HAL
open science

Approximating the smallest 2-vertex connected spanning subgraph of a directed graph

Loukas Georgiadis, Giuseppe F Italiano, Aikaterini Karanasiou

To cite this version:

Loukas Georgiadis, Giuseppe F Italiano, Aikaterini Karanasiou. Approximating the smallest 2-vertex connected spanning subgraph of a directed graph. Theoretical Computer Science, 2019, pp.1-16. 10.1016/j.tcs.2019.09.040 . hal-02335015

HAL Id: hal-02335015

https://inria.hal.science/hal-02335015v1

Submitted on 28 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Approximating the Smallest 2-Vertex Connected Spanning Subgraph of a Directed Graph*

Loukas Georgiadis[1] Giuseppe F. Italiano[2] Aikaterini Karanasiou[2]

Abstract

We consider the problem of approximating the smallest 2-vertex connected spanning subgraph (2VCSS) of a 2-vertex connected directed graph, and provide new efficient algorithms. We provide two linear-time algorithms, the first based on a linear-time test for 2-vertex connectivity and divergent spanning trees, and the second based on low-high orders, that correspondingly give 3- and 2-approximations. Then we show that these linear-time algorithms can be combined with an algorithm of Cheriyan and Thurimella that achieves a 3/2-approximation. The combined algorithms preserve the 3/2 approximation guarantee of the Cheriyan-Thurimella algorithm and improve its running time from $O(m^2)$ to $O(m\sqrt{n}+n^2)$, for a digraph with n vertices and m edges. Finally, we present an experimental evaluation of the above algorithms for a variety of input data. The experimental results show that our linear-time algorithms perform very well in practice. Furthermore, the experiments show that the combined algorithms not only improve the running time of the Cheriyan-Thurimella algorithm, but it may also compute a better solution.

1 Introduction

The problem of approximating subgraphs that satisfy certain connectivity requirements has received a lot of attention (see, e.g., [12], and the survey [27]). 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 problem, 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 with many practical applications [15]. This problem is NP-complete for $k \geq 2$ for undirected graphs, and for $k \geq 1$ for directed graphs [14]. Recently, the more general problem of approximating minimum-cost subgraphs that satisfy certain connectivity requirements has also received a lot of attention. See, e.g., [12], and the survey [27].

Here we consider the problem of approximating the smallest 2VCSS. The current best approximation ratio for this problem is 3/2, achieved by the algorithm by Cheriyan and Thurimella [7], which runs in $O(m^2)$ time. Our first contribution is to provide two linear-time algorithms that attain 3- and 2-approximate solutions, based on linear-time tests for 2-vertex-connectivity in digraphs [16, 25]. The 3-approximation algorithm

*This work is a rewritten combination of two conference papers, "Approximating the Smallest 2-Vertex Connected Spanning Subgraph of a Directed Graph," *Proc. 19th Annual European Symposium on Algorithms,* pp. 13–24, 2011, and "Approximating the Smallest 2-Vertex-Connected Spanning Subgraph via Low-High Orders," *Proc. 16th International Symposium on Experimental Algorithms,* pp. 9:1-9:16, 2017.

[1]University of Ioannina, Greece. loukas@cs.uoi.gr

[2]University of Rome Tor Vergata, Italy. firstname.lastname@uniroma2.it

also uses the concept of divergent spanning trees. The 2-approximation algorithm refines this approach by utilizing low-high orders [20]. We review these definitions in Section 2. Then we show how to combine these algorithms with the algorithm of Cheriyan and Thurimella in order to obtain a 3/2-approximation in $O(m\sqrt{n} + n^2)$ time.

To assess the practical value, we conducted a thorough experimental evaluation of all the above algorithms on a variety of input graphs. Our experimental results show that our new algorithms perform very well in practice. In particular, in our experiments the new 3/2-approximation algorithms kept essentially the same approximation ratio as the algorithm of Cheriyan and Thurimella, but it was significantly faster.

We observe that recent work [17, 19] 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 that we use in our algorithms. A *flow graph* $G = (V, E, s)$ is a directed graph (digraph) with a distinguished start vertex $s \in V$ 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. The dominator tree is a central tool in program optimization and code generation [8], and it has applications in other diverse areas [22]. The dominator tree of a flow graph can be computed in linear time [2, 5].

We denote the vertex set and the edge set of a graph G by $V(G)$ and $E(G)$, respectively. If G is a digraph, then we denote by $E^r(G)$ the edges of G with their orientation reversed. We also let G^r be the digraph with $V(G^r) = V(G)$ and $E(G^r) = E^r(G)$.

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 $v \in V$ in T. If v is an ancestor of w, $T[v, w]$ is the path in T from v to w. 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. 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.

Testing 2-vertex-connectivity. Consider the flow graph $G = (V, E, s)$ of a strongly connected graph, where s is an arbitrarily selected start vertex. From [25] we have that a vertex $x \neq s$ is a strong articulation point of G if and only if x is not a leaf in the dominator tree of G or in the dominator tree of G^r. Hence, if G is 2-vertex-connected, both these dominator trees are flat. Moreover, $G \setminus s$ must be strongly connected.

Property 2.1. *Let G be a strongly connected digraph, and let s be any vertex of G. G is 2-vertex-connected if and only if the flow graphs G and G^r with start vertex s have flat dominator tree, and $G \setminus s$ is strongly connected.*

Low-high orders. A *low-high order δ of G* [20] is a preorder of the dominator tree D such for all vertices $v \neq s$, $(d(v), v) \in E$ 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. See Figure 1. Every flow graph G has a low-high order, computable in linear-time [20]. Low-high orders provide a correctness certificate for dominator trees that is straightforward to verify [34], and also have applications in path-determination problems [20, 33] and in fault-tolerant network design [3, 4, 21].

Fix a low-high order δ of G and let $E' \subseteq E$ be a subset of edges. We say that E' *satisfies* δ if for any vertex $v \neq s$ we have $(d(v), v) \in E'$ 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 $D(s)$.

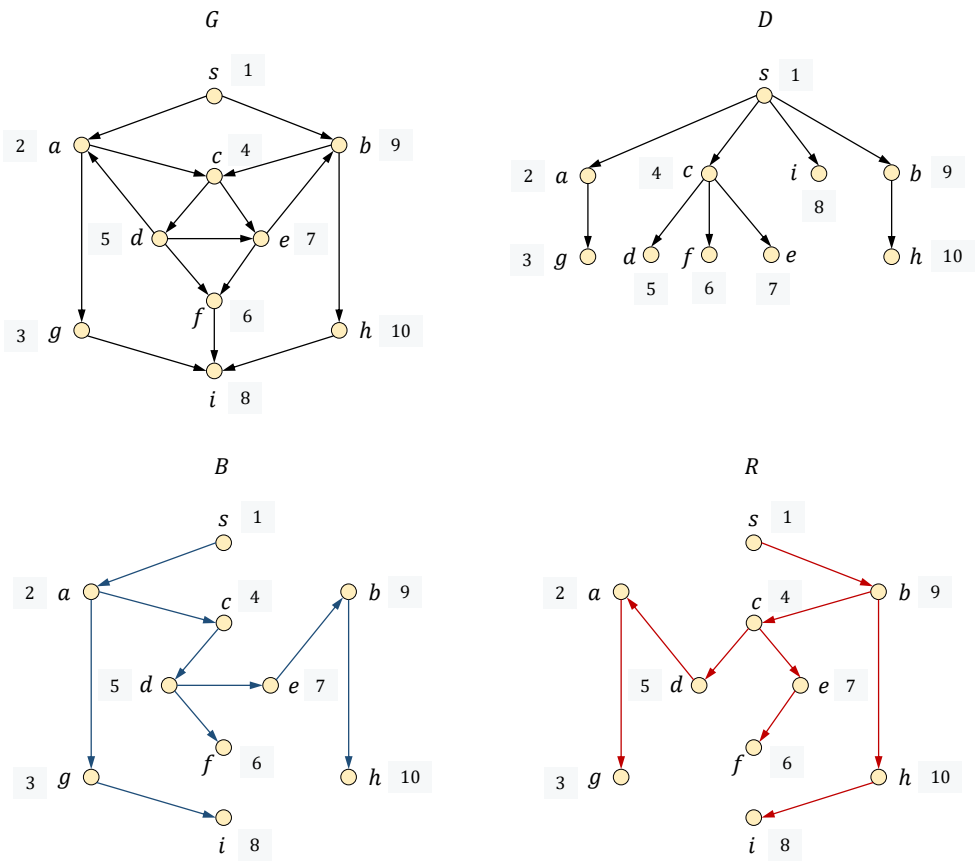


Figure 1: A flow graph G, its dominator tree D, and two divergent spanning trees B and R. The numbers correspond to a preorder numbering of D that is a low-high order of G.

