Sparse Certificates for 2-Connectivity in Directed Graphs*

Loukas Georgiadis† Giuseppe F. Italiano‡ Aikaterini Karanasiou‡
Charis Papadopoulos† Nikos Parotsidis‡

Abstract

Motivated by the emergence of large-scale networks in today's applications, we show how to compute efficiently smaller subgraphs that maintain some properties of an input graph. In particular, let G be a strongly connected directed graph. We consider the problem of computing the smallest strongly connected spanning subgraph of G that maintains certain connectivity relations of G. Specifically, for 2-edge-connectivity, we consider how to maintain the maximal 2-edge-connected subgraphs (2ECS) or the 2-edge-connected components (2ECC) of G, or both the maximal 2-edge-connected subgraphs and the 2-edge-connected components (2EC). Similarly, for 2-vertex-connectivity, we consider how to maintain the maximal 2-vertex-connected subgraphs (2VCS) or the 2-vertex-connected components (2VCC) of G, or both the maximal 2-vertex-connected subgraphs and the 2-vertex-connected components (2VC). All those problems are NP-hard, and thus we are interested in approximation algorithms. Additionally, we aim at designing algorithms with a good practical performance, so that they are able to scale effectively to very large graphs.

While for 2ECS and 2VCS one can obtain an approximation ratio smaller than 2 by combining previously known results, providing good approximations for the 2-edge and the 2-vertex-components case seems more challenging. Here, we present linear-time approximation algorithms that achieve the following approximation guarantees:

- 4-approximation for 2ECC and 2EC, and
- 6-approximation for 2VCC and 2VC.

Also, augmented versions of our 2VCC algorithm computes a 6-approximation for maintaining both the 2-edge and the 2-vertex-connected components (2CC), and for maintaining all the 2-connectivity relations of G (2C), i.e., both the 2-edge and the 2-vertex-connected subgraphs and components. Moreover, we provide heuristics that improve the size of the computed subgraphs in practice, and conduct a thorough experimental study to assess their merits in practical scenarios.

1 Introduction

Let $G = (V, E)$ be a directed graph (digraph), with m edges and n vertices. Digraph G is *strongly connected* if there is a directed path from each vertex to every other vertex. The *strongly connected components* of G are its maximal strongly connected subgraphs. An edge (resp., a vertex) of G is a *strong bridge* (resp., a *strong articulation point*) if its removal increases the number of strongly

*This work is a rewritten and expanded combination of two conference papers [14, 17].

†University of Ioannina, Greece. E-mails: {loukas,charis}@cs.uoi.gr.

‡Università di Roma "Tor Vergata", Italy. E-mails: giuseppe.italiano@uniroma2.it, aikaranasiou@gmail.com, and nikos.parotsidis@uniroma2.it. Partially supported by MIUR under Project AMANDA.

connected components. A digraph G is 2-*edge-connected* if it has no strong bridges; G is *2-vertex-connected* if it has at least three vertices and no strong articulation points. Let $C \subseteq V$. The induced subgraph of C, denoted by $G[C]$, is the subgraph of G with vertex set C and edge set $E \cap (C \times C)$. If $G[C]$ is 2-edge-connected (resp., 2-vertex-connected), and there is no set of vertices C' with $C \subsetneq C' \subseteq V$ such that $G[C']$ is also 2-edge-connected (resp., 2-vertex-connected), then $G[C]$ is a *maximal 2-edge-connected (resp., 2-vertex-connected) subgraph of* G. Let v and w be two distinct vertices: v and w are 2-*edge-connected* (resp., 2-*vertex-connected*), denoted by $v \leftrightarrow_{2e} w$ (resp., $v \leftrightarrow_{2v} w$), if there are two edge-disjoint (resp., two internally vertex-disjoint) directed paths from v to w and two edge-disjoint (resp., two internally vertex-disjoint) directed paths from w to v (a path from v to w and a path from w to v need not be either vertex- or edge- disjoint). A 2-*edge-connected component* (resp., 2-*vertex-connected component*) of a digraph $G = (V, E)$ is a maximal subset $C \subseteq V$ such that $u \leftrightarrow_{2e} v$ (resp., $u \leftrightarrow_{2v} v$) for all $u, v \in C$. Note that, as a (degenerate) special case, a 2-edge-connected component (resp., 2-vertex-connected component) might consist of a singleton vertex only: we denote this as a *trivial* 2-*edge-connected component* (resp., 2-*vertex-connected component*). In the following, we will consider only non-trivial 2-edge and 2-vertex-connected components. Since there is no danger of ambiguity, we will call them simply 2-edge and 2-vertex-connected components.

Differently from undirected graphs, in digraphs 2-edge and 2-vertex connectivity have a much richer and more complicated structure, and indeed 2-connectivity problems on directed graphs appear to be more difficult than their undirected counterparts. In particular, in digraphs the 2-edge- (resp., 2-vertex-) connected components can be different from the maximal 2-edge- (resp., 2-vertex-) connected subgraphs, i.e., two vertices may be 2-edge- (resp., 2-vertex-) connected but lie in different maximal 2-edge- (resp., 2-vertex-) connected subgraphs (see Figure 1). This is not the case for undirected graphs. Moreover, for undirected graphs it has been known for over 40 years how to compute the 2-edge- and 2-vertex- connected components in linear time [30]. In the case of digraphs, however, it was shown only recently how to compute the 2-edge- and 2-vertex- connected components in linear time [15, 16], and the best current bound for computing the maximal 2-edge- and 2-vertex- connected subgraphs in digraphs is not even linear, but it is $O(\min\{m^{3/2}, n^2\})$ [4, 20].

A *spanning subgraph* G' of G has the same vertices as G and contains a subset of the edges of G. Computing a smallest spanning subgraph (i.e., one with minimum number of edges) that maintains the same edge or vertex connectivity properties of the original graph is a fundamental problem in network design, with many practical applications [29]. These involve the constructions of low-cost (smallest spanning subgraph) and highly connected networks so that the networks are fault tolerant to potential failures of nodes or edges. To name only few of them, telecommunication and traffic networks, and VLSI chip design are just two examples (see for e.g., [1, 29]). We also note that the type of sparse certificates that we consider allow to speed up computations, such as finding edge-disjoint or vertex-disjoint paths that connect a pair of vertices, or finding edges and vertices that separate a pair of vertices: indeed, these computations can be carried out in the sparse certificate instead of the input digraph, thus replacing m (the number of edges) by n (the number of vertices) in the complexity of the executed algorithm (see also [28].) Another important benefit is that a sparse certificate may occupy much less storage space than the original graph, which also makes it faster to transmit.

In this paper we consider the problem of finding the smallest spanning subgraph of G that maintains certain 2-connectivity requirements in addition to strong connectivity. In particular, for 2-edge-connectivity, we wish to compute the smallest strongly connected spanning subgraph of a digraph G that maintains the following properties:

- the pairwise 2-edge-connectivity of G, i.e., the 2-edge-connected components of G (2ECC);

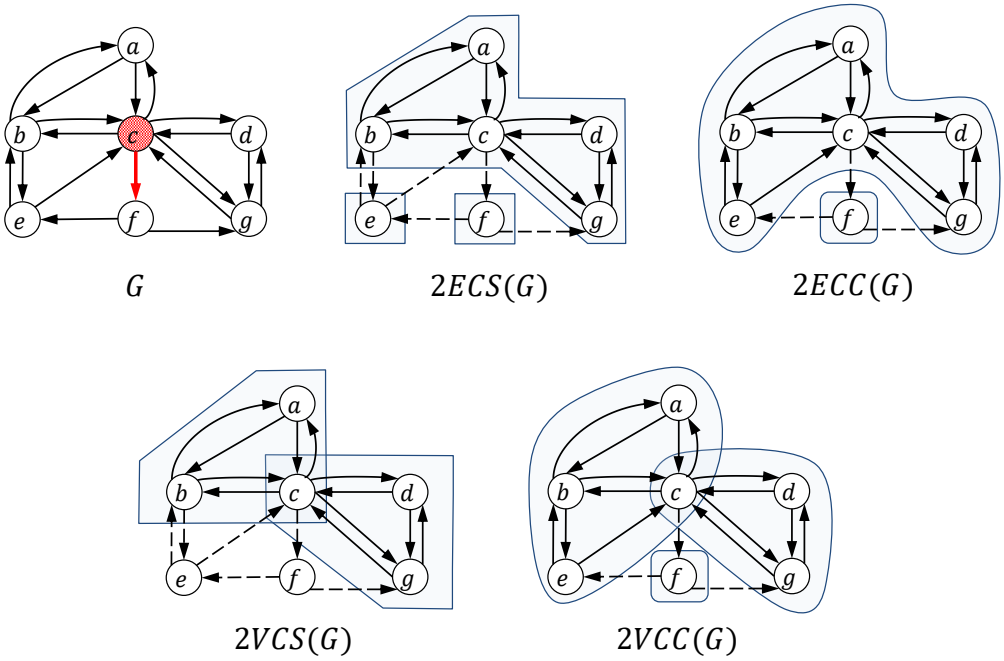


Figure 1: A strongly connected digraph G with a strong bridge (c, f) and a strong articulation point c shown in red (better viewed in color), the maximal 2-edge-connected subgraphs ($2ECS(G)$) and the 2-edge-connected components ($2ECC(G)$) of G, and the maximal 2-vertex-connected subgraphs ($2VCS(G)$) and the 2-vertex-connected components ($2VCC(G)$) of G. Vertex f forms a trivial 2-edge-connected and 2-vertex-connected component.

- the maximal 2-edge-connected subgraphs of G (2ECS);
- both the 2-edge-connected components and the maximal 2-edge-connected subgraphs of G (2EC).

Analogously, for 2-vertex-connectivity, we wish to compute the smallest strongly connected spanning subgraph G that maintains:

- the pairwise 2-vertex-connectivity of G, i.e., the 2-vertex-connected components of G (2VCC);
- the maximal 2-vertex-connected subgraphs of G (2VCS);
- both the 2-vertex-connected components and the maximal 2-vertex-connected subgraphs of G (2VC).

Finally, we also consider computing a smallest spanning subgraph of G that maintains:

- the pairwise 2-connectivity of G (2CC), that is, simultaneously the 2-edge- and 2-vertex-connected components;
- all the 2-connectivity relations of G (2C), that is, simultaneously the 2-edge- and 2-vertex-connected components and the maximal 2-edge- and 2-vertex- connected subgraphs.

Note that all these problems are NP-hard [12], so one can only settle for efficient approximation algorithms. Computing small spanning subgraphs is of particular importance when dealing with

large-scale graphs, say graphs having hundreds of million to billion edges. In this framework, one big challenge is to design linear-time algorithms (or close to linear-time algorithms), since algorithms with higher running times might be practically infeasible on today's architectures for such large-scale graphs. While for 2ECS and 2VCS one can achieve an approximation ratio less than 2 using known results, for the other problems no efficient approximation algorithms were previously known.

1.1 Related work

Computing a smallest k-vertex-(resp., k-edge-) connected spanning subgraph of a given k-vertex-(resp. k-edge-) connected digraph is NP-hard for any $k \geq 1$ (and for $k \geq 2$ for undirected graphs) [12]. The case for $k = 1$ is to compute a smallest strongly connected spanning subgraph (SCSS) of a given digraph. This problem was originally studied by Khuller et al. [23], who provided a polynomial-time algorithm with an approximation guarantee of 1.64. This was improved to 1.61 by the same authors [24]. Later on, Vetta announced a further improvement to 3/2 [32], and Zhao et al. [33] presented a faster linear-time algorithm at the expense of a larger 5/3-approximation factor. For the smallest k-edge-connected spanning subgraph (kECSS), Laehanukit et al. [26] gave a randomized $(1+1/k)$-approximation algorithm. For the smallest k-vertex-connected spanning subgraph (kVCSS), Cheriyan and Thurimella [5], gave a $(1 + 1/k)$-approximation algorithm that runs in $O(km^2)$ time. For $k = 2$, the running time of Cheriyan and Thurimella's algorithm was improved to $O(m\sqrt{n} + n^2)$, based on a linear-time 3-approximation for 2VCSS [13]. We also note that there has been extensive work on more general settings where one wishes to approximate minimum-cost subgraphs that satisfy certain connectivity requirements. See, e.g., [8], and the survey [25].

The previous results on kECSS and kVCSS immediately imply an approximation ratio smaller than 2 for 2ECSS and 2VCSS. A faster and deterministic 2-approximation algorithm for 2ECS can be obtained by combining techniques based on edge-disjoint spanning trees [7, 31] with the SCSS algorithm of Zhao et al. [33]. On the other hand, computing sparse subgraphs with the same pairwise 2-edge or 2-vertex connectivity seems substantially harder. Consider 2ECC, for instance. If the input graph consists of a single 2-edge-connected block then the problem asks for the smallest 2-edge-connected subgraph, whereas if the input graph consists of n singleton 2-edge-connected blocks then the problem coincides with SCSS. Jaberi [22] was the first to consider several optimization problems related to 2ECC and 2VCC and proposed approximation algorithms. The approximation ratio in his algorithms, however, is linear in the number of strong bridges for 2ECC and in the number of strong articulation points for 2VCC, and hence $O(n)$ in the worst case.

1.2 Our results

In this paper we provide both theoretical and experimental contributions to the 2ECC and 2VCC problems, and their generalizations 2EC, 2VC, 2CC, and 2C. Our goal is to provide practical algorithms that achieve good approximations with fast running times. Our starting point is the recent framework on strong connectivity and 2-connectivity problems in digraphs [15, 16, 18], combined with the notions of *divergent spanning trees* and *low-high orders* [19] (defined below). Building on this new framework, we can obtain *sparse certificates* also for the 2-edge- and 2-vertex- connected components of a digraph. In our context, a sparse certificate of a strongly connected digraph G is a strongly connected spanning subgraph $C(G)$ of G with $O(n)$ edges that maintains the 2-edge- or the 2-vertex- connected components of G. Sparse spanning subgraphs of this kind imply $O(1)$-approximation algorithms for 2ECC and 2VCC. Such sparse certificates have been provided in [16] and [15] for the 2-edge-connected components and the 2-vertex-connected components, respectively, but unfortunately, no good bound for the approximation constant was previously known, and indeed

achieving a small constant seemed to be non-trivial. In this paper, we make a substantial progress in this direction by presenting new 4-approximation algorithms for 2ECC and 6-approximation algorithms for 2VCC that run in linear time. Then, we extend our algorithms so that they compute a 4-approximation for 2EC, and a 6-approximation for 2VC, 2CC and 2C. These algorithms also run in linear time once the maximal 2-vertex and 2-edge-connected subgraphs of G are available; if not, the current best running time for computing them is $O(\min\{m^{3/2}, n^2\})$ [4, 20].

From the practical viewpoint, we provide efficient implementations of our algorithms that are very fast in practice. We further propose and implement several heuristics that improve the size (i.e., the number of edges) of the computed spanning subgraphs in practice. Some of these heuristics require $O(mn)$ time in the worst case, so we also present several techniques to achieve significant speedups in their running times. With all these implementations, we conduct a thorough experimental study and report its main findings. We believe that this is crucial to assess the merits of all the algorithms considered in practical scenarios.

