1	0.001	0.001	0.016	0.044	0.060	0.052
web-NotreDame-3	0.004	0.001	0.036	0.060	0.088	0.088
web-BerkStan-2	0.004	0.004	0.332	0.452	1.148	1.792
advogato	0.004	0.004	0.196	0.352	0.792	0.916
web-Stanford-2	0.012	0.008	1.380	1.628	4.640	7.668
web-Stanford-1	0.012	0.016	2.472	3.424	6.948	11.104
amazon-302	0.040	0.036	86.120	131.376	156.956	195.064
wiki-signed	0.028	0.024	9.132	17.436	42.984	66.948
soc-Epinions	0.040	0.028	13.792	19.132	58.780	79.216
web-BerkStan-3	0.028	0.024	20.072	23.264	53.524	77.900
web-Google	0.084	0.072	168.096	220.480	378.764	687.964
wikiTalk	0.116	0.092	140.464	169.580	838.528	1191.800
amazon-601	0.396	0.400	3678.768	6959.168	69434.112	11374.128



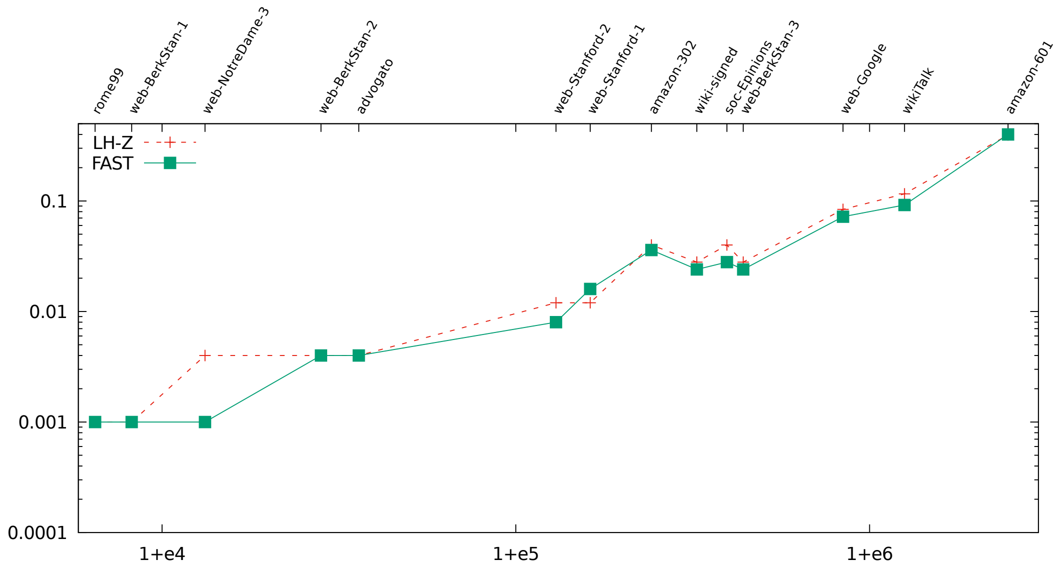
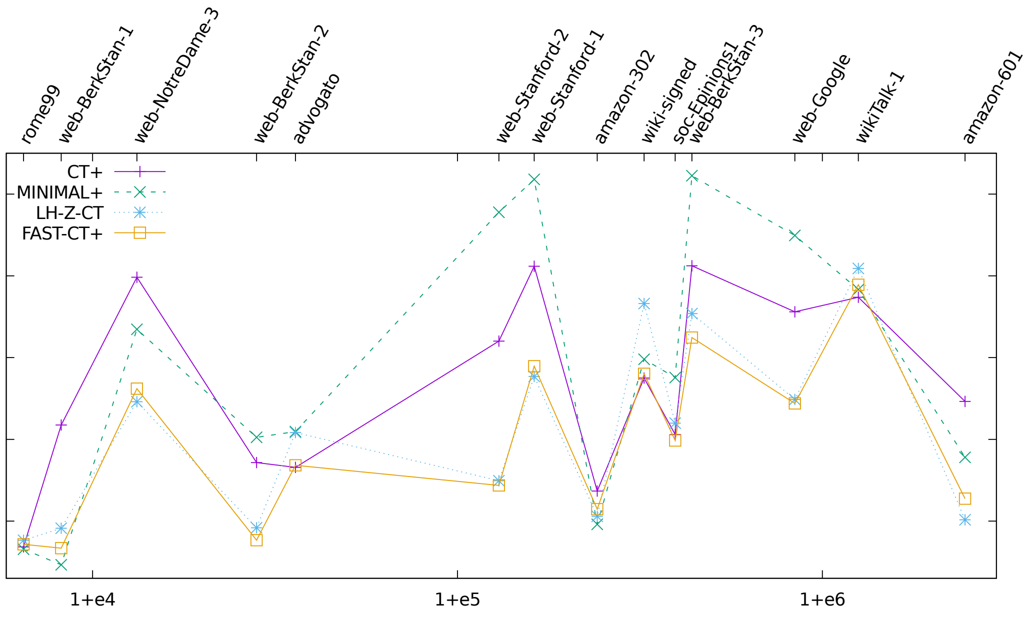


Figure 3 Performance of linear-time algorithms: solution quality (top) and running time in seconds (bottom). Running times and graph sizes (number of edges) are shown in log scale.