Property 2.2. *Let δ be a low-high order of flow graph $G = (V, E)$ with start vertex s and let $E' \subseteq E$ be a subset of edges that satisfies δ. Then, the flow graph $G' = (V, E')$ has the same dominator tree as G.*

Divergent spanning trees. A notion closely related to low-high orders is that of divergent spanning trees [20]. 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. 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. Every flow graph has a pair of strongly divergent spanning trees. Given a low-high order of G, it is straightforward to compute two strongly divergent spanning trees of G in $O(m)$ time [20].

Matchings. For an undirected graph G and a subset of edges $M \subseteq E(G)$, we let $\deg_M(v)$ denote the degree of vertex v in M, i.e., the number of edges in M adjacent to v. The subset M is a *matching* if for all vertices v, $\deg_M(v) \leq 1$. A vertex v is *free* (with respect to M) if $\deg_M(v) = 0$. If for all vertices v, $\deg_M(v) \geq k$, then we call M a $(\geq k)$*-matching*. Given a function $b : V \mapsto \mathbf{Z}$ the *b-matching* problem is to compute a maximum-size subgraph $G' = (V, E(G'))$ of G such that $\deg_{E(G')}(v) \leq b(v)$ for all v. This problem is also referred to as the *degree-constrained subgraph problem* [13]. For a directed graph G and a subset of edges $M \subseteq E(G)$, we let $\deg^-_M(v)$ denote the in-degree of vertex v in M, i.e., the number of edges in M entering v, and let $\deg^+_M(v)$ denote the out-degree of vertex v in M, i.e., the number of edges in M leaving v. If for all vertices v, $\deg^+_M(v) \geq k$ and $\deg^-_M(v) \geq k$, then we call M a $(\geq k)$*-matching*.

3 The Cheriyan-Thurimella Algorithm

In this section we review the algorithm of Cheriyan and Thurimella [7] that gives a 3/2-approximation of the smallest 2VCSS. This algorithm is actually more general, as it computes a $(1+1/k)$-approximation of the smallest kVCSS. First, we mention a straightforward algorithm to compute a minimal kVCSS, based on edge filtering.

Let H be a kVCSS of a k-vertex-connected digraph G. We say that H is *minimal* if $H \setminus e$ is not k-vertex-connected for any edge e. Results of Edmonds [11] and Mader [30] imply that every minimal kVCSS has at most $2kn$ edges [7]. This fact implies that the following simple heuristic, that we refer to as MINIMAL, guarantees a 2-approximation of the smallest kVCSS. The algorithm processes the edges of G in an arbitrary order, and while doing so maintains a current subgraph $H = (V, E_H)$ of G. Initially $E_H \leftarrow E$. When an edge (x, y) it is processed, MINIMAL tests if H contains at least $k+1$ vertex-disjoint paths from x to y (including the edge (x, y)). If this is the case then the algorithm sets $E_H \leftarrow E_H \setminus (x, y)$. At the end of this procedure H is a minimal kVCSS of G. Testing if a digraph G has $k+1$ vertex-disjoint paths from x to y can be carried out efficiently, for constant k, by computing $k+1$ edge-disjoint paths in an auxiliary graph $\widehat{G}$ using a flow-augmenting algorithm [1]. The vertex set $V(\widehat{G})$ contains a pair of vertices v^- and v^+ for each vertex $v \in V(G)$. The edge set $E(\widehat{G})$ contains the edge (v^-, v^+) corresponding to all $v \in V(G)$. Also, for each edge (v, w) in $E(G)$, $E(\widehat{G})$ contains the edge (v^+, w^-). A single flow-augmentation step in $\widehat{G}$ takes $O(m)$ time, therefore the $k+1$ edge-disjoint paths from x to y paths are constructed in $O(km)$, and the overall running time of the algorithm is $O(km^2)$.

Cheriyan and Thurimella [7] proposed an elegant algorithm that achieves an $(1+1/k)$-approximation of the smallest kVCSS. The algorithm consists of the following two phases:

Phase 1 (matching). Given the input digraph $G = (V, E)$, this phase computes a minimum $(\geq k-1)$-matching M of G. This is transformed to a b-matching problem on a bipartite graph B associated with G. The bipartite graph is constructed as follows. For each vertex $v \in V$ there is a pair of vertices v^- and v^+ in $V(B)$. For each edge (v, w) in $E(G)$ there is an edge $\{v^+, w^-\}$ in $E(B)$. The problem of computing M in G is equivalent to computing a minimum $(\geq k-1)$-matching M_B in B. This, in turn, is equivalent to computing a maximum b-matching M'_B in B, where $b(v) = \deg_{E(B)}(v) - (k-1)$, since $M_B = E(B) \setminus M'_B$. A b-matching problem on a graph with n vertices and m edges can be solved in $O(\sqrt{m\alpha(m,m)\log m}\, m \log m)$ time [13].

Phase 2 (filtering). The second phase runs the MINIMAL algorithm but only for the arcs in $E \setminus M$. More specifically, the algorithm maintains a current graph $H(V, E_H)$. Initially we set $E_H \leftarrow E$, and at the end of this phase $E_H = M \cup F$, where $F \subseteq E \setminus M$ is a minimal subset of edges such that H is k-vertex connected. Hence, phase 2 takes $O(k|E|^2)$ time, which is also the asymptotic running time of the whole algorithm.

3.1 Simplification for $k = 2$.

Although the Cheriyan-Thurimella algorithm is conceptually simple (but its analysis is intricate), it is challenging to provide an efficient implementation, especially for phase 1. Fortunately, in the case $k = 2$ we can implement phase 1 as follows. First we compute a maximum matching M in B. Then we augment this matching to a set M_2 by adding an edge incident to each free vertex. Next we show that M_2 is indeed a minimum (≥ 1)-matching in B. (This fact is mentioned in [7], but for completeness we give a proof.)

Lemma 3.1. *M_2 is a minimum (≥ 1)-matching in B.*

Proof. Clearly, M_2 is a (≥ 1)-matching in B, so it remains to show that it is minimum. Let M' be a minimum (≥ 1)-matching in B. Let $X = \{x \in V(B) \mid \deg_{M'}(x) > 1\}$. If $X = \emptyset$ then $\deg_{M'}(x) = 1$ for all $x \in V(B)$. In this case M' is a perfect matching, hence $|M'| = |M_2|$.

Consider now $X \neq \emptyset$. Let x be any vertex in X. Then, for any edge $\{x, y\}$ in M', $\deg_{M'}(y) = 1$. Otherwise, $M' - \{x, y\}$ is a (≥ 1)-matching in B which contradicts the fact that M' is minimum. Therefore, there is no edge $\{x, y\} \in M'$ such that both x and y are in X. Let N be a subset of M' that is left after removing $\deg_{M'}(x) - 1$ edges for each $x \in X$. Suppose that ℓ edges are removed from M' to form N. Then

Algorithm 1: DST(G)

Input: 2-vertex-connected digraph $G = (V, E)$
Output: 3-approximation of a smallest 2-vertex-connected spanning subgraph $H = (V, E_H)$ of G

1 Set $E_H \leftarrow \emptyset$.
2 Choose an arbitrary vertex s of G.
3 Compute two divergent spanning trees B and R of flow graph G with start vertex s.
4 Compute two divergent spanning trees B' and R' of flow graph G^r with start vertex s.
5 Compute a strongly connected spanning subgraph $H = (V \setminus s, E_H)$ of $G \setminus s$.
6 Set $E_H \leftarrow E_H \cup E(B) \cup E(R) \cup E^r(B') \cup E^r(R')$.
7 **return** $H = (V, E_H)$

$|N| = |M'| - \ell$ and B has ℓ free vertices with respect to N. We show that N is a maximum matching in B. Suppose, for contradiction, that it is not. Let M be a maximum matching. Then, for some $\ell' \geq 1$ we have $|M| = |N| + \ell' = |M'| + (\ell' - \ell)$. Next note that there are $\ell - 2\ell'$ free vertices with respect to M. Therefore $|M_2| \leq |M| + (\ell - 2\ell') = |M'| + \ell' - \ell + \ell - 2\ell' = |M'| - \ell' < |M'|$, a contradiction. So $|M| = |N|$ which implies $|M_2| = |M'|$. □

We refer to the above version of the Cheriyan-Thurimella algorithm, for $k = 2$, as CT.