The remainder of this paper is organized as follows. We introduce some preliminary definitions and graph-theoretical terminology in Section 2. Then, in Section 3 we describe our main algorithms that achieve a 4-approximation for 2ECC and a 6-approximation for 2VCC. In Sections 4 and 5 we extend these algorithms in order to obtain a 4-approximation for 2EC and a 6-approximation for 2VC and 2C. Our empirical study is presented in Section 6. Finally, in Section 7 we discuss some open problems and directions for future work.

2 Preliminaries

In this section, we introduce some basic terminology and results that will be useful throughout the paper.

2.1 Flow graphs, dominators, bridges, and edge-disjoint spanning trees.

A *flow graph* is a digraph such that every vertex is reachable from a distinguished start vertex. Let $G = (V, E)$ be a strongly connected digraph. For any vertex $s \in V$, we denote by $G(s) = (V, E, s)$ the corresponding flow graph with start vertex s; all vertices in V are reachable from s since G is strongly connected. The *dominator relation* in $G(s)$ is defined as follows: A vertex u is a *dominator* of a vertex w (u *dominates* w) if every path from s to w contains u. The dominator relation in $G(s)$ can be represented by a rooted tree, the *dominator tree* $D(s)$, such that u dominates w if and only if u is an ancestor of w in $D(s)$. See Figure 2. If $w \neq s$, we denote by $d(w)$ the parent of w in $D(s)$. The dominator tree of a flow graph can be computed in linear time, see, e.g., [2, 3].

An edge (u, w) is a *bridge* in $G(s)$ if all paths from s to w include (u, w).[1] Italiano et al. [21] gave linear-time algorithms for computing all the strong bridges and all the strong articulation points of a digraph G. Their algorithms use the dominators and the bridges of flow graphs $G(s)$ and $G^R(s)$, where s is an arbitrary start vertex and G^R is the digraph that results from G after reversing edge directions. A spanning tree T of a flow graph $G(s)$ is a tree with root s that contains a path from s to v for all vertices v. Two spanning trees T_1 and T_2 rooted at s are *edge-disjoint* if they have no edge in common. A flow graph $G(s)$ has two such spanning trees if and only if it has no bridges [31]. Two spanning trees are *maximally edge-disjoint* if the only edges they have in common are the bridges of $G(s)$. Two (maximally) edge-disjoint spanning trees can be computed

[1] Throughout, we use consistently the term *bridge* to refer to a bridge of a flow graph $G(s)$ and the term *strong bridge* to refer to a strong bridge in the original graph G.

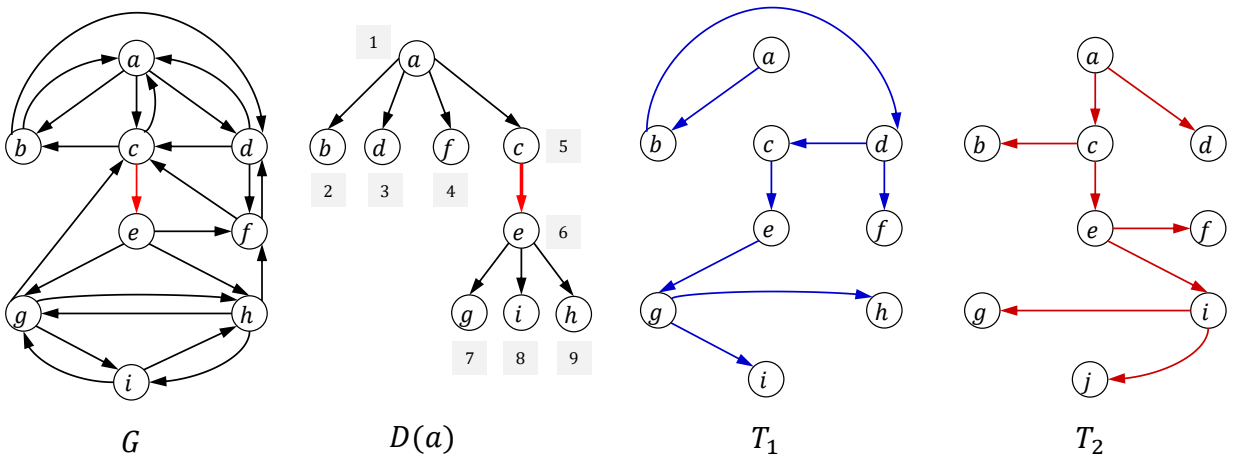


Figure 2: A strongly connected digraph G, with a strong bridge shown in red (better viewed in color). The dominator tree $D(a)$ of flow graph $G(a)$; the numbers correspond to a preorder numbering of $D(a)$ that is a low-high order of $G(a)$. Also, two divergent spanning trees T_1 and T_2 of $G(a)$ that are maximally edge-disjoint.

in linear-time by an algorithm of Tarjan [31], using the disjoint set union data structure of Gabow and Tarjan [11].

2.2 Divergent spanning trees and low-high orders

Two spanning trees T_1 and T_2 rooted at s are *divergent* if for all vertices v, the paths from s to v in T_1 and T_2 share only the dominators of v. A *low-high order* δ *on* $G(s)$ is a preorder of the dominator tree $D(s)$ such for all $v \neq s$, $(d(v), v) \in E$ or there are two edges $(u, v) \in E$, $(w, v) \in E$ such that u is less than v in δ ($u <_\delta v$), v is less than w in δ ($v <_\delta w$), and w is not a descendant of v in $D(s)$. See Figure 2. Every flow graph $G(s)$ has a pair of maximally edge-disjoint divergent spanning trees and a low-high order, both computable in linear-time [19]. Moreover, given a low-high order of G, it is straightforward to compute two strongly divergent spanning trees of G in $O(m)$ time [19].

Next, we state two useful properties of divergent spanning trees and low-high orders that follow from [19].

Property 2.1. *Let $T_1 = (V, E_1)$ and $T_2 = (V, E_2)$ be two divergent spanning trees of $G(s)$. Then, the flow graph $G'(s) = (V, E_1 \cup E_2, s)$ has the same dominator tree as $G(s)$.*

Fix a low-high order δ of $G(s)$ 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 $D(s)$.

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

2.3 Loop-nesting trees

Let $G = (V, E)$ be a strongly connected digraph. A *loop nesting tree* represents a hierarchy of strongly connected subgraphs of G [31], and is defined with respect to a dfs tree T of G as follows. Let T be a dfs tree of flow graph $G(s)$, rooted at some arbitrary vertex s. For a vertex u, we

denote by $loop(u)$ the set of all descendants x of u in T such that there is a path from x to u in G containing only descendants of u in T. Since any two vertices in $loop(u)$ reach each other, $loop(u)$ induces a strongly connected subgraph of G. Furthermore, *loops* define a laminar family (i.e., for any two vertices u and v, we have $loop(u) \cap loop(v) = \emptyset$, or $loop(v) \subseteq loop(u)$, or $loop(u) \subseteq loop(v)$). The above property allows us to define the *loop nesting tree* $L(s)$ of $G(s)$, with respect to T, as the rooted tree in which the parent of any vertex v, denoted by $\ell(v)$, is the nearest proper ancestor u of v in T such that $v \in loop(u)$. Since G is strongly connected and T is a dfs tree, $\ell(v)$ is well-defined. Moreover, every cycle contains a back edge with respect to T [30], and hence every cycle of G is contained in a loop [31]. Thus, $loop(u)$ is the set of all descendants of vertex u in $L(s)$.

The loop-nesting tree with respect to the dfs tree T of $G(a)$, is defined by at most $n-1$ additional edges (which are back-edges or cross-edges w.r.t. T) that specify the loops. See Figure 3. (We refer the interested reader to [19, 31] for the details.) A loop nesting tree can be computed in linear time [3, 31].

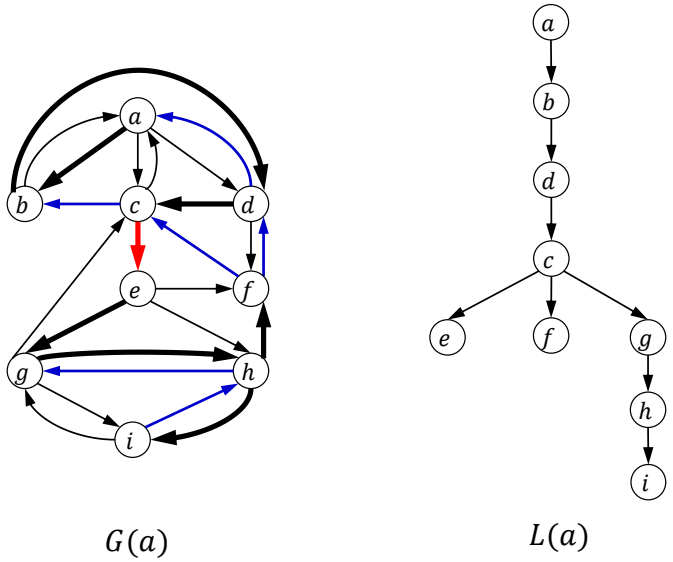


Figure 3: The flow graph $G(a)$ of Figure 2, where the edges of a dfs tree T rooted at vertex a are shown with thick lines (left); the blue edges together with the edges of T define the loop-nesting tree $L(a)$ of $G(a)$ (right).

2.4 Condensed graphs

We define two notions of condensed graphs, formed by contracting maximal 2-edge- or 2-vertex-connected subgraphs.

The *2ECS-condensed graph* is the digraph $K_E(G)$ obtained from G by contracting each maximal 2-edge-connected subgraph of G into a single supervertex. Note that $K_E(G)$ is a multigraph since the contractions can create loops and parallel edges; see Figure 4. For any vertex v of G, we denote by $\kappa(v)$ the supervertex of $K_E(G)$ that contains v. Every edge $(\kappa(u), \kappa(v))$ of $K_E(G)$ is associated with the corresponding original edge (u, v) of G. Given the condensed graph $K_E(G)$, we can obtain the *expanded graph* by reversing the contractions; each supervertex $\kappa(v)$ is replaced by the subgraph induced by the original vertices u with $\kappa(u) = \kappa(v)$, and each edge $(\kappa(u), \kappa(v))$ of $K_E(G)$ is replaced with the corresponding original edge (u, v). See Figure 4.

The 2ECS-condensed graph can be used to compute spanning subgraphs that maintain the

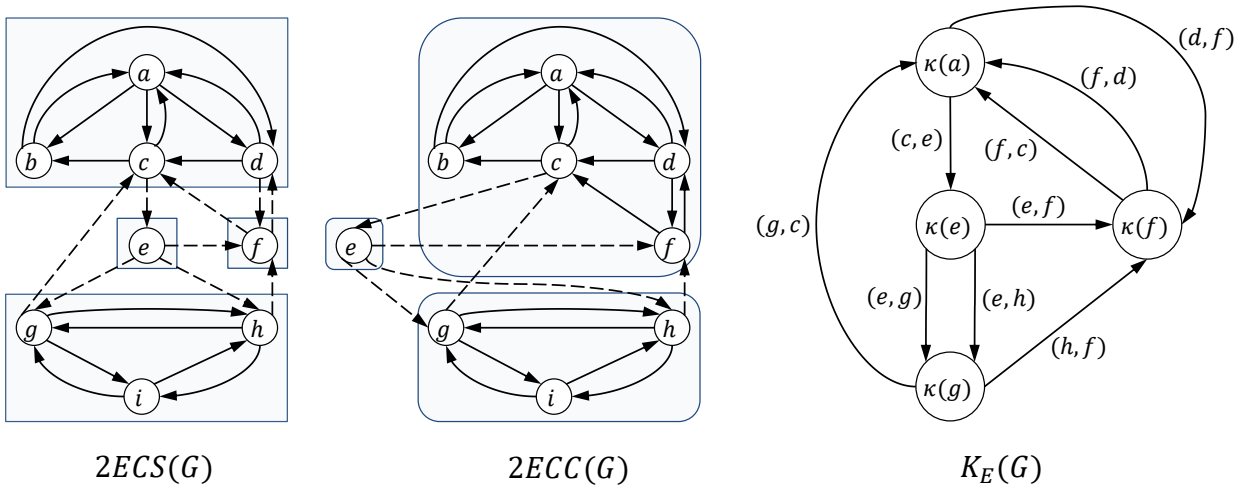


Figure 4: The maximal 2-edge-connected subgraphs (left) and the 2-edge-connected components (middle) of the digraph G of Figure 2. Also, the 2ECS-condensed graph $K_E(G)$ of G (right), where each edge of $K_E(G)$ is labeled with the corresponding original edge of G. Note that in $K_E(G)$ there are two edge-disjoint paths from $\kappa(a)$ to $\kappa(f)$ and two edge-disjoint paths from $\kappa(f)$ to $\kappa(a)$, hence the vertices $\{a, b, c, d\}$ contracted into $\kappa(a)$ and the vertices in $\{f\}$ contracted into $\kappa(f)$ are in the same 2-edge-connected component.

maximal 2-edge-connected subgraphs of G (problem 2ECS), as follows. First, we can apply an approximation algorithm for the 2ECSS problem independently for each 2-edge-connected subgraph of G. Then, it remains to preserve the strong connectivity of G. We can do that by solving the SCSS problem on the graph that results from the 2ECS-condensed graph $K = K_E(G)$ of G after removing loops and parallel edges. Let K' be the computed solution to SCSS. We obtain a solution G' to 2ECS by expanding the supervertices of K', i.e., we substitute each supervertex $\kappa(v)$ with the previously computed approximate solution to the corresponding 2ECSS problem.

For instance, we can obtain an approximation ratio less than 2 by applying the algorithm of Laehanukit et al. [26] for 2ECSS, combined with an efficient approximation algorithm for SCSS [23, 24, 32]. A faster (and more practical) deterministic 2-approximation algorithm for 2ECS can be obtained as follows. For the 2ECS problems we use edge-disjoint spanning trees to obtain a 2-approximation algorithm [7, 31]. Let C be a 2-edge-connected subgraph of G. We select an arbitrary vertex $v \in C$ as a root and compute two edge-disjoint spanning trees in the flow graph $C(v)$ and two edge-disjoint spanning trees in the reverse flow graph $C^R(v)$. The edges of these spanning trees give a 2-approximate solution C' for 2ECS on C. Then, we solve SCSS on K by applying the linear-time algorithm of Zhao et al. [33]. This yields a 2-approximation algorithm for 2ECS that runs in linear time once the 2-edge-connected subgraphs of G are available (if not, they can be computed in $O(\min\{m^{3/2}, n^2\})$ time [4, 20]). We refer to this algorithm as ZNI.

We will use 2ECS-condensed graphs in Section 4, where we show that they maintain the 2-edge-connected components of the original graph. This fact enables us to use condensed graphs in order to provide efficient approximation algorithms for 2EC.