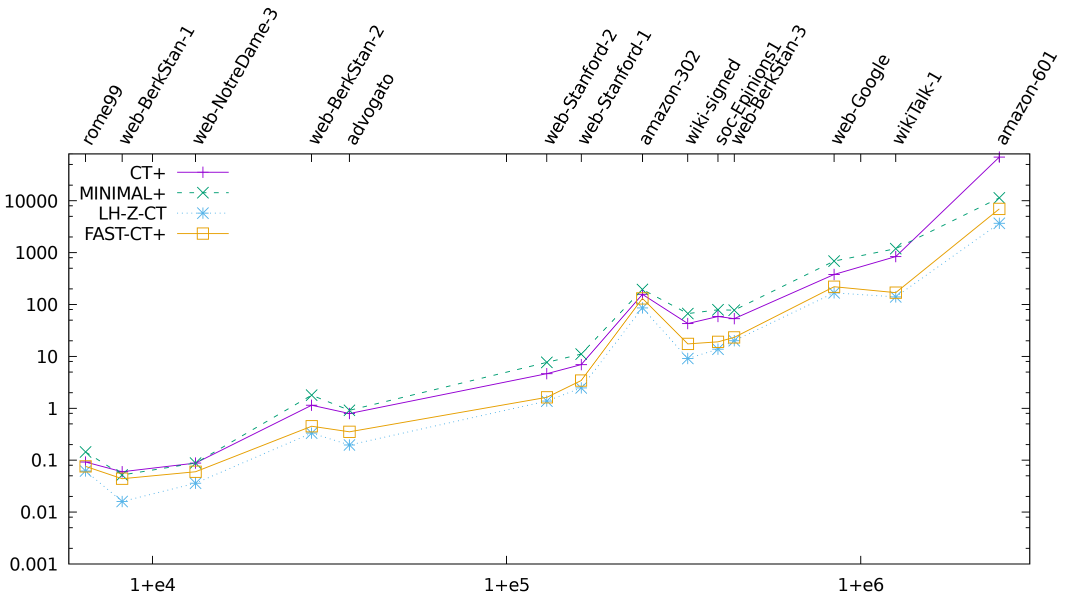


Figure 4 Performance of algorithms CT+, MINIMAL+, LH-Z-CT and FAST-CT+: solution quality (top) and running time (bottom). (Better viewed in color.) Running times and graph sizes (number of edges) are shown in log scale.

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 S. Baswana, K. Choudhary, and L. Roditty. Fault tolerant reachability for directed graphs. In Yoram Moses, editor, *Distributed Computing*, volume 9363 of *Lecture Notes in Computer Science*, pages 528–543. Springer Berlin Heidelberg, 2015.

3 S. Baswana, K. Choudhary, and L. Roditty. Fault tolerant reachability subgraph: Generic and optimal. In *Proc. Symposium on Theory of Computing*, pages 509–518, 2016.

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

6 R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991.

7 D. Delling, A. V. Goldberg, I. Razenshteyn, and R. F. Werneck. Graph partitioning with natural cuts. In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, 2011.

8 C. Demetrescu, A. V. Goldberg, and D. S. Johnson. 9th DIMACS Implementation Challenge: Shortest Paths, 2007. URL: `http://www.dis.uniroma1.it/~challenge9/`.

9 J. Fakcharoenphol and B. Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In *Proc. Symposium on Theory of Computing*, STOC'08, pages 153–158, New York, NY, USA, 2008. ACM.

10 D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010.

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

12 L. Georgiadis, G. F. Italiano, A. Karanasiou, C. Papadopoulos, and N. Parotsidis. Sparse subgraphs for 2-connectivity in directed graphs. In *Proc. Symposium on Experimental Algorithms*, (SEA 2016), pages 150–166, 2016.

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

14 L. Georgiadis and R. E. Tarjan. Dominator tree certification and divergent spanning trees. *ACM Transactions on Algorithms*, 12(1):11:1–11:42, November 2015.

15 L. Georgiadis and R. E. Tarjan. Addendum to "Dominator tree certification and divergent spanning trees". *ACM Transactions on Algorithms*, 12(4):56:1–56:3, August 2016.

16 L. Georgiadis, R. E. Tarjan, and R. F. Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications (JGAA)*, 10(1):69–94, 2006.

17 A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35:921–940, October 1988.

18 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

19 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.

20 S. Khuller, B. Raghavachari, and N. E. Young. Approximating the minimum equivalent digraph. *SIAM J. Comput.*, 24(4):859–872, 1995. Announced at *SODA 1994*, 177-186.

21 G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. *Approximation Algorithms and Metaheuristics*, 2007.

22 Jérôme Kunegis. KONECT: the Koblenz network collection. In *22nd International World Wide Web Conference, WWW'13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1343–1350, 2013.

23 J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

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

25 T. Tholey. Linear time algorithms for two disjoint paths problems on directed acyclic graphs. *Theoretical Computer Science*, 465:35–48, 2012.

26 J. Zhao and S. Zdancewic. Mechanized verification of computing dominators for formalizing compilers. In *Proc. 2nd International Conference on Certified Programs and Proofs*, pages 27–42. Springer, 2012.

27 L. Zhao, H. Nagamochi, and T. Ibaraki. A linear time 5/3-approximation for the minimum strongly-connected spanning subgraph problem. *Information Processing Letters*, 86(2):63–70, 2003.