Implementation details. For the first phase of CT we used an implementation of the push-relabel maximum-flow algorithm of Goldberg and Tarjan [23] from [9]. This implementation does not use a dynamic tree data structure [31], which means that the worst-case bound for the first phase of these algorithms is $O(n^3)$. However, as we confirmed experimentally, the push-relabel algorithm runs very fast in practice.

4 Linear Approximation Algorithms

In this section we present our linear-time algorithms that compute an approximate smallest 2VCSS of a 2-vertex-connected digraph G.

4.1 Linear-time 3-approximation

Our first algorithm, referred to as DST, applies Property 2.1 and a linear-time construction of divergent spanning trees [20]. (See Algorithm 1.)

Theorem 4.1. *Algorithm DST computes a 3-approximation of the smallest 2VCSS in linear time.*

Proof. Let H be the subgraph computed by DST. By construction, H and H^r have flat dominator trees, and $H \setminus s$ is strongly connected. Thus, H is 2-vertex-connected by Property 2.1. Now we establish that DST runs in linear time. For the computation of the divergent spanning trees in lines 3 and 4 we can use a linear-time algorithm from [20]. In line 5 we can compute an approximate smallest strongly connected spanning subgraph of $G \setminus s$ [26]. For this, we can use the linear-time algorithm of Zhao et al. [35], which selects at most $2(n-1)$ edges.

Finally, we show that the algorithm selects at most $6n$ edges in total. Then, the approximation ratio of 3 follows from the fact that any 2VCSS of G must contain at least $2n$ edges. Since H is formed by 4 spanning trees on n vertices and at most $2(n-1)$ additional edges (to guarantee that $G \setminus s$ is strongly connected), we have $|E(H)| \leq 4(n-1) + 2(n-2) < 6n$. □

Practical implementation. In line 3 (and similarly in line 4), we do not actually need to compute two divergent spanning trees of G, as it suffices to find the edges that are contained in two such spanning trees. This turns out to be a simpler task that we can accomplish with the help of *semi-dominators*, introduced by Lengauer and Tarjan [28] in their fast algorithm for computing dominators in flow graphs. Let T be a depth-first search tree of G, rooted at s. We denote by $t(v)$ the parent of any vertex $v \neq s$ in T. We assign to each vertex a preorder number with respect to T and identify the vertices by their preorder numbers. Then, $u < v$

Algorithm 2: LH(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 \ s, E_H) of G \ s.
3  Set H ← (V, E_H).
4  Compute a low-high order δ of flow graph G with start vertex s.
5  foreach vertex v ≠ s do
6      if there are two edges (u, v) and (w, v) in E_H such that u <_δ v and v <_δ w then
7          do nothing
8      end
9      else if there is no edge (u, v) ∈ E_H such that u <_δ v then
10         find an edge e = (u, v) ∈ E with u <_δ v
11         set E_H ← E_H ∪ {e}
12     end
13     else if there is no edge (w, v) ∈ E_H such that v <_δ w then
14         find an edge e = (w, v) ∈ E with v <_δ w or w = s
15         set E_H ← E_H ∪ {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)
```

means that u was visited before v during the depth-first search. A path $P = (u = v_0, v_1, \ldots, v_{k-1}, v_k = v)$ is a *semi-dominator path* if $v < v_i$ for $1 \leq i \leq k-1$. The *semi-dominator* of vertex v, denoted by $sdom(v)$, is defined as the minimum vertex u such that there is a semi-dominator path from u to v. From the properties of depth-first search it follows that, for every $v \neq s$, $sdom(v)$ is a proper ancestor of v in T [28]. For any vertex $v \neq s$, we define $p(v)$ to be the last vertex before v in a semi-dominator path from $sdom(v)$ to v. Such vertices can be found easily during the computation of semi-dominators. Therefore, by [5], we can compute $p(v)$ for all $v \neq s$ in linear time. If follows that the spanning subgraph of G consisting of the edges $(t(v), v)$ and $(p(v), v)$, for all $v \neq s$, has the same dominator tree as G and thus contains two divergent spanning trees of G.

In our implementation we also incorporate the following optimization, in order to reduce the number of edges of the computed 2VCSS. Let G' be the subgraph of G that is induced by the edges in $E(B) \cup E(R) \cup E^r(B^r) \cup E^r(R^r)$. Before we execute line 5, we use Property 2.1 to test if G' is already 2-vertex-connected. If this is the case, then we omit line 5 and return $H = G'$. Also, note that if G' is 2-vertex-connected, then we compute a 2-approximation of the smallest 2VCSS.

4.2 Linear-time 2-approximation

Here we present a more sophisticated algorithm that refines our previous approach. Our new algorithm, which we call LH, exploits the properties of low-high orders as follows. (See Algorithm 2.) 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. [35] (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 LH is 2-vertex-connected.

Lemma 4.2. *Algorithm LH computes a 2VCSS of G.*

Proof. We need to show that the computed subgraph H satisfies Property 2.1. By line 2 we have that $H \setminus s$ is strongly connected, so it remains to show that H has flat dominator tree. The same argument applies

for H^r, thus completing the proof. Let δ be the low-high order δ of G, computed in line 3. We argue that after the execution of the for loop in lines 5–17, δ is also a low-high order for all vertices in H. Consider an arbitrary vertex $v \neq s$. Let (x, v) be an edge entering v in the 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 $y <_\delta v$. 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 an edge $(y, v) \in E$ such that $y >_\delta v$ or $y = s$. Notice that in either case $y \neq x$. So, again, 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 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. By [20], this means that H contains two divergent spanning trees B and R of G. Since G has flat dominator tree, 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 flat. □

We remark that the construction of H in algorithm LH guarantees that s will have in-degree and out-degree at least 2 in H. (This fact is implicit in the proof of Lemma 4.2.) Indeed, H will contain the edges from s to the vertices in $V \setminus s$ with minimum and maximum order with respect to a low-high order of G, and the edges entering s from the vertices in $V \setminus s$ with minimum and maximum order with respect to a low-high order of G^r.

Theorem 4.3. *Algorithm LH computes a 2-approximation of the smallest 2VCSS in linear time.*

Proof. We establish the approximation ratio of LH 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. As in algorithm DST, in line 2 we can compute an approximate smallest strongly connected spanning subgraph of $G \setminus s$ by using the linear-time algorithm of Zhao et al. [35], 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 graph $H \setminus s$ 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, E_H contains at most $4(n-1)$ at the end of the execution. □

5 Hybrid Approximation Algorithms

In this section we consider how to combine our two linear algorithms of Section 4 with the algorithm of Cheriyan and Thurimella, so that we both maintain the 3/2 approximation guarantee of the latter and decrease its running time.

First, we observe that we can immediately combine DST (or LH) with MINIMAL, as follows. We begin by computing a sparse 2VCSS of G, with $O(n)$ edges, by setting $H \leftarrow$ DST(G). Then we return the 2VCSS computed by MINIMAL(H). Since we still compute a minimal 2VCSS, we get a 2-approximation algorithm, with its running time improved from $O(m^2)$ to $O(n^2)$. Note that the above idea does not work for CT, since DST (or LH) may filter out the wrong edges, i.e., the edges of a minimum (≥ 1)-matching, and therefore, the final 2VCSS may not give a 3/2-approximation. To fix this problem we propose the following combination of DST and CT, referred to as DST-CT, which runs DST between the two phases of CT.

5.1 Algorithm DST-CT

Following the computation of a minimum (≥ 1)-matching M, we run DST with the initial digraph G as input. This returns a 2VCSS G' of G. The input to the second phase of CT is the subgraph of G induced by the set of edges $E(G') \cup M$. See Algorithm 3.

Theorem 5.1. *Algorithm DST-CT computes a 3/2-approximation of the smallest 2VCSS in $O(m\sqrt{n} + n^2)$ time.*

Proof. We prove the 3/2 approximation ratio of DST-CT by showing that a specific execution of CT produces the same output subgraph. Let $H = (V, E_H)$ be the current subgraph of DST-CT computed in line 3, just

Algorithm 3: DST-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 minimum (≥ 1)-matching M of G.
2 Compute H ← DST(G).
3 Set E_H ← E_H ∪ M.
4 foreach edge (x, y) of E_H \ M do
5     if there are two vertex-disjoint paths from x to y in H \ (x, y) then
6         Set E_H ← E_H \ (x, y).
7     end
8 end
9 return H = (V, E_H)
```