We define similarly the *2VCS-condensed graph* of G, which is the digraph $K_V(G)$ obtained from G by contracting each maximal 2-vertex-connected subgraph of G into a single supervertex. See Figure 5. Note that any two 2-vertex-connected subgraphs may have at most one vertex in common: if two such subgraphs share a vertex, they are contracted into the same supervertex. It is not difficult to see that, differently from the 2ECS-condensed graph, the 2VCS-condensed graph

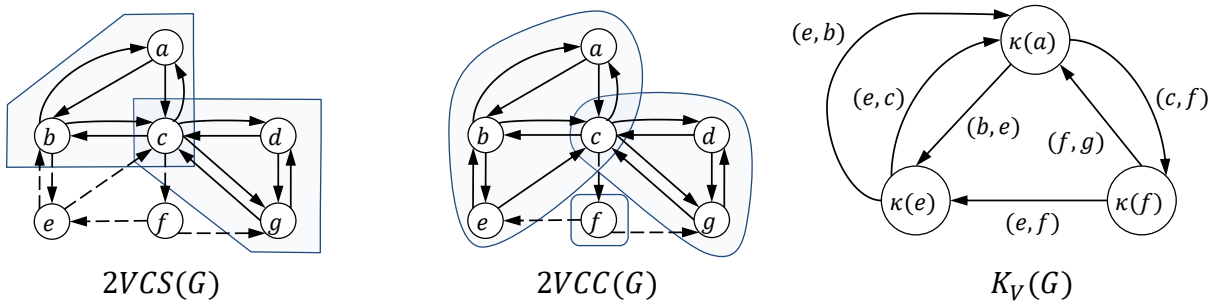


Figure 5: The maximal 2-vertex-connected subgraphs (left) of the digraph G of Figure 1, and the 2VCS-condensed graph $K_V(G)$ of G (right). Note that in $K_V(G)$ there are two vertex-disjoint paths from $\kappa(a)$ to $\kappa(e)$ and two vertex-disjoint paths from $\kappa(e)$ to $\kappa(a)$. However vertex e of $\kappa(e)$ is not in the same 2-vertex-connected component (middle) with any of d, g that are contracted into $\kappa(a)$. Thus $K_V(G)$ does not maintain the 2-vertex-connected components of G.

does not maintain the 2-vertex-connected components of the original graph. Still, we will be able to use the 2VCS-condensed graph in our approximation algorithm for 2C in Section 5.1.

3 Approximation algorithms for 2ECC and 2VCC

Let $G = (V, E)$ be the input strongly connected digraph. In problems 2ECC and 2VCC, we wish to compute a strongly connected spanning subgraph G' of G that has the same 2-edge-connected components and 2-vertex-connected components of G, respectively, with as few edges as possible. We consider the following approach for both problems. Start with the empty graph $G' = (V, \emptyset)$, and add as few edges as possible until G' is guaranteed to have the same 2-edge-connected components or 2-vertex-connected components as G. We present four new linear-time algorithms that apply this approach, one for each of the 2ECC and 2VCC problems, and two for 2CC, i.e., that solve both 2ECC and 2VCC simultaneously. The first two algorithms, that we refer to as DST-2ECC and DST-2VCC, use divergent spanning trees and are based on the sparse certificates for 2-edge- and 2-vertex-connected components from [15, 16]. The third algorithm, DLN, uses divergent spanning trees and loop-nesting trees, and is based on the sparse certificate from [18]. Finally, our fourth algorithm, LHL, combines low-high orders and loop-nesting trees.

3.1 Approximating 2ECC via Divergent Spanning Trees

Here we show how to achieve a 4-approximation for 2ECC by using divergent spanning trees. Our algorithm DST-2ECC is based on the sparse certificate $C(G)$ for the 2-edge-connected components of a digraph G [16] that we review first.

3.1.1 Sparse certificate for 2-edge-connected components

Let s be an arbitrarily chosen start vertex of the strongly connected digraph G. The *bridge decomposition* of the dominator tree $D(s)$ is the forest of rooted trees that results from $D(s)$ after the deletion of all the bridges of $G(s)$. Let D_v denote the tree containing vertex v in this decomposition. We refer to the subtree roots in the bridge decomposition as *marked vertices*. For each marked vertex r we define the *auxiliary graph* $G_r = (V_r, E_r)$ *of* r as follows. The vertex set V_r of G_r consists

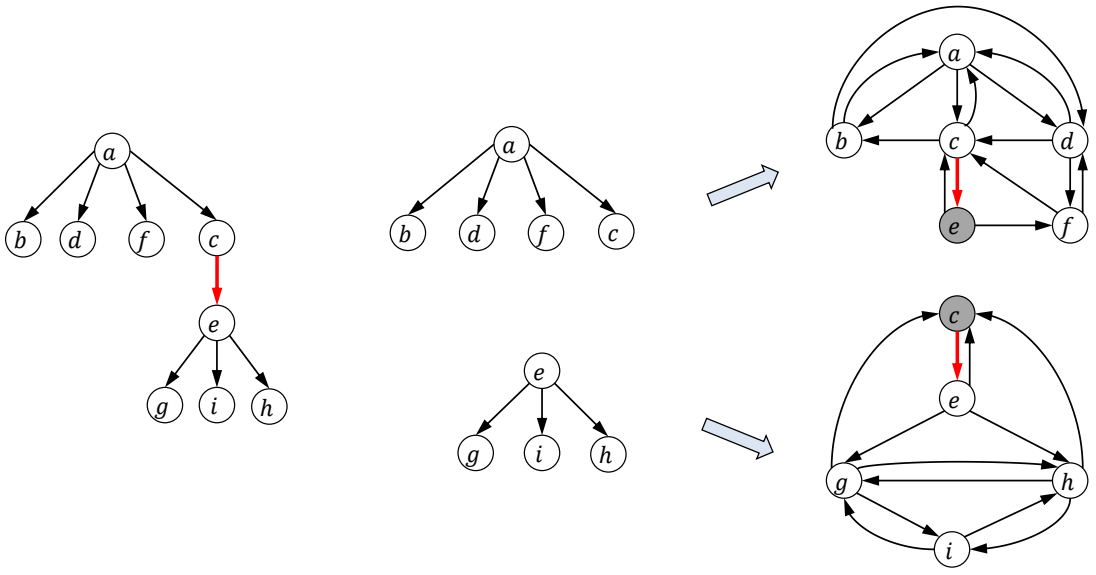


Figure 6: (a) The dominator tree $D(a)$ of the flow graph of Figure 2 with start vertex a. The strong bridge (c, e), shown in red (better viewed in color), appears as an edge of the dominator tree. (b) The subtrees D_a and D_e of the bridge decomposition of $D(a)$ after the deletion of (c, e), and (c) their corresponding first-level auxiliary graphs G_a and G_e. Auxiliary vertices are shown grey.

of all the vertices in D_r, referred to as *ordinary* vertices, and a set of *auxiliary* vertices, which are obtained by contracting vertices in $V \setminus D_r$:

- Let v be a vertex in D_r. We say that v is a *boundary vertex in* D_r if v has a marked child in $D(s)$. Let w be a marked child of a boundary vertex v: all the vertices that are descendants of w in $D(s)$ are contracted into w.
- All vertices in $V \setminus D_r$ that are not descendants of r are contracted into $d(r)$ ($r \neq s$ if any such vertex exists).

Figures 6 and 7 illustrate the bridge decomposition of a dominator tree and the corresponding auxiliary graphs.

During those contractions, parallel edges are eliminated. We call an edge in $E_r \setminus E$ *shortcut edge*. Such an edge has an auxiliary vertex as an endpoint. We associate each shortcut edge $(u, v) \in E_r$ with a corresponding original edge $(x, y) \in E$, i.e., x was contracted into u or y was contracted into v (or both). If $G(s)$ has b bridges then all the auxiliary graphs G_r have at most $n + 2b$ vertices and $m + 2b$ edges in total and can be computed in $O(m)$ time. As shown in [16], two ordinary vertices of an auxiliary graph G_r are 2-edge-connected in G if and only if they are 2-edge-connected in G_r. Thus the 2-edge-connected components of G are a refinement of the vertex sets in the trees of the bridge decomposition. The sparse certificate of [16] is constructed in three phases. We maintain a list (multiset) Λ of the edges to be added in $C(G)$; initially $\Lambda = \emptyset$. The same edge may be inserted into Λ multiple times, but the total number of insertions will be $O(n)$. So the edges of $C(G)$ can be obtained from Λ after we remove duplicates, e.g. by using radix sort. Also, during the construction, the algorithm may choose a shortcut edge or a reverse edge to be inserted into L. In either case we insert the associated original edge instead.

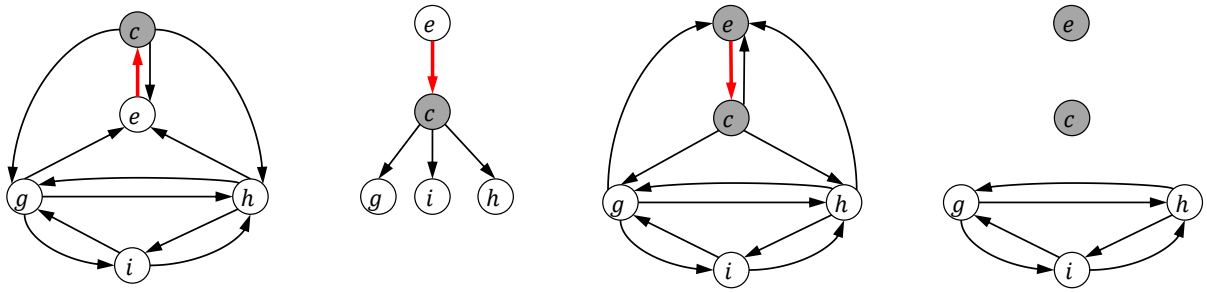


Figure 7: (a) The reverse graph H^R of the auxiliary graph $H = G_e$ of Figure 6. The strong bridge (c, e) of the original digraph, shown in red (better viewed in color), appears as the strong bridge (e, c) in H^R. (b) The dominator tree of $H^R(e)$ with start vertex e. (c) The second-level auxiliary graph H^R_c. Auxiliary vertices are shown grey. (d) The strongly connected components of $H^R_c \setminus (e, c)$. The strongly connected component $\{i, j, g\}$ is a 2-edge-connected block of the original digraph.

Phase 1. We insert into Λ the edges of two divergent spanning trees, T_1 and T_2 of $G(s)$.

Phase 2. For each auxiliary graph $H = G_r$ of $G(s)$, that we refer to as the *first-level auxiliary graphs*, we compute two divergent spanning trees T'_1 and T'_2 for the corresponding reverse flow graph $H^R(r)$ with start vertex r. We insert into Λ the edges of these two spanning trees. We note that Λ induces a strongly connected spanning subgraph of G at the end of this phase.

Phase 3. Finally, in the third phase we process the *second-level auxiliary graphs*, which are the auxiliary graphs of H^R for all first-level auxiliary graphs H. Let (p, q) be a bridge of $H^R(r)$, and let H^R_q be the corresponding second-level auxiliary graph. For every strongly connected component S of $H^R_q \setminus (p, q)$, we choose an arbitrary vertex $v \in S$ and compute a spanning tree of $S(v)$ and a spanning tree of $S^R(v)$, and insert their edges into Λ; see Figure 7.

The above construction inserts $O(n)$ edges into $C(G)$, and therefore achieves a constant approximation ratio for 2ECC. It is not straightforward, however, to give a good bound for this constant, since the spanning trees that are used in this construction contain auxiliary vertices that are created by computing twice a bridge decomposition and the corresponding auxiliary graphs, that is, once in the original graph and once in the reverse of each auxiliary graph. In the next section we introduce and analyze an improved version of the sparse certificate construction, and show that it achieves a 4-approximation for 2ECC.

3.1.2 Improved algorithm DST-2ECC

We now describe how to improve the sparse certificate of [16]. The main idea behind our improved version of DST-2ECC is to limit the number of edges added to the sparse certificate $C(G)$ because of auxiliary vertices. In particular, we show that in Phase 2 of the construction it suffices to add at most one new edge for each first-level auxiliary vertex, while in Phase 3 at most $2b$ additional edges are necessary for all second-level auxiliary vertices, where b is the number of bridges in $G(s)$.

We will use the following lemma about the strong bridges in auxiliary graphs, which implies that for any second-level auxiliary vertex x that was not an auxiliary vertex in the first level, subgraph $C(G)$ contains the unique edge leaving x in the first-level auxiliary graph.

Lemma 3.1. *Let (u, v) be a strong bridge of a first-level auxiliary graph $H = G_r$ that is not a bridge in $G(s)$. Then (v, u) is a bridge in the flow graph $H^R(r)$.*

Proof. Consider the dominator tree $D_H(r)$ of the flow graph $H(r)$. Let D' be the tree that results from $D_H(r)$ after the deletion of the auxiliary vertices. Then we have $D' = D_r$. Moreover, for each auxiliary vertex $x \neq d(r)$, $(d(x), x)$ is the unique edge entering x in H, which implies that $(d(x), x)$ is a bridge in both $G(s)$ and $H(r)$. Also, $(d(r), r)$ is the unique edge leaving $d(r)$ in H, hence $(d(r), r)$ is also a bridge in both $G(s)$ and $H(r)$. Thus, by construction, an edge of H is a bridge in $H(r)$ if and only if it is a bridge in $G(s)$. By [21] we have that a strong bridge of H must appear as a bridge of $H(r)$ or as the reverse of a bridge in $H^R(r)$, so the lemma follows. □

First we will describe our improved construction and apply a charging scheme for the edges added to $C(G)$ that are adjacent to auxiliary vertices. Then, we use this scheme to prove that the improved algorithm achieves the desired 4-approximation. Phase 1 remains the same and we explain the necessary modifications for Phases 2 and 3.

Improved Phase 2. Let $H = G_r$ be a first-level auxiliary graph. In the sparse certificate we include two divergent spanning trees, T'_1 and T'_2, of the reverse flow graph $H^R(r)$ with start vertex r. In our new construction, each auxiliary vertex x in H^R will contribute at most one new edge in $C(G)$. Suppose first that $x = d(r)$, which exists if $r \neq s$. The only edge entering $d(r)$ in H^R is $(r, d(r))$ which is the reverse edge of the bridge $(d(r), r)$ of $G(s)$. So $d(r)$ does not add a new edge in $C(G)$, since all the bridges of $G(s)$ were added in the first phase of the construction. Next we consider an auxiliary vertex $x \neq d(r)$. In H^R there is a unique edge (x, z) leaving x, where $z = d(x)$. This edge is the reverse of the bridge $(d(x), x)$ of $G(s)$. Suppose that x has no children in T'_1 and T'_2. Deleting x and its two entering edges in both spanning trees does not affect the existence of two edge-disjoint paths from v to r in H, for any ordinary vertex v. However, the resulting graph $C(G)$ at the end may not be strongly connected, since it may not have a path from x to r. To fix this, it suffices to include in $C(G)$ the reverse of an edge entering x from only one spanning tree. Finally, suppose that x has children, say in T'_1. Then $z = d(x)$ is the unique child of x in T'_1, and the reverse of the edge (x, z) of T'_1, i.e., edge $(d(x), x)$, is already included in $C(G)$ by Phase 1. Therefore, in all cases, we can charge to x at most one new edge.