before the execution of the filtering phase. After computing a minimum (≥ 1)-matching M of the input digraph G, the Cheriyan-Thurimella algorithm can process the edges in $E(G) \setminus M$ in an arbitrary order. That is, the approximation guarantee 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 in $E' = E(G) \setminus E(H)$ first. Notice that for each edge $(x, y) \in E'$, H contains two vertex-disjoint paths from x to y. Thus, all the edges in E' will be removed from the current 2VCSS that is maintained during the second phase, so these edges will not be included in the subgraph computed by CT. So, if we fix the order in which the edges in $E(G)$ are processed, the filtering phase in both CT and DST-CT will remove exactly the same redundant edges. Therefore, the approximation guarantee of the Cheriyan-Thurimella algorithm is preserved.

Regarding the running time of DST-CT, by Lemma 3.1 we have that a minimum (≥ 1)-matching in line 1 can be computed in $O(m\sqrt{n})$ time using the Hopcroft-Karp maximum bipartite matching algorithm [24]. Also, by Section 4, the computation of H in line 2 takes linear time. Then, we are left with a 2VCSS with $O(n)$ edges, so the filtering phase of the algorithm runs in $O(n^2)$ time. □

We can apply the same idea in order to combine LH with CT. Here, however, we can also take advantage of the fact that the edges in the (≥ 1)-matching of G can be used when we add edges to our subgraph H until it satisfies a given low-high order of G.

5.2 Algorithm LH-CT

Let $G = (V, E)$ be the input 2-vertex-connected digraph. Algorithm LH-CT (whose pseudocode is given in Algorithm 4) works as follows. First, it computes a minimum (≥ 1)-matching M as in 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 $C_1, \ldots, C_k$ 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. [35] 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 2VCSS H of G using as many edges from Z and M as possible, as in LH. Finally, we run the filtering phase of CT for the edges in H.

Theorem 5.2. *Algorithm LH-CT computes a 3/2-approximation of the smallest 2VCSS in $O(m\sqrt{n} + n^2)$ time.*

Proof. First, we note that the spanning subgraph computed by algorithm LH-CT is 2-vertex-connected since it satisfies Property 2.2. 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-CT by showing that a specific execution of CT produces the same output subgraph. We apply the analogous arguments as in the proof of Theorem 5.1. Let

Algorithm 4: LH-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 minimum (≥ 1)-matching M of G.
2 Choose an arbitrary vertex s of G as start vertex.
3 Let H 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 $C_1, \ldots, C_k$ in H.
5 Form a contracted version G' of G as follows. For each strongly connected component C_i of H, we contract all vertices in C_i into a representative vertex $u_i \in C_i$.
6 Compute a strongly connected spanning subgraph H' of G'. Let Z be the original edges of G that correspond to the edges of G' selected in H'.
7 Set $H \leftarrow (V, E_H = M \cup Z)$.
8 Compute a low-high order δ 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** $H = (V, E_H)$

S be the set of edges of H' (i.e., the edges of H just after the execution of lines 1–22). We can assume that CT processes the edges of $E' = E \setminus S$ first. Since H' is a 2VCSS of G, for each $(x, y) \in E \setminus S$, H' contains two vertex-disjoint paths from x to y. Hence, every edge in E' 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-CT will remove exactly the same redundant edges.

Finally, we consider the running time of LH-CT. Line 1 takes $O(m\sqrt{n})$ time [24], and lines 2–5 take $O(m)$ time [32]. In line 6, we can compute a SCSS of $\breve{G}$ in $O(m)$ time [35], and in line 8 we can compute a low-high order of G in $O(m)$ time [20]. Finally, the loops in lines 9–2 and 23–27 take $O(m)$ and $O(n^2)$ time, respectively. □

Dataset	#nodes (n)	#edges (m)	File size	Type
Rome99	3353	8859	100KB	road network
P2p-Gnutella25	5153	17695	203KB	peer2peer
P2p-Gnutella31	14149	50916	621KB	peer2peer
Web-NotreDame	53968	296228	3,9MB	web graph
Soc-Epinions1	32223	443506	5,3MB	social network
USA-road-BAY	321270	800172	12MB	road network
Amazon301	241761	1131217	16MB	prod. co-purchase
WikiTalk	111881	1477893	18MB	social network
Web-Stanford	150532	1576314	22MB	web graph
Amazon601	395234	3301092	49MB	prod. co-purchase
Web-Google	434818	3419124	50MB	web graph

Table 1: Real-world graphs used in the experiments. From each original graph, we extracted its Largest Strongly Connected Component. The number of vertices n and of edges m refer to each such graph. The graphs are sorted by file size of their Largest SCC.

6 Empirical Analysis

Here we report the results of the experiments that we conducted. We implemented all our algorithms in `C++` without the use of any external graph library. We compiled our codes with `g++ v.4.8.4` with full optimization (flag `-O3`). The experiments were conducted on a 64-bit GNU/Linux machine running Ubuntu 14.04LTS, with an 3696MHz Intel i74790 octa-core processor, 20GB of RAM, 16MB of L3 cache, and each core has a 2MB private L2 cache. All experiments were executed on a single core without using any parallelization. We report CPU times measured with the `getrusage` function, averaged over ten different runs.

For our experimental study, we used a collection of real-world graphs, shown in Table 6, from [29] and [10], from which we constructed 2-vertex-connected graphs as follows. We extracted the largest strongly connected component (SCC) from each real-world graph, and computed its largest maximal 2-vertex-connected subgraph (Max2VCS) with the algorithm of Chechik et al. [6], using the implementation of [18]. We also constructed a second type of instances, where we make the largest SCC of each real-world graph 2-vertex-connected by adding a bidirectional Hamiltonian cycle as follows. We take a random permutation of the vertices of the input digraph, say $v_0, v_1, \ldots, v_{n1}$, and adding the edges (v_i, v_{i+1}) and (v_{i+1}, v_i), where the addition is computed $\mod n$. Note that a bidirectional Hamiltonian cycle is by itself a minimum 2VCSS with exactly $2n$ edges, so in this case it is easy to assess how close to optimal are the computed 2VCSS.

The characteristics of the graphs that have been created are summarized in Table 2 and we refer to them with the *-suffix*: the first type of instances are indicated as Max2VCS, while the second type are indicated as Hamilton. We measure the quality of the produced solution $H = (V, E_H)$ of each algorithm by calculating the relative distance from the naive theoretical lower bound $2|V|$, i.e., $\frac{|E_H|-2|V|}{2|V|} \times 100\%$. We refer to this relative distance as the *quality ratio* . In our experimental study, we evaluate the performance of six algorithms, outlined in Table 3. Next, we report the experimental results for each type of instances.

Graph	**Largest SCC**		Suffix			
			Max2VCS		**Hamilton**	
	n	**m**	**n**	**m**	**n**	**m**
Rome99 - suffix	3353	8859	2249.0	6467	3353	15559
P2p-Gnutella25 - suffix	5153	17695	-	-	5153	27992
P2p-Gnutella31 - suffix	14149	50916	-	-	14149	79205
Web-NotreDame - suffix	53968	296228	1462	7279	53968	404154
Soc-Epinions - suffix	32223	443506	17117	395183	32223	507924
USA-road-BAY - suffix	321270	800172	211590	568546	321270	1437366
Amazon301 - suffix	241761	1131217	55414	241663	241761	1614733
WikiTalk - suffix	111881	1477893	49430	1254898	111881	170163
Web-Stanford - suffix	150532	1576314	10893	162295	150532	1877356
Amazon601 - suffix	395234	3301092	276049	2461072	395234	4091547
Web-Google - suffix	434818	3419124	77480	840829	434818	4288743

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

Algorithm	**Complexity**	**Approximation**	**Technique**	**Reference**
DST	$O(n)$	3	Divergent Spanning Trees	This paper
LH	$O(n)$	2	Low-High Orders	This paper
MINIMAL	$O(n^2)$	2	2 Vertex-Disjoint Paths	[7]
CT	$O(m\sqrt{n}+m^2)$	3/2	Matchings, 2 Vertex-Disjoint Paths	[7]
DST-CT	$O(m\sqrt{n}+n^2)$	3/2	Divergent Spanning Trees, Matchings, 2 Vertex-Disjoint Paths	This paper
LH-CT	$O(m\sqrt{n}+n^2)$	3/2	Low High Orders, Matchings, 2 Vertex-Disjoint Paths	This paper

Table 3: An overview of the algorithms considered in our experimental study. The bounds refer to a digraph with n vertices and m edges.

Graph	Quality Ratio %					
	DST	**LH**	**CT**	**MINIMAL**	**DST-CT**	**LH-CT**
Rome99-Max2VCS	26.99	19.63	**6.51**	6.76	7.69	7.83
Web-NotreDame-Max2VCS	44.22	36.08	**5.68**	12.14	6.81	17.75
Web-Stanford-Max2VCS	73.31	49.38	36.06	47.06	**28.60**	31.10
Amazon301-Max2VCS	39.05	25.32	11.19	14.68	**10.06**	10.83
Soc-Epinions-Max2VCS	75.66	49.78	20.00	26.85	**20.03**	22.35
USA-BAY-Max2VCS	21.12	15.77	7.99	**6.94**	8.44	8.68
Web-Google-Max2VCS	67.76	45.18	31.16	44.48	**24.02**	26.54
WikiTalk-Max2VCS	79.27	58.19	37.34	45.36	**37.88**	40.84
Amazon601-Max2VCS	55.84	32.83	15.22	29.80	**9.13**	11.27

Table 4: Solution qualities for the Max2VCS instances.

Graph	Running Time					
	DST	**LH**	**CT**	**MINIMAL**	**DST-CT**	**LH-CT**
Rome99-Max2VCS	0.004	0.004	0.064	0.072	0.068	0.048
Web-NotreDame-Max2VCS	0.004	0.004	0.064	0.144	0.052	0.028
Web-Stanford-Max2VCS	0.008	0.012	6.728	8.056	3.760	2.604
Amazon301-Max2VCS	0.036	0.048	99.516	156.232	81.360	59.400
Soc-Epinions-Max2VCS	0.028	0.036	50.968	59.484	20.080	11.544
USA-BAY-Max2VCS	0.120	0.128	614.512	744.040	822.536	661.288
Web-Google-Max2VCS	0.064	0.084	267.212	373.800	154.148	98.952
WikiTalk-Max2VCS	0.104	0.124	555.708	745.788	169.124	88.448
Amazon601-Max2VCS	0.496	0.576	6347.912	20214.252	4498.232	3862.028

Table 5: Running times in seconds for the Max2VCS instances.

Max2VCS instances. In this category we compute for every graph of Table 6 its largest 2-vertex-connected subgraph. (Characteristics of the input graphs are summarized in Table 2.)

The running times and quality measures for LH and DST are plotted in Figure 2. Figure 3 shows the corresponding plots for CT, MINIMAL DST-CT and LH-CT. (See also Tables 4 and 5.) It is easy to observe that the algorithms belong to two distinct classes, with DST and LH 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 10-50% 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 DST and LH (Figure 2). We observe that in all test instances LH was able to compute a 2VCSS with 6-25% fewer edges, which was expected due to its improved theoretical guarantee, at the price of a small overhead in the running time.

In our next experiment (Figure 3), we compare algorithms DST-CT and LH-CT which produce the best solutions overall. Observe that LH-CT is always faster, mainly due to the fact that it has to process fewer edges during the filtering phase. DST-CT on the other hand, produced better solutions in all of the test graphs, but the difference is marginal (at most 3.9% fewer edges)

Finally, we consider the performance of CT and MINIMAL. Notice that although MINIMAL, rather surprisingly, computes a better solution in one instance, its performance is rather unstable. Compared to MINIMAL, CT is much more robust and computed solutions of higher quality in all but one instance. Overall, DST-CT and CT achieved the highest solution quality, but the former is significantly faster than the latter.

Graph	Solution Quality %					
	DST	**LH**	**CT**	**MINIMAL**	**DST-CT**	**LH-CT**
Rome99-Hamilton	38.47	25.16	**6.35**	25.16	7.23	7.90
P2p-Gnutella25-Hamilton	42.23	27.08	9.83	27.08	**8.61**	9.68
P2p-Gnutella31-Hamilton	42.02	27.15	**8.23**	27.15	8.86	9.35
WikiTalk-Hamilton	41.76	32.37	**6.44**	32.37	8.50	12.44
Web-NotreDame-Hamilton	45.86	31.57	**7.11**	31.57	11.50	11.99
Soc-Epinions-Hamilton	48.26	31.85	**9.06**	31.85	9.70	11.30
USA-BAY-flat-Hamilton	38.22	25.48	**6.15**	25.48	7.40	8.14
Amazon301-Hamilton	44.58	27.67	**8.03**	27.67	8.80	9.50
Web-Standford-Hamilton	50.53	32.73	21.55	32.73	**12.95**	12.96
Amazon601-Hamilton	49.46	29.44	9.65	29.44	**9.46**	10.42
Web-Google-Hamilton	48.88	31.41	20.97	31.41	**11.39**	11.46

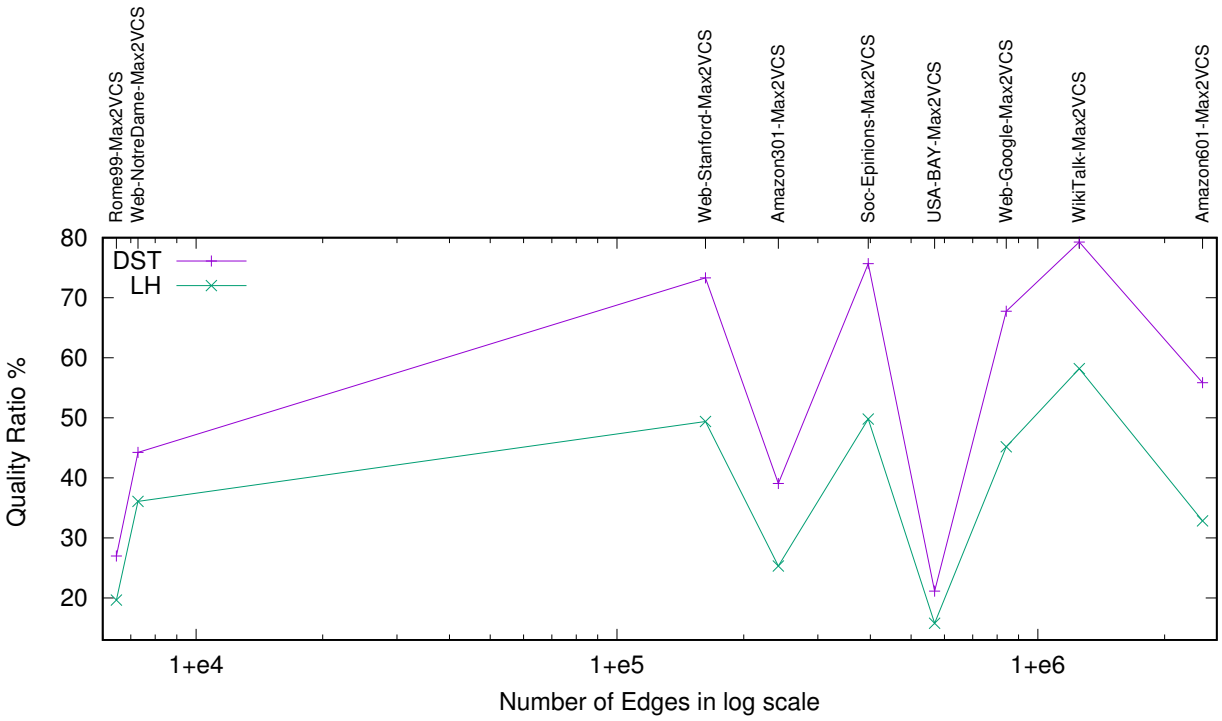
Table 6: Solution qualities for the Hamilton instances.