Improved Phase 3. Let H^R_q be a second-level auxiliary graph of H^R. Let e be the strong bridge entering q in H^R, and let S be a strongly connected component in $H^R_q \setminus e$. In our sparse certificate we include the edges of a strongly connected subgraph of S, so we have spanning trees T and T^R of $S(v)$ and $S^R(v)$, respectively, rooted at an arbitrary ordinary vertex v. Let x be an auxiliary vertex of S. We distinguish two cases:

(i) If x is a first-level auxiliary vertex in H then it has a unique entering edge (w, x) which is a bridge in $G(s)$ already included in $C(G)$.

(ii) If x is ordinary in H but a second-level auxiliary vertex in H_q then it has a unique leaving edge $e_x = (x, z)$. By Lemma 3.1, e^R_x is a bridge in $H^R(r)$, so by (the improved) Phase 2 we have that $C(G)$ already contains a corresponding original edge leaving x.

Consider the first case. If x is a leaf in T^R then we can delete the edge entering x in T^R. Otherwise, w is the unique child of x in T^R, and the corresponding edge (w, x) entering x in H has already been inserted in $C(G)$. The symmetric arguments hold if x is ordinary in H.

This analysis implies that we can associate each second-level auxiliary vertex with one edge in each of T and T^R that is either not needed in $C(G)$ or has already been inserted. If all such auxiliary vertices are associated with distinct edges then they do not contribute any new edges in $C(G)$. Suppose now that there are two second-level auxiliary vertices x and y that are associated with a common edge e. This can happen only if one of these vertices, say y, is a first-level auxiliary vertex, and x is ordinary in H. Then y has a unique entering edge in H, which means that $e = (x, y)$ is a strong bridge, and thus already in $C(G)$. Also $e \in T$ and $e^R = (y, x) \in T^R$. In this case, we can treat x and y as a single auxiliary vertex that results from the contraction of e, which contributes at most two new edges in $C(G)$. Since y is a first-level auxiliary vertex, this can happen at most b times in all second-level auxiliary graphs, so a bound of $2b$ such edges follows.

Using the above construction we can now prove the following theorem.

Theorem 3.2. *Algorithm DST-2ECC runs in linear time and achieves an approximation ratio of 4 for the 2ECC problem.*

Proof. The running-time of DST-2ECC follows from the fact that the original sparse certificate of [16] can be computed in linear time. It is not difficult to verify that the improvements in Phases 2 and 3 require $O(n)$ additional time.

Let b denote (as above) the number of bridges in the flow graph $G(s)$. Note that $b \leq n - 1$. We consider the three phases of the construction of $C(G)$ separately and account for the new edges that are added in each phase. Consider the two divergent spanning trees T_1 and T_2 of $G(s)$ that are computed in the first phase. If an edge (u, v) is a bridge in $G(s)$ then it is the unique edge entering v in $T_1 \cup T_2$. Thus these two divergent spanning trees add into Λ exactly $2(n-b-1)+b = 2n-b-2$ edges.

Now we consider the Improved Phase 2. Let $H = G_r$ be a first-level auxiliary graph. Let o_r and a_r be, respectively, the number of ordinary and auxiliary vertices in G_r. In the sparse certificate we include two divergent spanning trees, T_1' and T_2', of the reverse flow graph $H^R(r)$ with start vertex r. As already explained in the analysis of this phase, each auxiliary vertex x in H^R may contribute at most one new edge in $C(G)$. Since r and $d(r)$ do not contribute any new edges, the total number of edges added for H is at most $2(o_r - 1) + (a_r - 1)$. Hence, the total number of edges added during the second phase is at most $\sum_r (2o_r + a_r - 3)$, where the sum is taken over all $b+1$ marked vertices r. Observe that $\sum_r o_r = n$ and $\sum_r a_r = 2b$, so we have $\sum_r (2o_r + a_r - 3) \leq 2n + 2b - 3b = 2n - b$. We note that, as in the original construction, $C(G)$ is strongly connected at the end of this phase. Moreover, in this phase we include in Λ the strong bridges of G that are not bridges in $G(s)$.

It remains to account for the edges added during the third phase. Here we consider the strongly connected components for each auxiliary graph H_q^R of H^R after removing the strong bridge entering q in H^R. By the argument in the description of the Improved Phase 3, the second-level auxiliary vertices contribute at most $2b$ new edges in total.

We note that the 2-edge-connected components of G are formed by the ordinary vertices in each strongly connected component computed for the second-level auxiliary graphs. Consider such a strongly connected component S. Let o_S be the number of ordinary vertices in S. If $o_S \leq 1$ then we do not include any edges for S. So suppose that $o_S \geq 2$. Excluding at most $2b$ additional edges, the auxiliary vertices in S do not contribute any new edges. So the number of edges added by S

is bounded by $2o_S$. Then, the third phase adds $2n' + 2b$ edges in total, where $n' = \sum_S o_S$ and the sum is taken over all strongly connected components with $o_S \geq 2$.

Overall, the number of edges added in $C(G)$ is at most $(2n-b-2)+(2n-b)+(2n'+2b) = 4n-2+2n' \leq 4(n+n')$. Next, we observe that these n' vertices must have indegree and outdegree at least equal to 2 in any solution to the 2ECC problem. The remaining $n - n'$ vertices must have indegree and outdegree at least equal to one, since the spanning subgraph must be strongly connected. Therefore, the smallest 2ECC has at least $(n - n') + 2n' = n + n'$ edges. The approximation ratio of 4 follows. □

Implementation details. In order to obtain an even more efficient implementation of DST-2ECC that achieves better solution quality in practice, we try to reuse as many edges as possible when we build the spanning trees in the three phases of the algorithm. In the third phase of the construction we need to solve the smallest SCSS problem for each subgraph H_S induced by a strongly connected component S in the second-level auxiliary graphs after the deletion of a strong bridge. To that end, we apply a modified version of the linear-time 5/3-approximation algorithm of Zhao et al. [33]. This algorithm computes a SCSS of a strongly connected digraph by performing a depth-first search (DFS) traversal. During the DFS traversal, any cycle that is detected is contracted into a single vertex. We modify this approach so that we can avoid inserting new edges into the sparse certificate as follows. Since we only care about the ordinary vertices in S, we can construct a subgraph of S that contains edges already added in $C(G)$. We compute the strongly connected components of this subgraph and contract them. Then we apply the algorithm of Zhao et al. on the contracted graph of S. Furthermore, during the DFS traversal we give priority to edges already added in $C(G)$. We can apply a similar idea in the second phase of the construction as well. The algorithm of [19] for computing two divergent spanning trees of a flow graph uses the edges of a DFS spanning tree, together with at most $n - 1$ other edges. Hence, we can modify the DFS traversal so that we give priority to edges already added in $C(G)$.

3.2 Approximating 2VCC via Divergent Spanning Trees

A sparse certificate $C(G)$ for the 2-vertex-connected components of a strongly connected digraph G was provided in [15]. Similarly to the 2-edge-connected components case, this sparse certificate is also based on divergent spanning trees and auxiliary graphs, but here the definition of auxiliary graphs is substantially different. As above, we review the construction of [15] first, and then describe our new improved algorithm DST-2VCC. Unfortunately, unlike the 2-edge-connected components case, we are not able to bound the approximation ratio of this algorithm by a small constant. We remedy this in the next sections, where we provide 6-approximations by constructions based on loop-nesting trees. These have the additional benefit of maintaining the 2-edge-connected components as well. Although our improved algorithm DST-2VCC does not achieve a very good approximation factor, our experimental study reveals that it produces certificates of good quality, and in several cases better than all the rest linear-time algorithms that we consider.

3.2.1 Sparse certificate for 2-vertex-connected components

Let s be an arbitrarily chosen start vertex in G. Recall that we denote by $G(s)$ the flow graph with start vertex s, by $G^R(s)$ the flow graph obtained from $G(s)$ after reversing edge directions, and by $D(s)$ and $D^R(s)$ the dominator trees of $G(s)$ and $G^R(s)$ respectively. Also, let $C(v)$ and $C^R(v)$ be the set of children of v in $D(s)$ and $D^R(s)$ respectively. For each vertex r, let $C^k(r)$ denote the level k descendants of r, where $C^0(r) = \{r\}$, $C^1(r) = C(r)$, and so on. For each vertex $r \neq s$ that

is not a leaf in $D(s)$ we build the *auxiliary graph* $G_r = (V_r, E_r)$ *of* r as follows. The vertex set of G_r is $V_r = \cup_{k=0}^{3} C^k(r)$ and it is partitioned into a set of *ordinary* vertices $V_r^o = C^1(r) \cup C^2(r)$ and a set of *auxiliary* vertices $V_r^a = C^0(r) \cup C^3(r)$. The auxiliary graph G_r results from G by contracting the vertices in $V \setminus V_r$ as follows. All vertices that are not descendants of r in $D(s)$ are contracted into r. For each vertex $w \in C^3(r)$, we contract all descendants of w in $D(s)$ into w. We use the same definition for the auxiliary graph G_s of s, with the only difference that we let s be an ordinary vertex. In order to bound the size of all auxiliary graphs, we eliminate parallel edges during those contractions. We call an edge $e \in E_r \setminus E$ a *shortcut* edge of G_r. That is, a shortcut edge is formed by the contraction of a part of G into an auxiliary vertex of G_r. Thus, a shortcut edge is not an original edge of G but corresponds to at least one original edge, and is adjacent to at least one auxiliary vertex.

The algorithm of [15] selects the edges that are inserted into $C(G)$ in three phases. During the construction, the algorithm may choose a shortcut edge or a reverse edge to be inserted into $C(G)$. In this case we insert the associated original edge instead. Also, an edge may be selected multiple times, so we remove multiple occurrences of such edges in a postprocessing step.

Phase 1. In the first phase, we insert into $C(G)$ the edges of two maximally edge-disjoint divergent spanning trees, T_1 and T_2 of $G(s)$.

Phase 2. In the second phase we process the auxiliary graphs of $G(s)$ that we refer to as the *first-level auxiliary graphs*. For each such auxiliary graph $H = G_r$, we compute two maximally edge-disjoint divergent spanning trees T_1' and T_2' of the corresponding reverse flow graph $H^R(r)$ with start vertex r. We insert into $C(G)$ the edges of these two spanning trees. It can be proved that, at the end of this phase, $C(G)$ induces a strongly connected spanning subgraph of G.

Phase 3. Finally, in the last phase we process the *second-level auxiliary graphs*, which are the auxiliary graphs of H^R for all first-level auxiliary graphs H. Let H_q^R be a second-level auxiliary graph of H^R. For every strongly connected component S of $H_q^R \setminus q$, we choose an arbitrary vertex $v \in S$ and compute a spanning tree of S and a spanning tree of S^R, and insert their edges into $C(G)$.

This construction inserts $O(n)$ edges into $C(G)$, and therefore achieves a constant approximation ratio for 2VCC. However, due to the use of auxiliary vertices and two levels of auxiliary graphs, we do not have a good bound for this constant. (The first-level auxiliary graphs have at most $4n$ vertices and $4m + n$ edges in total [15].)

3.2.2 Improved algorithm DST-2VCC

We can obtain a more efficient version of algorithm DST-2VCC by applying analogous optimizations to the ones mentioned for DST-2ECC at the end of Section 3.1.2. That is, we try to reuse as many edges as possible when we build the divergent spanning trees of $G(s)$ and of its auxiliary graphs (Phases 1 and 2), and when we solve the SCSS instances in the second-level auxiliary graphs (Phase 3).

As for the original version of DST-2VCC, we are not able to bound the approximation ratio of the improved algorithm by a small constant. In the next section, however, we provide algorithms that achieve a 6-approximation with the help of loop-nesting trees.

3.3 Approximating 2CC via Loop Nesting Trees

In this section we present two new linear-time algorithms that achieve a 6-approximation for the problem of computing a smallest spanning subgraph that maintains both the 2-edge-connected and the 2-vertex-connected components of the input digraph.

3.3.1 Divergent Spanning Trees and Loop Nesting Trees

A linear-time algorithm to compute a sparse certificate $C(G)$ for both the 2-edge- and 2-vertex-connected components can be obtained via divergent spanning trees and loop nesting trees [18]. We refer to this algorithm as DLN. As in algorithm DST-2VCC, in DLN we compute two maximally edge-disjoint divergent spanning trees T_1 and T_2 of $G(s)$, and insert their edges into $C(G)$. But instead of computing auxiliary graphs, we compute a loop nesting tree L of $G(s)$ and insert into $C(G)$ the edges that define L. These are the edges of the dfs tree of $G(s)$ that defines L, and at most $n-1$ additional edges that are required to maintain the loops of $G(s)$. (See Figure 3.) These additional edges can be selected while the algorithm discovers the loops during a dfs of $G(s)$. (We refer the interested reader to [19, 31] for the details.) Notice that by adding the edges of the dfs tree and the edges that define L, we inserted into $C(G)$ the same loop nesting tree as in G (L is defined with respect to a dfs tree, but it is sufficient that the same dfs tree can be also produced in $C(G)$). Then, we repeat the same process in the reverse direction, i.e., for $G^R(s)$. By Property 2.1, $C(G)$ has the same dominator trees (one in the forward graph and one in the reverse graph) as $G(s)$. During this construction, DLN tries to reuse as many edges as possible. As shown in [18], a spanning subgraph having the same dominator trees and loop nesting trees (in both directions) as the digraph G, has the same 2-edge- and 2-vertex-connected components as G.

Theorem 3.3. *Algorithm DLN runs in linear time and achieves an approximation ratio of* 6 *for problem 2CC.*

Proof. Consider first the "forward" pass of the algorithm. It adds at most $2(n-1)$ edges for the two divergent spanning trees, and at most $2(n-1)$ edges that define a loop nesting tree of $G(s)$. By [19, 31], both these constructions use the edges of a dfs tree of $G(s)$ and some additional edges. Hence, we can use the same dfs tree to compute the divergent spanning trees and the loop nesting tree. This gives a total of at most $3(n-1)$ edges. Similarly, the "reverse" pass computes at most $3(n-1)$ edges, so algorithm DLN selects at most $6(n-1)$ edges. Since the resulting subgraph must be strongly connected, any valid solution to problem 2CC has at least n edges, so DLN achieves a 6-approximation. By [19, 31], both the computation of a pair of divergent spanning trees and of a loop nesting tree can be done in linear time. Hence, DLN also runs in linear time. □

3.3.2 Low-High Orders and Loop Nesting Trees

Now we introduce a new linear-time construction of a sparse certificate for 2CC, via low-high orders, that we refer to as LHL. The algorithm consists of two phases. In the first phase, we insert into $C(G)$ the edges that define the loop nesting trees L and L^R of $G(s)$ and $G^R(s)$, respectively, as in algorithm DLN. In the second phase, we insert enough edges so that $C(G)$ (resp., $C^R(G)$) maintains a low-high order of $G(s)$ (resp., $G^R((s))$. Let δ be a low-high order on $G(s)$. Subgraph $C(G)$ satisfies the low-high order δ if, for each vertex $v \neq s$, one of the following holds:

(a) there are two edges (u, v) and (w, v) in $C(G)$ such that $u <_\delta v$, $v <_\delta w$, and w is not a descendant of v in $D(s)$;

(b) $(d(v), v)$ is a strong bridge of G and is contained in $C(G)$; or

(c) $(d(v), v)$ is an edge of G that is contained in $C(G)$, and there is another edge (u, v) in $C(G)$ such that $u <_\delta v$ and $u \neq d(v)$.

We note that the output of LHL, i.e., the sparse certificate $C(G)$, also maintains the 2-edge- and 2-vertex-connected components of the input digraph. This follows from the fact that $C(G)$ has the same loop nesting trees and, by Property 2.2, the same dominator trees as $G(s)$ (in both directions).

Theorem 3.4. *Algorithm LHL is correct and achieves an approximation ratio of* 6 *for problem 2CC in linear time.*

Proof. By construction, the sparse certificate $C(G)$ computed by LHL satisfies a low-high order δ of $G(s)$. This implies that $C(G)$ contains two divergent spanning trees T_1 and T_2 of $G(s)$ [19]. Moreover, cases (b) and (c) of the construction ensure that T_1 and T_2 are maximally edge-disjoint. This is because when case (a) does not apply for a vertex v, then $C(G)$ contains $(d(v), v)$. Also, $d(v)$ is the only vertex u that satisfies $u <_\delta v$ if and only if $(d(v), v)$ is a strong bridge. Hence, $C(G)$ indeed contains two maximally edge-disjoint divergent spanning trees of $G(s)$. Similarly, $C(G)$ also contains two maximally edge-disjoint divergent spanning trees of $G^R(s)$. So the correctness of LHL follows from the fact that DLN is correct.

Next we bound the approximation ratio of LHL. The edges selected to maintain a loop nesting tree L of $G(s)$ contain at least one entering edge for each vertex $v \neq s$. This means that it remains to include at most one edge for each vertex $v \neq s$ in order to satisfy a low-high order of $G(s)$. The symmetric arguments holds for the reverse direction as well, so $C(G)$ contains at most $6(n-1)$ edges, which gives an approximation ratio of 6. □

3.4 Speed-up techniques and heuristics

In this section we describe some heuristics that are able to improve the quality of the solutions computed by our previous algorithms. These improvements come at the expense of the running times, as some of these heuristics require quadratic time in the worst case. Consequently, we also present several techniques to achieve significant speedups in their running times. The main idea is to use a simple filter that decides whether a specific edge of the current subgraph G' is really needed or can be discarded. Specifically, we consider two main techniques that process one edge (x, y) of the current subgraph G' of G at a time, and test if it is safe to remove (x, y). Initially $G' = C(G)$, where $C(G)$ is an appropriate sparse certificate depending on the problem we wish to solve, and the order in which the edges are processed is arbitrary.

Disjoint Paths Test. We test if $G' \setminus (x, y)$ contains two disjoint paths from x to y. If this is the case, then we remove the edge (x, y); otherwise, we keep the edge (x, y) in G' and proceed with the next edge. More precisely, for the 2-edge-connectivity problems, the disjoint paths from x to y need only be edge-disjoint, while for the 2-vertex-connectivity problems we require vertex-disjoint paths. Both cases can be tested by running two iterations of the Ford-Fulkerson augmenting paths algorithm [9].

For the vertex-disjoint paths test we define a modified graph G'' of G' after vertex-splitting (see, e.g., [1]): for each vertex v, replace v by two vertices v^+ and v^-, and add the edge (v^-, v^+). Then, we replace each edge (u, v) in G' by (u^+, v^-) in G'', so v^- has the edges entering v and v^+ has the edges leaving v. By this construction, G' has two vertex-disjoint paths from x to y if and only if G'' has two edge-disjoint paths from x^+ to y^-. Note that we need to compute G'' once for all such tests. If an edge (x, y) is deleted from G', then we also delete (x^+, y^-) from G''.

Also note that both G' and G'' have $O(n)$ edges. Hence, both the two edge-disjoint paths test (Test2EDP) and the two vertex-disjoint paths test (Test2VDP) take $O(n)$ time per edge, so their total running time is $O(n^2)$.

2-Connected Components Test. If (x, y) is not a strong bridge in G', we test if $G' \setminus (x, y)$ has the same 2-connected components as G'. If this is the case then we remove the edge (x, y). Specifically, for the 2-edge-connectivity problems, we test the 2-edge connected components of G', while for the 2-vertex-connectivity problems we test the 2-vertex connected components of G'. We refer to these filters as Test2ECC and Test2VCC, respectively. Since the 2-edge-connected components and the 2-vertex-connected components of a graph can be computed in linear time [15, 16], both filters run in $O(n)$ time per edge, and so $O(n^2)$ time overall.

We note that Test2EDP (resp., Test2VDP) computes a minimal 2-approximate solution for the 2ECSS (resp., 2VCSS) problem [5], which is not necessarily minimal for the 2ECC (resp., 2VCC) problem. On the other had, Test2ECC and Test2VCC produce minimal solutions of the problems 2ECC and 2VCC, respectively. Also, we remark that one may apply the above filters directly on the input graph G, i.e., initialize $G' = G$ rather than $G' = C(G)$, but then their total running time becomes $O(m^2)$ which is impractical for large graphs.

The two approaches above can be combined into a hybrid algorithm:

Hybrid 2-Edge Connected Components Test. If the tested edge (x, y) connects vertices in the same 2-edge-connected component (i.e., $x \leftrightarrow_{2e} y$), then apply the Test2EDP filter; otherwise, apply the Test2ECC filter.

Hybrid 2-Vertex Connected Components Test. If the tested edge (x, y) connects vertices in the same 2-vertex-connected component (i.e., $x \leftrightarrow_{2v} y$), then apply the Test2VDP filter; otherwise, apply the Test2VCC filter.

One can show that for the same order of edges, the hybrid 2-edge connected components test returns the same minimal sparse subgraph as Test2ECC.

Lemma 3.5. *Let (x, y) be an edge of G. Algorithm Test2EDP deletes (x, y) only if Test2ECC does as well. Moreover, if x and y belong to the same 2-edge-connected component of G, then algorithms Test2EDP and Test2ECC are equivalent for (x, y), i.e., edge (x, y) is deleted by Test2ECC if and only if it is deleted by Test2EDP.*

Proof. To prove the first part of the lemma, suppose that (x, y) is deleted by Test2EDP. We show that the 2-edge-connected components of G are not affected by this deletion. Consider any pair of 2-edge-connected vertices u and w that was affected by the deletion of (x, y), that is, the number of edge-disjoint paths from u to w was reduced. Let (U, W) be a minimum u-w cut in $G \setminus (x, y)$, i.e., $U, W \subseteq V$, $U \cap W = \emptyset$, $u \in U$ and $w \in W$. Then we also have $x \in U$ and $y \in W$. Since $G \setminus (x, y)$ has at least two edge-disjoint paths from x to y, Menger's theorem [27] implies that there are at least two edges directed from U to W (see also, e.g., [1]). Thus, Menger's theorem implies $G \setminus (x, y)$ has at least two edge-disjoint paths from u to w.

We now prove the second part of the lemma. Suppose that x and y lie in the same 2-edge-connected component of G, and edge (x, y) is deleted by algorithm Test2ECC. This implies that $G \setminus (x, y)$ has two edge-disjoint paths from x to y, so algorithm Test2EDP would also delete (x, y). □

The proof for the 2-vertex-connectivity case is analogous. In our experiments, Test2VCC and the corresponding hybrid filter were not competitive with the other algorithms in terms of running times, so we exclude them from further consideration. Therefore, we refer to the hybrid 2-edge connected components test simply as Hybrid.

Trivial Edges. In order to improve the running time of the above filters, we can apply an additional speed-up heuristic in order to avoid testing edges that trivially belong to the computed solution. In the 2-edge-connectivity case, we say that (x, y) is a *trivial edge* of the current graph G' if it satisfies one of the following conditions:

- x belongs to a 2-edge-connected component of size at least two (nontrivial component) and has outdegree two, or y belongs to a 2-edge-connected component of size at least two (nontrivial component) and has indegree two;
- x has outdegree one, or y has indegree one. This is necessary to preserve strong connectivity.

Clearly, the removal of a trivial edge will result in a digraph that either has different 2-edge-connected components or is not strongly connected. Therefore these edges should remain in G'. As we show later in our experiments, such a simple test can yield significant performance gains. We apply the analogous heuristic in the 2-vertex-connectivity case.

3.5 Additional heuristics applied on auxiliary graphs for 2ECC

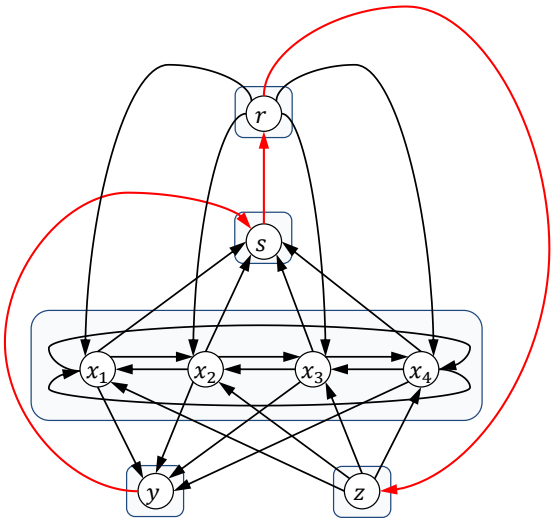
To speed up the Test2EDP and Hybrid algorithms, we applied them to the first-level and second-level auxiliary graphs. Since auxiliary graphs are supposed to be smaller than the original graph, one could expect to obtain some performance gain at the price of a slightly worse approximation. However, this performance gain cannot be taken completely for granted, as auxiliary vertices and shortcut edges may be repeated in several auxiliary graphs. Our experiments indicated that applying this heuristic to second-level auxiliary graphs yields better results than the ones obtained on first-level auxiliary graphs. We refer to those algorithms as Test2EDP-Aux and Hybrid-Aux

We also applied the corresponding heuristics for the 2VCC algorithms but they did not perform so well due to the more complicated structure of the auxiliary graph that maintain 2-vertex-connectivity.

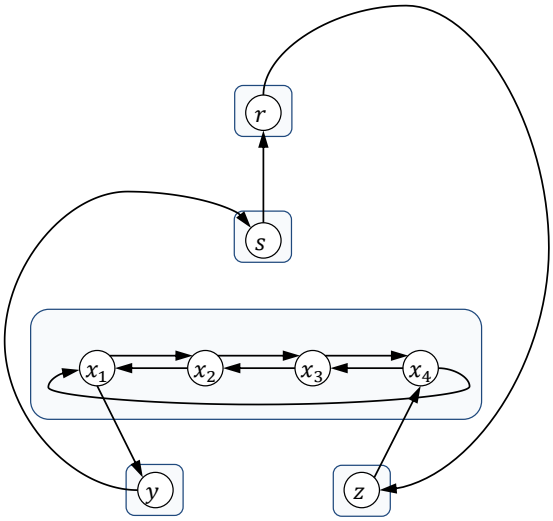
3.6 Tightness of approximation

We do not know if the approximation ratio of 4 (resp., 6) that we provided for algorithms DST-2ECC (resp., DLN and LHL) are tight. Figure 8(a) shows a digraph G such that a sparse certificate constructible by algorithm DST-2ECC has $6n + O(1)$ edges. This digraph has a single nontrivial 2-edge-connected component consisting of the vertices $x_1, x_2, \ldots, x_k$, which also form a 2-edge-connected subgraph. An optimal solution for 2ECC on this instance, shown in Figure 8(b), has $2n + O(1)$ edges, where each vertex x_i has indegree and outdegree equal to two, while the other four vertices have indegree and oudegree equal to one. Figure 8(c) shows a minimal solution with $3n + O(1)$ edges, where again each vertex x_i has indegree and outdegree equal to two but vertex y has indegree equal to k and vertex z has outdegree equal to k; removing any edge of this minimal solution either destroys the strong connectivity of the subgraph or partitions the nontrivial 2-edge-connected component.

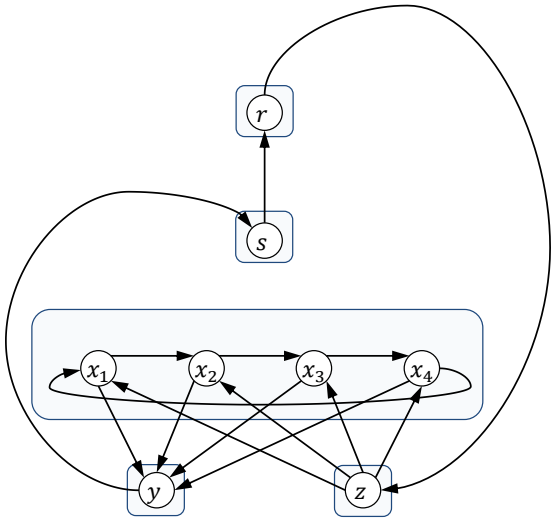
Thus, for the graph family of Figure 8(a), DST-2ECC achieves a 3-approximation. The three phases of the sparse certificate construction by DST-2ECC are given in Figures 9, 10 and 11. Algorithms Test2ECC and Hybrid, on the other hand, achieve a 3/2-approximation for this instance. We also note that this example is not a worst-case instance for Test2ECC and Hybrid. If the input digraph is 2-edge-connected then we seek for a smallest 2-edge-connected spanning subgraph, and Lemma 3.5 implies that Test2ECC and Hybrid produce the same output as Test2EDP. So, in this case Test2ECC and Hybrid achieve an approximation ratio of 2, which is known to be tight [5].



(a)



(b)



(c)

Figure 8: (a) A digraph G with $n = k + 4$ vertices and $m = 6n - 21$ edges (in this instance $k = 4$). Strong bridges are shown in red (better viewed in color). Digraph G has a single nontrivial 2-edge-connected component consisting of the vertices $x_1, x_2, \ldots, x_k$. (b) A minimum solution for the 2ECC problem with $2n - 4$ edges. (c) A minimal solution for the 2ECC problem with $3n - 9$ edges.