Graph	Running Time					
	DST	**LH**	**CT**	**MINIMAL**	**DST-CT**	**LH-CT**
Rome99-Hamilton	0.008	0.004	0.796	0.852	0.284	0.476
P2p-Gnutella25-Hamilton	0.004	0.020	2.316	2.616	0.704	1.188
P2p-Gnutella31-Hamilton	0.016	0.016	20.688	24.208	7.176	11.072
WikiTalk-Hamiltonian	0.228	0.328	2671.052	4366.740	466.640	882.376
Web-NotreDame-Hamilton	0.064	0.084	334.120	420.344	114.476	187.388
Soc-Epinions-Hamilton	0.044	0.068	165.820	227.320	41.980	68.792
USA-BAY-flat-Hamilton	0.780	0.976	31624.608	53595.184	12376.752	18840.544
Amazon302-Hamilton	0.660	0.900	19015.068	31608.632	5862.820	10186.228
Web-Standford-Hamilton	0.368	0.504	6085.084	8613.912	1378.880	2709.608
Amazon601-Hamilton	1.724	2.668	66588.004	141155.596	19175.004	32394.968
Web-Google-Hamilton	2.04	2.724	72394.252	91918.144	26947.428	40047.832

Table 7: Running times in seconds for the Hamilton instances.

Hamilton instances In Figure 4 we compare the performance of the linear-time algorithms DST and LH. (See also Tables 6 and 7.) As in the previous experiment, LH is able to provide a better solution with a very small overheard in the running time.

Figure 5 plots the running times and the solution of qualities for CT, MINIMAL,DST-CT and LH-CT. One important observation is that the solution quality of these algorithms is much better than in the Max2VCS instances. We also observe that LH-CT is always faster than any other superlinear algorithm, but produces slightly worse solutions (at most 3.8% more edges) than DST-CT. CT on the one hand produces the best solutions in 7 out of 11 instances. In two instances (Web-Standford-Hamilton, Web-Google-Hamilton), however, it computed solutions that were significantly worse compared to DST-CT and LH-CT. Notice also that CT is significantly slower than DST-CT and LH-CT. Finally, unlike the Max2VCS instances, here MINIMAL has a more stable performance but computes a significantly worse solution than any other nonlinear algorithm.



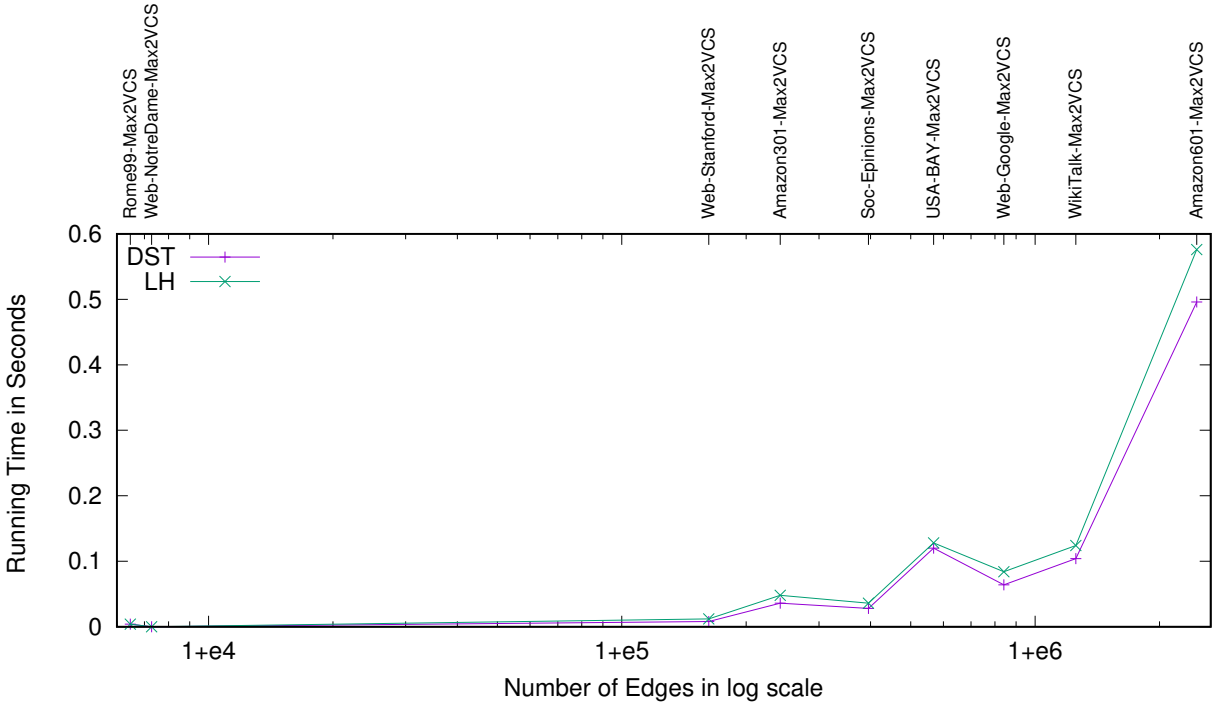
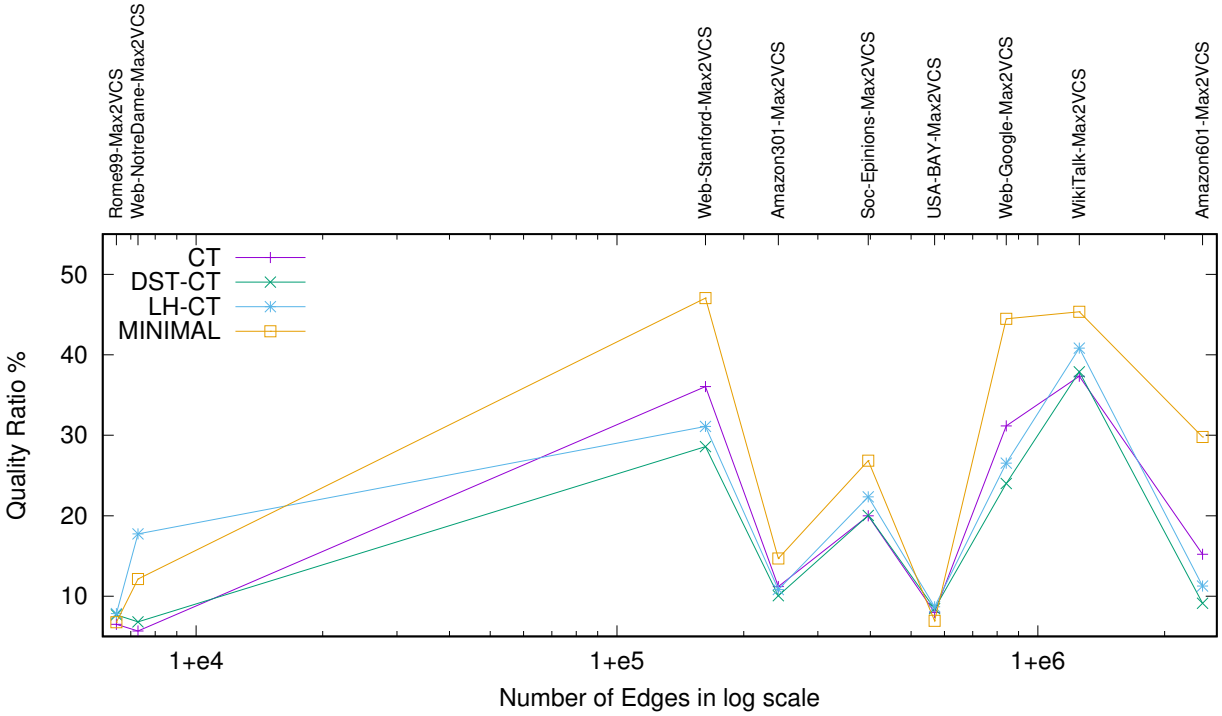


Figure 2: Performance of linear-time algorithms for the Max2VCS instances. Solution qualities (top) and running times in seconds (bottom). Running times and graph sizes (number of edges) are shown in log scale.



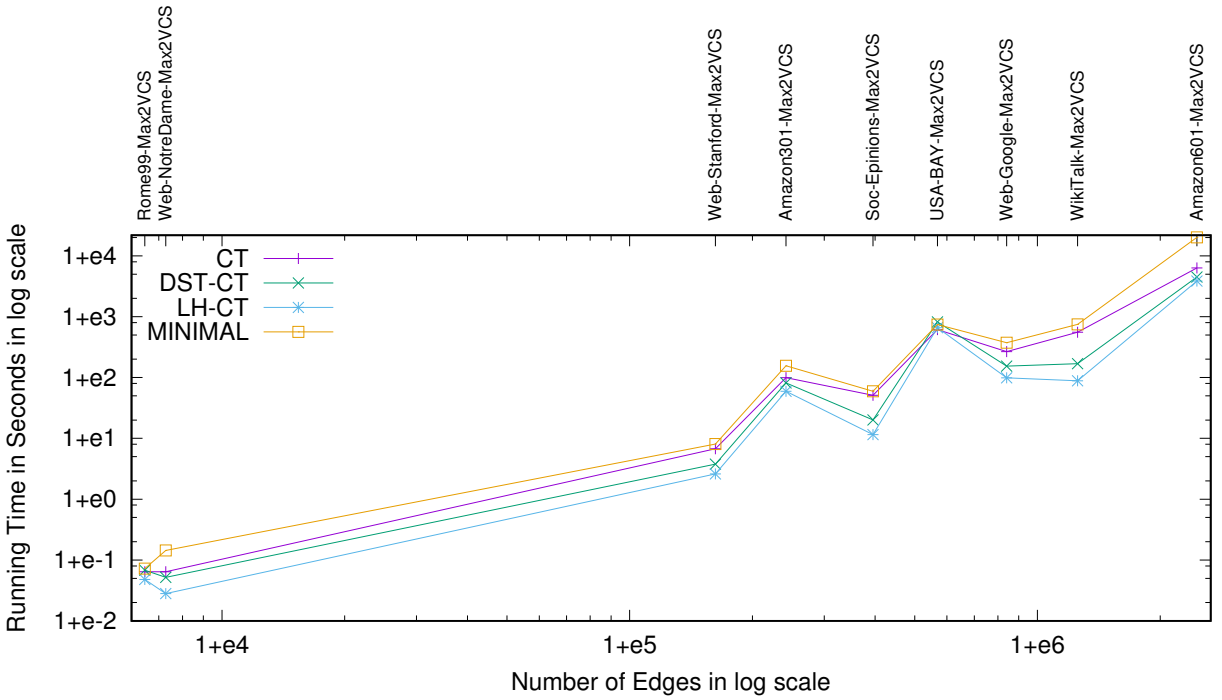
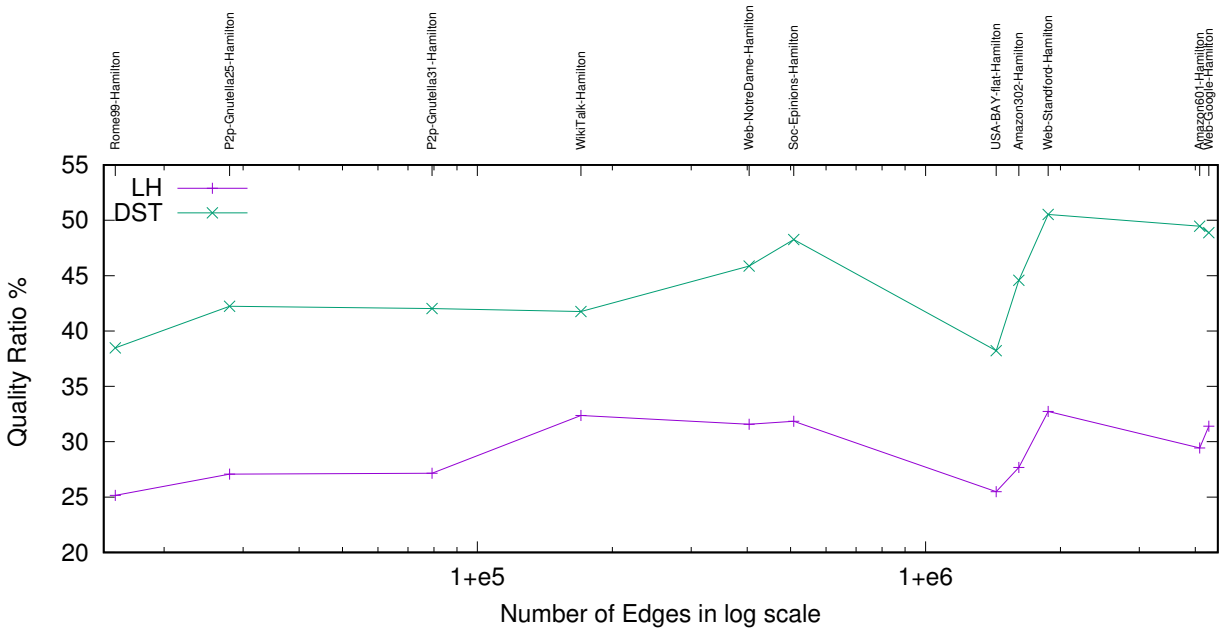


Figure 3: Performance of algorithms CT, MINIMAL, DST-CT and LH-CT for the Max2VCS instances. Solution qualities (top) and running times (bottom). (Better viewed in color.) Running times and graph sizes (number of edges) are shown in log scale.