4 Approximation algorithms and heuristics for 2EC

Although all the above algorithms do not maintain the 2-edge-connected subgraphs of the original graph, we can still apply them to get an approximation for 2EC, as follows. First, we compute

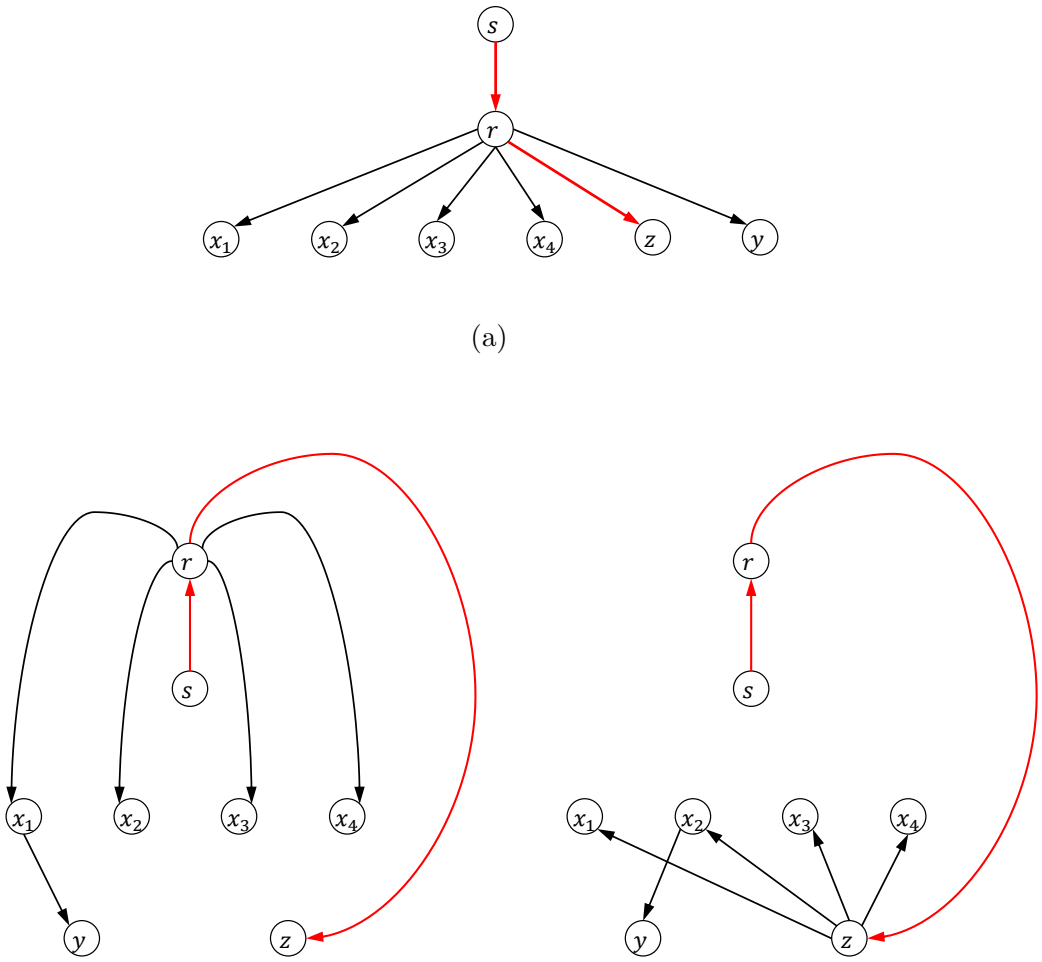


Figure 9: (a) The dominator tree of the flow graph $G(s)$ that corresponds to digraph G of Figure 8. (b) and (c) Two divergent spanning trees of $G(s)$ that may be selected by Phase 1 of the sparse certificate construction.

the maximal 2-edge-connected subgraphs of G and solve the 2ECS problem independently for each such subgraph. Then, we can apply any of the algorithms for 2ECC (Test2EDP, Test2ECC, Hybrid or DST-2ECC) for the edges that connect different subgraphs. To speed them up, we apply them to the 2ECS-condensed graph $K = K_E(G)$ of G. Let K' be the subgraph of K computed by any of the above heuristics, and let G' be the expanded graph of K', where we replace each supervertex

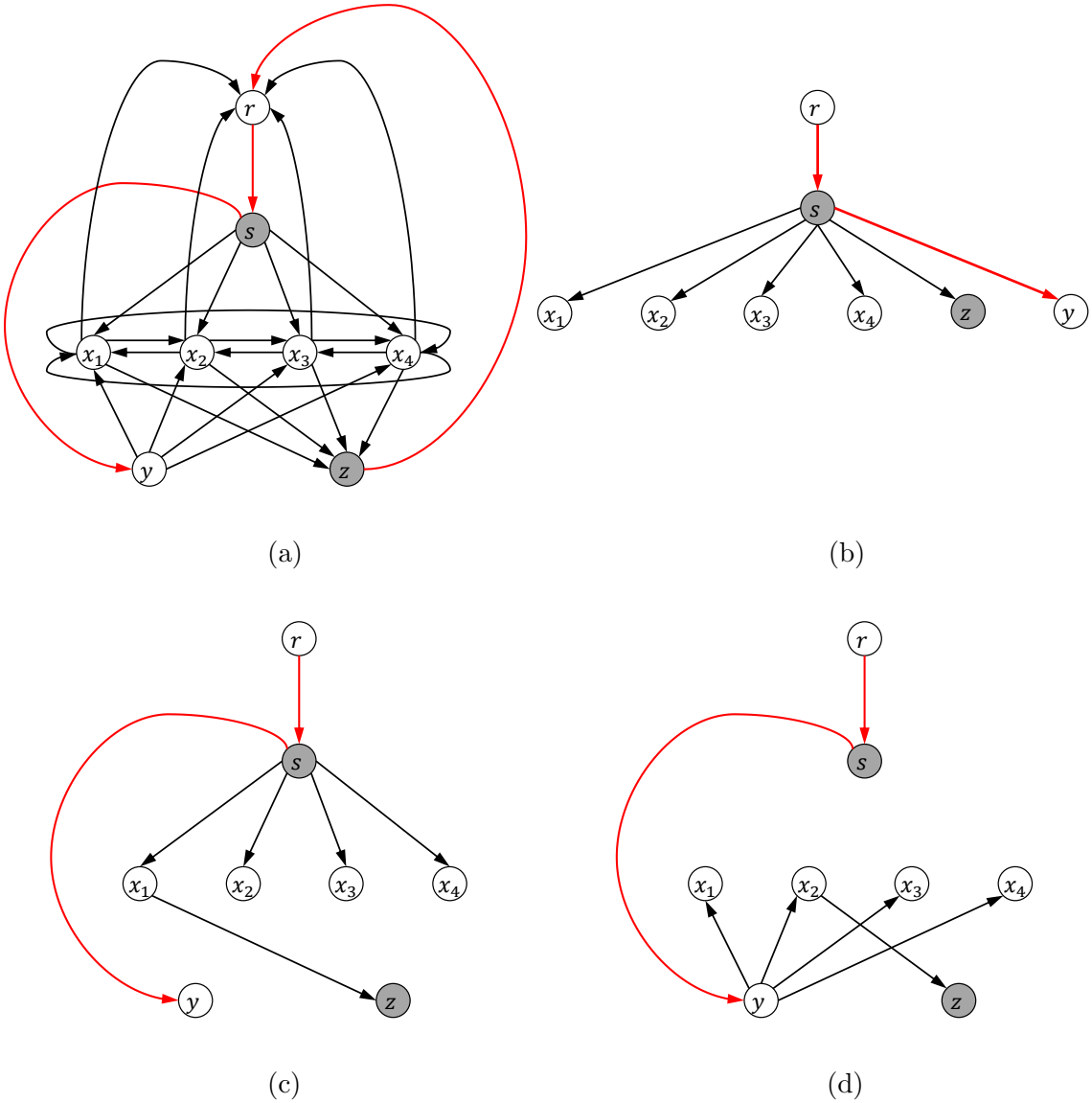
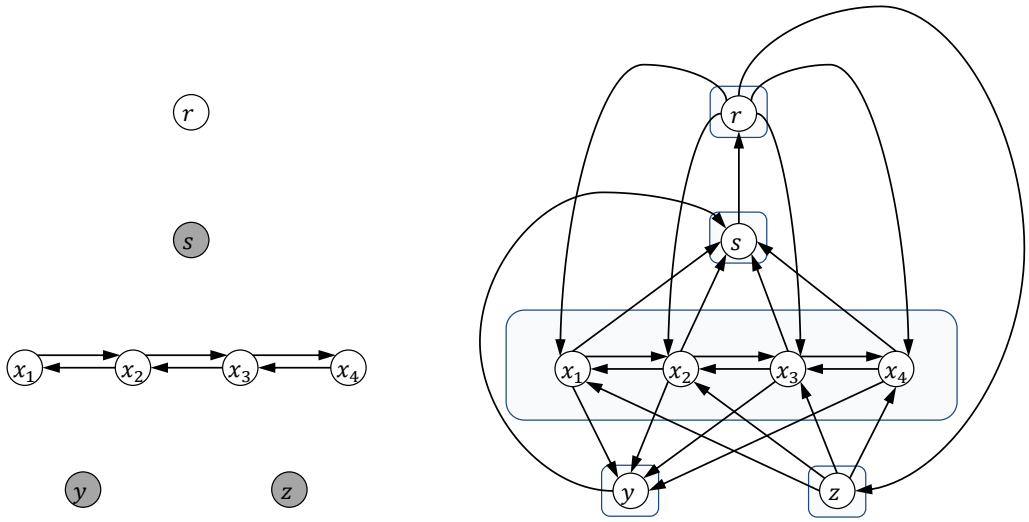


Figure 10: (a) The reverse graph H^R of the auxiliary graph $H = G_r$ of Figure 9. Auxiliary vertices are shown in grey. (b) The dominator tree of $H^R(r)$ with start vertex r. (c) and (d) Two divergent spanning trees of $H^R(r)$ that may be selected by Phase 2 of the sparse certificate construction.

of K with the corresponding 2-edge-connected sparse subgraph computed before. We refer to the corresponding algorithms obtained this way as Test2EDP-2EC, Test2ECC-2EC, Hybrid-2EC, and DST-2EC. The next lemma shows that indeed G' is a valid solution to the 2EC problem.

Lemma 4.1. *Digraph G' is strongly connected and has the same 2-edge-connected subgraphs and components as G.*

Proof. Digraph G' is strongly connected by construction, since K' and the spanning subgraphs computed for each supervertex of K are strongly connected. It is also clear that G' and G have the same 2-edge-connected subgraphs. So it remains to consider the 2-edge-connected components. Let u and w be two arbitrary vertices of G. We show that u and w are 2-edge-connected in G' if



(a) (b)

Figure 11: (a) The strongly connected components of $H_s^R \setminus (r, s)$, where H_s^R is the second-level auxiliary graph of H^R of Figure 10. The only nontrivial component is induced by the vertices $x_1, x_2, \ldots, x_k$. The edges shown may be selected by Phase 3 of the sparse certificate construction. (b) The final sparse certificate with $6n - 23$ edges.

and only if they are 2-edge-connected in G. The "only if" direction follows from the fact that G' is a subgraph of G. We now prove the "if" direction. Suppose u and w are 2-edge-connected in G. If u and w are located in the same 2-edge-connected subgraph then obviously they are 2-edge-connected in G'. Suppose now that u and w are located in different subgraphs, so $\kappa(u) \neq \kappa(w)$. By construction, for any (S, T) cut in K such that $\kappa(u) \in S$ and $\kappa(v) \in T$ there are at least two edges directed from S to T and at least two edges directed from T to S. This property is maintained by all algorithms, so it also holds in K'. Then, for any (U, W) cut in the expanded graph G' such that $u \in U$ and $w \in W$ there are at least two edges directed from U to W and at least two edges directed from W to U (see [1, 27]). So u and w are 2-edge-connected in G' by Menger's theorem. □

As a special case of applying Test2EDP to $K = K_E(G)$, we can immediately remove loops and parallel edges $(\kappa(u), \kappa(v))$ if K has more than two edges directed from $\kappa(u)$ to $\kappa(v)$. To obtain faster implementations, we solve the 2ECS problems in linear-time using edge-disjoint spanning trees [7, 31], as specified in Section 2.4. The edges of these spanning trees give a 2-approximate solution C' for 2ECS on C. Moreover, as in 2ECC, we can apply algorithms Test2EDP-2EC, Test2ECC-2EC and Hybrid-2EC on the sparse subgraph computed by DST-2EC. Then, these algorithms produce a 4-approximation for 2EC in $O(n^2)$ time. Furthermore, for these $O(n^2)$-time algorithms, we can improve the approximate solution C' for 2ECS on each 2-edge-connected subgraph C of G, by applying the two edge-disjoint paths test on the edges of C'. We incorporate all these ideas in all our implementations.

Theorem 4.2. *There is a polynomial-time 4-approximation algorithm for the 2EC. Moreover, if the 2-edge-connected subgraphs of the input digraph are known in advance, we can compute a 4-approximation for the 2EC problem in linear time.*

Proof. Suppose that we apply algorithm DST-2ECC on the 2ECS-condensed graph $K = K_E(G)$ of G to obtain a sparse subgraph K'. By the analysis in Theorem 3.2, we have that K' has less than $4(N + N')$ edges, where N is the number of vertices of K and N' is number of vertices in the nontrivial 2-edge-connected components of K. Also, let k be the total number of edges computed by the 2-approximation 2ECSS algorithm for all maximal 2-edge-connected subgraphs of G. Let k^* be the total number of edges in the optimal solutions for all these 2ECSS problems. Then $k \leq 2k^*$. So, our algorithm computes a sparse certificate for G with less than $4(N + N') + k \leq 4(N + N') + 2k^* < 4(N + N' + k^*)$ edges. The smallest 2EC solution has at least $N + N' + k^*$ edges, so the approximation ratio of 4 follows. Finally, it is easy to verify that our algorithm runs in linear time if the 2-edge-connected subgraphs of G are known in advance. □

For completeness, we note that Theorem 3.2 implies that the Test2ECC algorithms also achieve a 4-approximation even when they are run on the original digraphs instead of the sparse certificates.

Corollary 4.3. *Algorithm Test2ECC (resp., Test2ECC-2EC) applied on the original input (resp., condensed) graph gives a 4-approximate solution for 2ECC (resp., 2EC).*

Proof. We consider first the algorithm Test2ECC for the 2ECC problem. Let G be a strongly connected digraph with n vertices, and let n' be the number of vertices in nontrivial components (i.e., 2-edge-connected components of size at least 2). Let G' be the spanning subgraph of G produced by running Test2ECC on G. It suffices to argue that G' contains less than $4(n+n')$ edges. Suppose that we run DST on G'. Let G'' be the resulting subgraph of G'. Then, G'' is also a solution to 2ECC for G, and by the proof of Theorem 3.2 it has at most $4(n + n')$ edges. But since G' is a minimal solution to 2ECC for G, we must have $G' = G''$.

For the 2EC problem, assume that the edge-disjoint spanning trees construction produces k edges. Then $k \leq 2k^*$, where k^* is the number of edges in an optimal solution. Let K be the condensed graph of G, and let N be the number of its vertices. Let K' be the spanning subgraph of K produced by running Test2ECC-2EC on K. By the proof of Theorem 4.2 and the same argument as for the 2ECC problem, we have that K' contains at most $4(N + N')$ edges, where N' is the total number of vertices in nontrivial components of K'. So the corresponding expanded graph has at most $4(N+N')+k < 4(N+N'+k^*)$ edges. Since the smallest 2EC solution has at least $N+N'+k^*$ edges, the 4-approximation follows. □

5 Approximation algorithms and heuristics for 2VC and 2C

Now we consider how to extend our algorithms in order to solve problems 2VC and 2C efficiently.

5.1 Approximation algorithms and heuristics for 2C

To get an approximate solution for problem 2C, we combine our algorithms for 2VCC with algorithms that approximate 2VCSS [5, 13]. We also take advantage of the fact that every 2-vertex-connected subgraph is contained in a 2-edge-connected subgraph [16]. This property suggests the following approach for 2C. First, we compute the 2-vertex-connected subgraphs of G and solve the 2VCSS problem independently for each such subgraph. Then, we apply one of the algorithms DLN or LHL for 2VCC on G. Since the sparse certificate from DLN or LHL also maintains the 2-edge-connected components, it remains to include edges that maintain the 2-edge-connected subgraphs of G. We can find these edges in the 2VCS-condensed graph $K = K_V(G)$ of G, where we contract each maximal 2-vertex-connected subgraph of G into a single supervertex. See Section 2.4. For any vertex v of G, we denote by $\kappa(v)$ the supervertex of K that contains v. Every edge $(\kappa(u), \kappa(v))$ of

K is associated with the corresponding original edge (u, v) of G. Now we describe the main steps of our algorithm for 2C:

1. Compute the maximal 2-vertex-connected subgraphs. Solve independently the 2VCSS problem for each such subgraph, using the linear-time algorithm of [13].

2. Form the 2VCS-condensed multigraph $K = K_V(G)$, and compute its 2-edge-connected subgraphs. Solve independently the 2ECSS problem for each such subgraph, using edge-disjoint spanning trees [31].

3. Execute the DLN or LHL algorithms on the original graph G and compute a sparse certificate for the 2-edge- and the 2-vertex-connected components.

The solution to the 2C problem consists of the edges selected in each step of the algorithm. Note that in Step 2, we should allow 2-edge-connected subgraphs of size two because such a subgraph may correspond to the union of 2-vertex-connected subgraphs of the original graph. We consider two versions of our algorithm, DLN-2C and LHL-2C, depending on the algorithm for the 2VCC problem used in Step 3.

Theorem 5.1. *Algorithms DLN-2C and LHL-2C compute a 6-approximation for problem 2C. Moreover, if the 2-edge- and the 2-vertex- connected subgraphs of G are available, then the algorithms run in linear time.*

Proof. Let n_v be the number of vertices of G that belong to some 2-vertex-connected subgraph of G. Also, let $\breve{n}$ be the number of vertices in K, and let $\breve{n}_e$ be the number of vertices of K that belong to some 2-edge-connected subgraph of K.

A result in [13] shows that, given a 2-vertex-connected digraph with ν vertices, we can compute in linear time a 2-vertex-connected spanning subgraph that has less than 6ν edges. Hence, by applying this algorithm to each maximal 2-vertex-connected subgraph of G in Step 1, we select less than $6n_c$ edges in total. Also, in Step 3, we apply one of the sparse certificates for 2CC that maintain both the 2-edge- and the 2-vertex-connected components of G. By Theorems 3.3 and 3.4, such a certificate contains at most $6(n-1)$ edges. Hence, Steps 1 and 3 select less than $6(n+n_c)$ edges in total.

It remains to consider the contribution of Step 2. For the 2ECSS problems, we can compute a 2-approximate solution in linear-time using edge-disjoint spanning trees [7, 31]. Let $\breve{C}$ be a 2-edge-connected subgraph of K. We select an arbitrary vertex $\kappa(v) \in \breve{C}$ as a root and compute two edge-disjoint spanning trees in the flow graph $\breve{C}(\kappa(v))$ and two edge-disjoint spanning trees in the reverse flow graph $\breve{C}^R(\kappa(v))$. Thus, we select less than $4\breve{n}_e$ edges. Hence, the subgraph computed by the algorithm has less than $6(n + n_c + \breve{n}_e)$ edges.

Now consider any solution to 2C. It has to include $2n_c + 2\breve{n}_e$ edges in order to maintain the 2-vertex and the maximal 2-edge-connected subgraphs of G. Moreover, since the resulting subgraph must be strongly connected, there must be at least one edge entering each of the $\breve{n} - \breve{n}_e$ vertices of K that do not belong in a 2-edge-connected subgraph of K. Thus, the optimal solution has at least $2n_c + \breve{n}_e + \breve{n}$ edges. Note that $\breve{n}_c + \breve{n} \geq n$, so the optimal solution has at least $n + n_c + \breve{n}_e$ edges and the approximation ratio of 6 follows.

Finally, we show that all three steps of the algorithms DLN-2C and LHL-2C run in linear time given the maximal 2-edge- and 2-vertex- connected subgraphs of G. This is immediate for Steps 1 and 3. In Step 2, we do not need to compute the 2-edge-connected subgraphs of K from scratch, but we can form them from the 2-edge-connected subgraphs of G using contractions. Let C be a 2-edge-connected subgraph of G. We contract each 2-vertex-connected subgraph of G contained

in C into a single supervertex. Then, the resulting digraph $\breve{C}$ is a 2-edge-connected subgraph of K. □

If we wish to improve the quality of the computed solution G', we can apply the Test2VDP filter, and the analogous 2-edge-disjoint paths filter Test2EDP, as follows. In Step 1, we run the Test2VDP filter for the edges computed by the linear-time algorithm of [13]. This produces a minimal solution for 2VCSS in each 2-vertex-connected subgraph of G. Similarly, in Step 2, we run the Test2EDP filter for the edges of the edge-disjoint spanning trees computed in each 2-edge-connected subgraph of K. This produces a minimal solution for 2ECSS in each 2-edge-connected subgraph of K. Finally, we run the Test2VDP filter on the whole G', but only consider the edges added in Step 3 of our algorithm, since the edges from Steps 1 and 2 are needed to maintain the 2-vertex- and the 2-edge-connected subgraphs. We implemented this algorithm, using DLN for Step 3, and refer to it as Test2VDP-2C.

5.2 Approximation algorithms and heuristics for 2VC

Executing Steps 1 and 3 of the algorithm of Section 5.1 is enough to produce a certificate for the 2VC problem. If we use DLN or LHL for Step 3, then we obtain a 6-approximate solution for 2VC. We call the corresponding algorithms DLN-2VC and LHL-2VC, respectively.

Theorem 5.2. *There is a polynomial-time algorithm for 2VC that achieves an approximation ratio of* 6*. Moreover, if the* 2*-vertex-connected subgraphs of* G *are available, then the algorithm runs in linear time.*

Proof. Let n_c be the number of vertices that belong in a maximal 2-vertex-connected subgraph of G. From the analysis in Theorem 5.1, we have that Steps 1 and 3 of the algorithm of Section 5.1 select less than $6(n + n_c)$ edges. One the other hand, any solution to 2VC has to include at least $2n_c$ edges for the maximal 2-vertex-connected subgraphs of G, and at least $n - n_c$ edges in order to obtain a strongly connected subgraph. Thus, the optimal solution has at least $n + n_c$ edges, so the approximation ratio of 6 follows. □

As in the 2VCC problem, we can improve the quality of the computed solution by applying the Test2VDP filter for the edges that connect different 2-vertex-connected subgraphs. We implemented this algorithm, using DLN, and refer to it as Test2VDP-2VC.

6 Experimental analysis

In order to assess the practical value of the algorithms previously described, we implemented them as summarized in Tables 1 and 2. All implementations were written in `C++` and compiled with `g++ v.4.4.7` with flag `-O3`. We performed our experiments on a GNU/Linux machine, with Red Hat Enterprise Server v6.6: a PowerEdge T420 server 64-bit NUMA with two Intel Xeon E5-2430 v2 processors and 16GB of RAM RDIMM memory. Each processor has 6 cores sharing a 15MB L3 cache, and each core has a 2MB private L2 cache and 2.50GHz speed. 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. All our running times were averaged over ten different runs.

For the experimental evaluation we use the datasets shown in Table 3. We note that one can compute an optimal solution of our datasets by using an ILP solver after formulating the problem appropriately [10]. However such an approach is hopeless to produce a solution in reasonable

Algorithm	Problem	Technique	Ratio	Ref.	Time
ZNI	2ECS	Zhao et al. [33] applied on the condensed graph	2	§2.4	$O(m+n)^\dagger$
DST-2ECC original	2ECC	Original sparse certificate from [16]	$O(1)$	§3.1.1	$O(m+n)$
DST-2ECC	2ECC	Improved sparse certificate	4	§3.1.2	$O(m+n)$
Test2EDP	2ECC	Two edge-disjoint paths test on sparse certificate of input graph	4	§3.4	$O(n^2)$
Test2ECC	2ECC	2-edge-connected components test on sparse certificate of input graph	4	§3.4	$O(n^2)$
Hybrid	2ECC	Hybrid of two edge-disjoint paths and 2-edge-connected components test on sparse certificate of input graph	4	§3.4	$O(n^2)$
Test2EDP-Aux	2ECC	Test2EDP applied on second-level auxiliary graphs	4	§3.5	$O(n^2)$
Hybrid-Aux	2ECC	Hybrid applied on second-level auxiliary graphs	4	§3.5	$O(n^2)$
DST-2EC	2EC	Improved sparse certificate preserving 2-edge-connected subgraphs (applied on condensed graph)	4	§4	$O(m+n)^\dagger$
Test2EDP-2EC	2EC	Two edge-disjoint paths test on sparse certificate of condensed graph	4	§4	$O(n^2)$
Test2ECC-2EC	2EC	2-edge-connected components test on sparse certificate of condensed graph	4	§4	$O(n^2)$
Hybrid-2EC	2EC	Hybrid of two edge-disjoint paths and 2-edge-connected components test on sparse certificate of condensed graph	4	§4	$O(n^2)$
Test2EDP-2EC-Aux	2EC	Test2EDP-2EC applied on second-level auxiliary graphs	4	§4	$O(n^2)$
Hybrid-2EC-Aux	2EC	Hybrid-2EC applied on second-level auxiliary graphs	4	§4	$O(n^2)$

Table 1: The algorithms considered in our experimental study for preserving 2-edge-connectivity relations. The worst-case bounds refer to a digraph with n vertices and m edges. Running times indicated by † assume that the maximal 2-edge-connected subgraphs of the input digraph are available.

time for large graphs. Instead we use an indirect measurement of the quality of the produced solutions as follows. We measure the quality of the solution computed by algorithm A on problem $\mathcal{P}$ by a *quality ratio* defined as $q(A,\mathcal{P}) = \delta^A_{avg}/\delta^{\mathcal{P}}_{avg}$, where δ^A_{avg} is the average vertex indegree of the spanning subgraph computed by A and $\delta^{\mathcal{P}}_{avg}$ is a lower bound on the average vertex indegree of the optimal solution for $\mathcal{P}$. Specifically, for 2ECC and 2EC (resp., 2VCC and 2VC) we define $\delta^{\mathsf{2ECC}}_{avg} = (n+k)/n$, (resp., $\delta^{\mathsf{2VCC}}_{avg}$) where n is the total number of vertices of the input digraph and k is the number of vertices that belong in nontrivial 2-edge-connected components (resp., 2-vertex-connected components). This follows from the fact that in the sparse subgraph the k vertices in nontrivial components must have indegree at least two, while the remaining $n-k$ vertices must have indegree at least one, since we seek for a strongly connected spanning subgraph. We use the lower bound $\delta^{\mathsf{2ECC}}_{avg}$ also for 2C, since every 2-vertex-connected subgraph or component is contained in a 2-edge-connected component. We set a similar lower bound $\delta^{\mathsf{2ECS}}_{avg} = (n+k+c)/n$ for 2ECS,

Algorithm	Problem	Technique	Ratio	Ref.	Time
DST-2VCC original	2VCC	Original sparse certificate from [15] based on divergent spanning trees	$O(1)$	§3.2.1	$O(m+n)$
DST-2VCC	2VCC	Improved sparse certificate	$O(1)$	§3.2.2	$O(m+n)$
DLN	2CC	Sparse certificate from [18] based on divergent spanning trees and loop-nesting trees	6	§3.3.1	$O(m+n)$
LHL	2CC	New sparse certificate based on low-high orders and loop-nesting trees	6	§3.3.2	$O(m+n)$
Test2VDP	2CC	Two vertex-disjoint paths test applied on the digraph produced by DLN	6	§3.4	$O(n^2)$
DLN-2VC	2VC	DLN combined with the linear-time 2VCSS algorithm of [13]	6	§5.2	$O(m+n)^\dagger$
LHL-2VC	2VC	LHL combined with the linear-time 2VCSS algorithm of [13]	6	§5.2	$O(m+n)^\dagger$
Test2VDP-2VC	2VC	Two vertex-disjoint paths test applied on the digraph produced by DLN-2VC	6	§5.2	$O(n^2)$
DLN-2C	2C	DLN-2VC combined with the linear-time 2ECSS algorithm using edge-disjoint spanning trees	6	§5.1	$O(m+n)^\ddagger$
LHL-2C	2C	LHL-2VC combined with the linear-time 2ECSS algorithm using edge-disjoint spanning trees	6	§5.1	$O(m+n)^\ddagger$
Test2VDP-2C	2C	Test2VDP and Test2EDP applied on the digraph produced by DLN-2C	6	§5.1	$O(n^2)$

Table 2: The algorithms considered in our experimental study for preserving 2-connectivity relations. The worst-case bounds refer to a digraph with n vertices and m edges. Running times indicated by † assume that the maximal 2-vertex-connected subgraphs of the input digraph are available; running times indicated by ‡ assume that both the 2-edge- and the 2-vertex-connected subgraphs are available.

Dataset	n	m	file size	δ_{avg}	b^*	a^*	$\delta_{avg}^{\mathsf{2ECC}}$	$\delta_{avg}^{\mathsf{2ECS}}$	$\delta_{avg}^{\mathsf{2VCC}}$	type
Rome99	3353	8859	100KB	2.64	1474	789	1.76	1.68	1.76	road network
P2p-Gnutella25	5153	17695	203KB	3.43	2181	1840	1.60	1.00	1.60	peer2peer
P2p-Gnutella31	14149	50916	621KB	3.60	6673	5357	1.56	1.00	1.56	peer2peer
Web-NotreDame	53968	296228	3.9MB	5.49	34879	9629	1.50	1.38	1.50	web graph
Soc-Epinions1	32223	443506	5.3MB	13.76	20975	8194	1.56	1.55	1.56	social network
USA-road-NY	264346	733846	11MB	2.78	104618	46476	1.80	1.80	1.80	road network
USA-road-BAY	321270	800172	12MB	2.49	196474	84627	1.69	1.70	1.69	road network
USA-road-COL	435666	1057066	16MB	2.43	276602	120142	1.68	1.68	1.68	road network
Amazon0302	241761	1131217	16MB	4.68	73361	69616	1.74	1.69	1.74	prod. co-purchase
WikiTalk	111881	1477893	18MB	13.21	85503	14801	1.45	1.44	1.45	social network
Web-Stanford	150532	1576314	22MB	10.47	64723	14801	1.62	1.34	1.58	web graph
Amazon0601	395234	3301092	49MB	8.35	83995	69387	1.82	1.82	1.82	prod. co-purchase
Web-Google	434818	3419124	50MB	7.86	211544	89838	1.59	1.49	1.58	web graph
Web-Berkstan	334857	4523232	68MB	13.51	164779	53666	1.56	1.40	1.51	web graph

Table 3: Real-world graphs sorted by file size of their largest SCC; n is the number of vertices, m the number of edges, and δ_{avg} is the average vertex indegree; b^* is the number of strong bridges; a^* is the number of strong articulation points; $\delta_{avg}^{\mathsf{2ECC}}$, $\delta_{avg}^{\mathsf{2ECS}}$, and $\delta_{avg}^{\mathsf{2VCC}}$ are lower bounds on the average vertex indegree of an optimal solution to 2ECC, 2ECS, and 2VCC, respectively.

where k is the number of vertices that belong in nontrivial 2-edge-connected subgraphs, and c is the number of nontrivial 2-edge-connected subgraphs (the 2-connected subgraphs need to be strongly connected to the rest of the graph). Note that the quality ratio is an upper bound of the actual approximation ratio of the specific input. The smaller the values of $q(A, \mathcal{P})$ (i.e., the closer to 1), the better is the approximation obtained by algorithm A for problem $\mathcal{P}$.