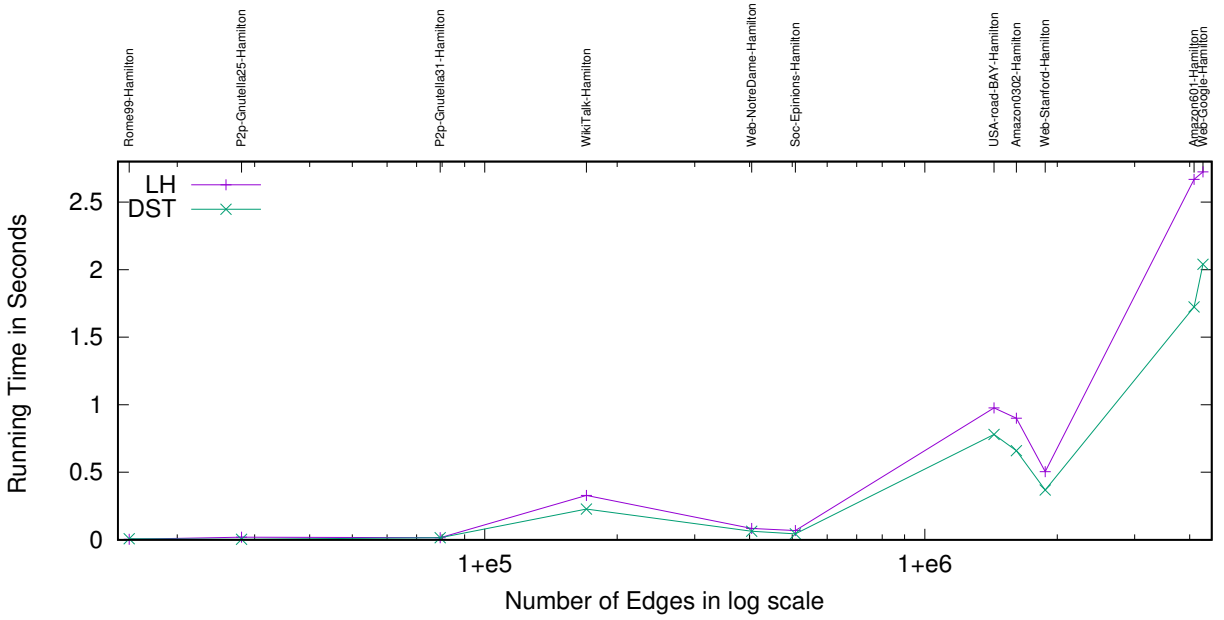
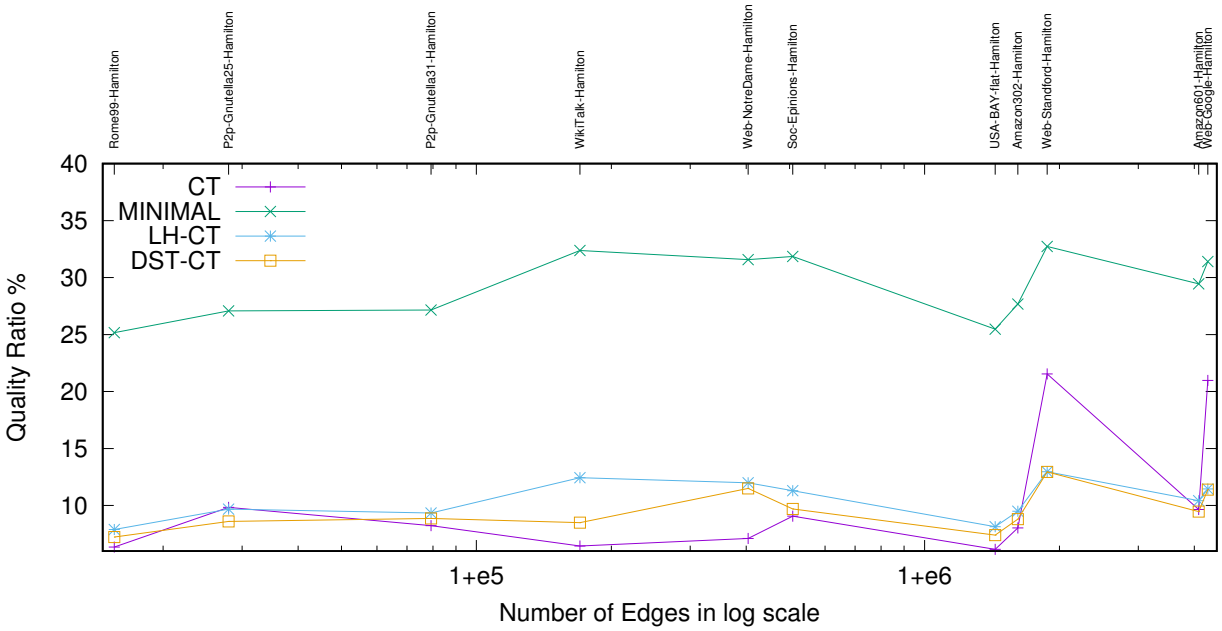


Figure 4: Performance of linear-time algorithms for the Hamilton instances. Solution qualities (top) and running time in seconds (bottom). Running times and graph sizes (number of edges) are shown in log scale.



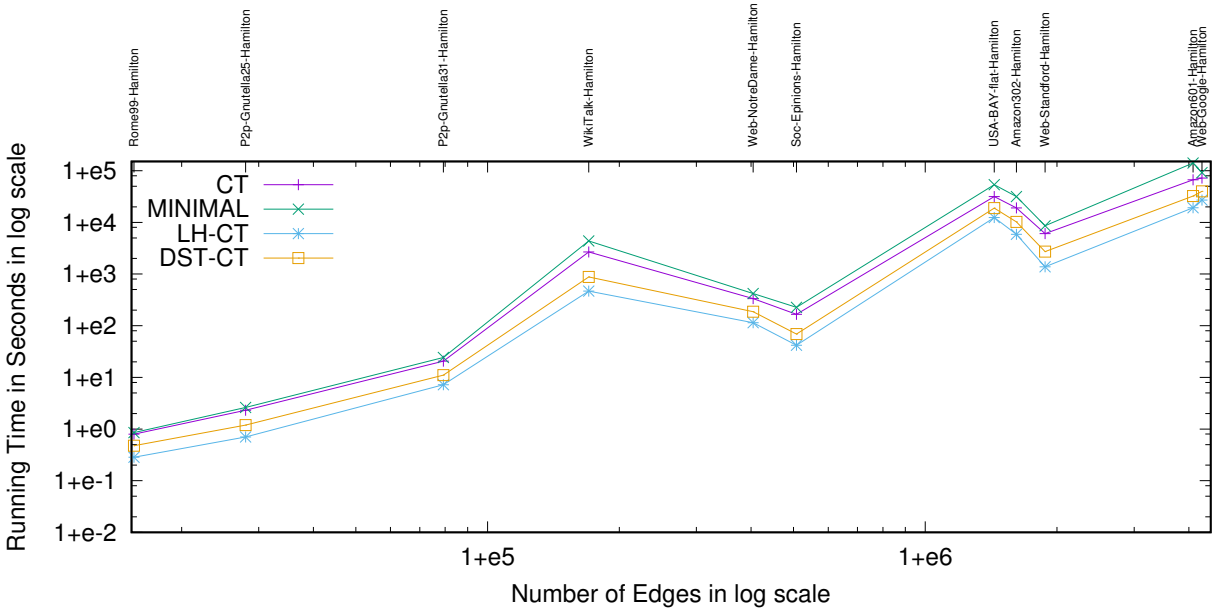


Figure 5: Performance of algorithms CT, MINIMAL, DST-CT and LH-CT for the Hamilton instances. Solution qualities (top) and running times (bottom). (Better viewed in color.) Running times and graph sizes (number of edges) are shown in log scale.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

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

[3] 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.

[4] S. Baswana, K. Choudhary, and L. Roditty. Fault tolerant reachability subgraph: Generic and optimal. In *Proc. 43rd ACM Symp. on Theory of Computing*, pages 509–518, 2016.

[5] 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.

[6] S. Chechik, T. D. Hansen, G. F. Italiano, V. Loitzenbauer, and N. Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1900–1918. SIAM, 2017.

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

[8] 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.

[9] 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.

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

[11] J. Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96, 1972.

[12] J. Fakcharoenphol and B. Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In *Proc. 40th ACM Symp. on Theory of Computing*, STOC '08, pages 153–158, New York, NY, USA, 2008. ACM.

[13] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38:815–853, 1991.

[14] 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.

[15] N. Garg, V. S. Santosh, and A. Singla. Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pages 103–111, 1993.

[16] L. Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. In *Proc. 37th Int'l. Coll. on Automata, Languages, and Programming*, pages 738–749, 2010.

[17] 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*, (SEA 2016), pages 150–166, 2016.

[18] L. Georgiadis, G. F. Italiano, A. Karanasiou, N. Parotsidis, and N. Paudel. Computing 2-connected components and maximal 2-connected subgraphs in directed graphs: An experimental study. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 169–183. SIAM, 2018.

[19] L. Georgiadis, G. F. Italiano, C. Papadopoulos, and N. Parotsidis. Approximating the smallest spanning subgraph for 2-edge-connectivity in directed graphs. In *Proc. 23rd European Symposium on Algorithms*, pages 582–594, 2015.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] W. Mader. Minimal n-fach zusammenhängende digraphen. *Journal of Combinatorial Theory, Series B*, 38(2):102–117, 1985.

[31] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

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

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

[34] 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.

[35] 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.