6.1 Experimental results for 2-edge-connectivity

We now report the results of our experiments with all the algorithms considered for problems 2ECS, 2ECC and 2EC. As previously mentioned, for the sake of efficiency, all variants of Test2EDP, Test2ECC and Hybrid were run on the sparse certificate computed by either DST-2ECC or DST-2EC (depending on the problem at hand) instead of the original digraph.

We group the experimental results into two categories: results on the 2ECC problem and results on both 2ECS and 2EC problems. In all cases we are interested in the quality ratio of the computed solutions and the corresponding running times. Specifically, we report the following experimental results:

- For the 2ECC problem:
  - the quality ratios of the spanning subgraphs computed by the different algorithms are shown in Table 4 and Figure 12 (top);
  - their running times are given in Table 6, while the corresponding plotted values are shown in Figure 13 (top).
- For the 2ECS and 2EC problems:
  - the quality ratios of the spanning subgraphs computed by the different algorithms are shown in Table 5 and Figure 12 (bottom);
  - their running times are given in Table 7, while the corresponding plotted values are shown in Figure 13 (bottom). We note that the running times include the time to compute the 2-edge-connected subgraphs of the input digraph. To that end, we use the algorithm from [6], which is fast in practice despite the fact that its worst-case running time is $O(mn)$.

6.2 Experimental results for 2-vertex-connectivity and 2-connectivity

We now report the results of our experiments with all the algorithms considered for problems 2VCC and 2C.

- For the 2VCC problem:
  - the quality ratios of the spanning subgraphs computed by the different algorithms are shown in Table 8 and Figure 14 (top);
  - their running times are given in Table 10, while the corresponding plotted values are shown in Figure 15 (top).
- For the 2VC and 2C problems:
  - the quality ratios of the spanning subgraphs computed by the different algorithms are shown in Table 9 and Figure 14 (bottom);

Dataset	DST-2ECC original (%)	DST-2ECC (%)	Test2EDP (%)	Test2ECC & Hybrid (%)	Test2EDP-Aux (%)	Hybrid-Aux (%)
Rome99	1.389 (92.5)	1.363 (90.7)	1.171 (78.0)	1.167 (77.7)	1.177 (78.3)	1.174 (78.2)
P2p-Gnutella25	1.656 (77.4)	1.512 (70.7)	1.220 (57.0)	1.143 (53.4)	1.251 (58.5)	1.234 (57.6)
P2p-Gnutella31	1.682 (73.0)	1.541 (66.9)	1.251 (54.3)	1.169 (50.7)	1.291 (56.0)	1.274 (55.3)
Web-NotreDame	1.964 (53.7)	1.807 (49.4)	1.489 (40.7)	1.417 (38.7)	1.500 (41.0)	1.471 (40.2)
Soc-Epinions1	2.047 (23.3)	1.837 (20.9)	1.435 (16.3)	1.379 (15.7)	1.441 (16.4)	1.406 (16.0)
USA-road-NY	1.343 (87.3)	1.245 (80.9)	1.174 (76.3)	1.174 (76.3)	1.175 (76.3)	1.175 (76.3)
USA-road-BAY	1.361 (92.7)	1.307 (89.0)	1.245 (84.9)	1.246 (84.9)	1.246 (84.9)	1.246 (84.9)
USA-road-COL	1.354 (94.0)	1.304 (90.5)	1.251 (86.9)	1.252 (86.9)	1.252 (86.9)	1.252 (86.9)
Amazon0302	1.762 (65.5)	1.570 (58.3)	1.186 (44.1)	1.134 (42.2)	1.206 (44.9)	1.196 (44.5)
WikiTalk	2.181 (23.9)	2.050 (22.5)	1.788 (19.6)	1.588 (17.4)	1.792 (19.7)	1.615 (17.7)
Web-Stanford	1.907 (29.6)	1.688 (26.2)	1.409 (21.9)	1.365 (21.2)	1.418 (22.0)	1.406 (21.9)
Amazon0601	1.866 (40.8)	1.649 (36.1)	1.163 (25.4)	1.146 (25.1)	1.170 (25.6)	1.166 (25.5)
Web-Google	1.921 (38.8)	1.728 (34.9)	1.389 (28.1)	1.322 (26.7)	1.401 (28.3)	1.377 (27.8)
Web-Berkstan	2.048 (23.7)	1.775 (20.5)	1.480 (17.1)	1.427 (16.5)	1.489 (17.2)	1.469 (17.0)

Table 4: Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2ECC. Within the parentheses we report the size (number of edges) of the computed solutions as a percentage of the size of the original input graphs.

Dataset	ZNI (%)	DST-2EC (%)	Test2EDP-2EC (%)	Test2ECC-2EC & Hybrid-2EC (%)	Test2EDP-2EC-Aux (%)	Hybrid-2EC-Aux (%)
Rome99	1.360 (90.5)	1.371 (91.3)	1.197 (79.7)	1.187 (79.0)	1.197 (79.7)	1.195 (79.6)
P2p-Gnutella25	1.276 (59.6)	1.517 (70.9)	1.218 (56.9)	1.141 (53.3)	1.249 (58.4)	1.232 (57.6)
P2p-Gnutella31	1.312 (57.0)	1.537 (66.7)	1.251 (54.3)	1.170 (50.8)	1.290 (56.0)	1.273 (55.3)
Web-NotreDame	1.620 (44.3)	1.747 (47.8)	1.500 (41.0)	1.426 (39.0)	1.510 (41.3)	1.484 (40.6)
Soc-Epinions1	1.790 (20.4)	1.847 (21.0)	1.488 (16.9)	1.435 (16.3)	1.489 (17.0)	1.476 (16.8)
USA-road-NY	1.343 (87.2)	1.341 (87.1)	1.163 (75.5)	1.163 (75.5)	1.163 (75.5)	1.163 (75.5)
USA-road-BAY	1.360 (92.7)	1.357 (92.5)	1.237 (84.3)	1.237 (84.3)	1.237 (84.3)	1.237 (84.3)
USA-road-COL	1.343 (93.2)	1.339 (93.0)	1.242 (86.2)	1.242 (86.2)	1.242 (86.2)	1.242 (86.2)
Amazon0302	1.464 (54.5)	1.580 (58.8)	1.279 (47.6)	1.228 (45.7)	1.292 (48.1)	1.284 (47.8)
WikiTalk	1.891 (20.8)	2.099 (23.0)	1.837 (20.2)	1.630 (17.9)	1.838 (20.2)	1.827 (20.1)
Web-Stanford	1.560 (24.2)	1.679 (26.1)	1.430 (22.2)	1.390 (21.6)	1.436 (22.3)	1.427 (22.1)
Amazon0601	1.709 (37.4)	1.727 (37.8)	1.200 (26.3)	1.186 (26.0)	1.202 (26.3)	1.200 (26.3)
Web-Google	1.637 (33.1)	1.728 (35.0)	1.437 (29.1)	1.381 (27.9)	1.446 (29.2)	1.431 (28.9)
Web-Berkstan	1.637 (18.9)	1.753 (20.3)	1.516 (17.5)	1.472 (17.0)	1.523 (17.6)	1.511 (17.5)

Table 5: Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2ECS and 2EC. Within the parentheses we report the size (number of edges) of the computed solutions as a percentage of the size of the original input graphs.

– their running times are given in Table 11, while the corresponding plotted values are shown in Figure 15 (bottom).

As in the 2ECS and 2EC problems, the running times for 2VC include the time to compute the maximal 2-vertex-connected subgraphs of the input digraph, and the running times for 2C include the time to compute both the maximal 2-edge-connected subgraphs and the maximal 2-vertex-connected subgraphs. (Again, the computation of those maximal subgraphs is performed by the algorithms of [6].)

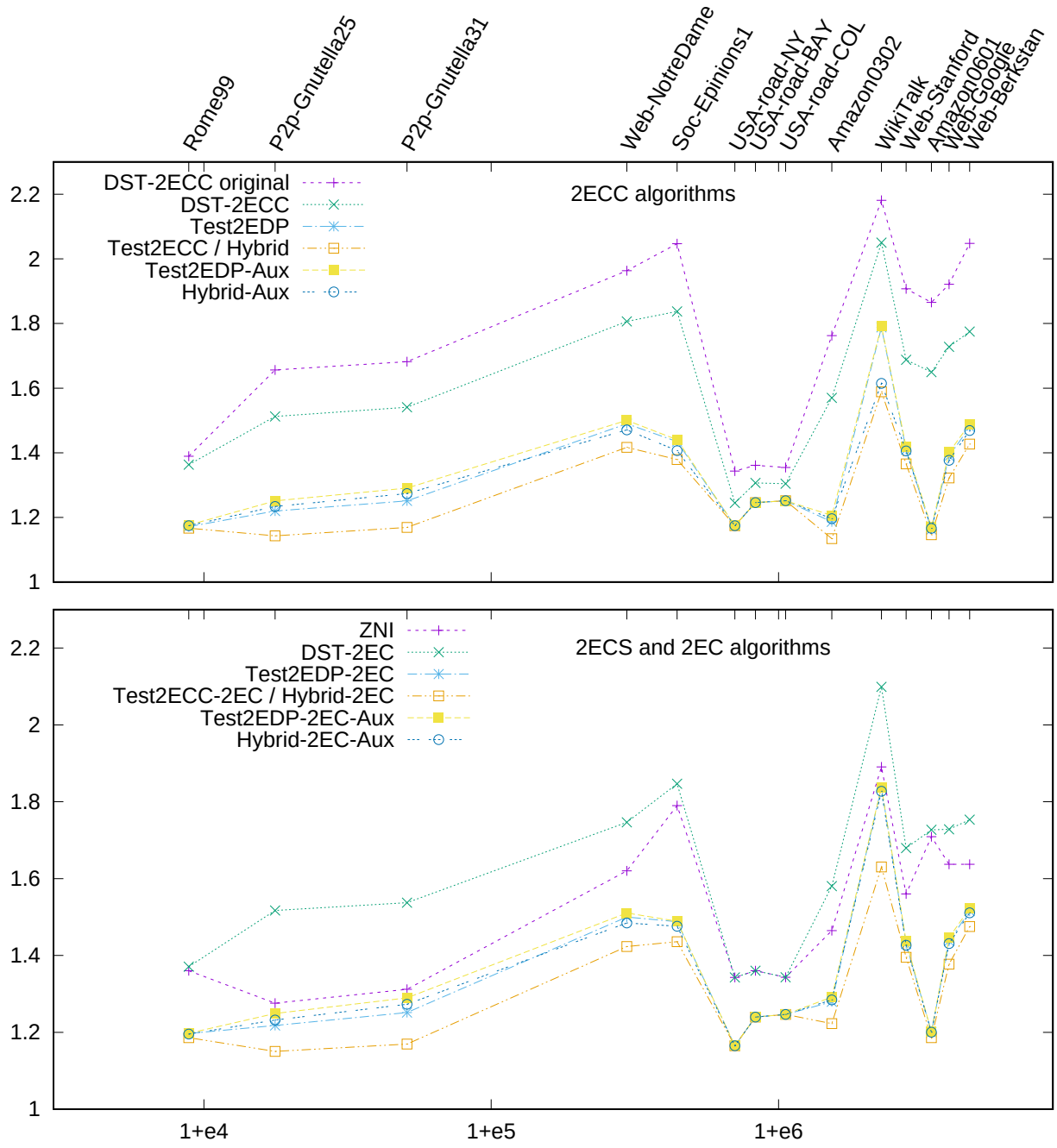


Figure 12: The plotted quality ratios taken from Tables 4 and 5, respectively.

6.3 Evaluation of the experimental results for 2EC

From the analysis of our experimental data, there are two peculiarities related to road networks that emerge immediately. First, all algorithms achieve consistently better approximations for road networks than for most of the other graphs in our data set. Second, for the 2ECC problem the Hybrid algorithms (Hybrid-2EC and Hybrid-2EC-Aux) seem to achieve substantial speedups on road networks; for the 2EC problem, this is even true for Test2ECC-2EC. The first phenomenon can be explained by taking into account the macroscopic structure of road networks, which is rather

Dataset	DST-2ECC original	DST-2ECC	Test2EDP	Test2ECC	Hybrid	Test2EDP-Aux	Hybrid-Aux
Rome99	0.008	0.010	0.160	14.297	0.224	0.056	0.183
P2p-Gnutella25	0.017	0.019	0.848	44.377	3.767	0.295	2.595
P2p-Gnutella31	0.052	0.064	6.716	352.871	31.935	2.467	20.923
Web-NotreDame	0.211	0.281	46.937	4723.904	352.834	3.192	215.492
Soc-Epinions1	0.194	0.224	47.869	2073.662	135.098	16.387	234.066
USA-road-NY	0.648	0.788	750.874	81990.402	206.055	110.616	108.463
USA-road-BAY	0.979	1.212	1002.689	132171.251	475.378	186.816	187.277
USA-road-COL	1.333	1.681	1794.103	231785.495	976.019	217.215	214.586
Amazon0302	1.068	1.253	1398.438	164047.057	8499.349	331.569	3985.706
WikiTalk	0.763	0.918	637.879	28339.485	5057.806	91.674	10877.771
Web-Stanford	0.908	1.309	607.356	49532.517	2120.636	25.184	952.585
Amazon0601	2.406	2.698	4847.592	446475.698	8408.463	968.964	8382.981
Web-Google	3.362	3.898	4801.787	612329.017	38031.588	422.058	25899.907
Web-Berkstan	1.829	3.841	2180.488	212587.201	10805.487	96.372	5641.406

Table 6: Running times in seconds of the algorithms for 2ECC.

Dataset	ZNI	DST-2EC	Test2EDP-2EC	Test2ECC-2EC	Hybrid-2EC	Test2EDP-Aux-2EC	Hybrid-Aux-2EC
Rome99	0.012	0.019	0.051	1.013	0.126	0.054	0.154
P2p-Gnutella25	0.010	0.029	0.855	77.274	3.727	0.320	2.574
P2p-Gnutella31	0.025	0.090	6.438	664.936	31.348	2.495	20.644
Web-NotreDame	0.159	0.448	11.062	2635.104	267.482	2.036	165.532
Soc-Epinions1	0.177	0.442	10.778	203.688	61.531	10.022	36.404
USA-road-NY	0.339	2.000	208.987	244.003	214.563	209.334	209.309
USA-road-BAY	0.437	4.539	151.786	289.465	178.197	152.488	152.407
USA-road-COL	0.547	5.275	198.795	526.362	305.525	199.768	199.711
Amazon0302	1.687	3.671	237.584	38201.229	3184.360	148.871	1909.122
WikiTalk	0.923	6.182	131.766	3538.042	2620.733	66.261	407.962
Web-Stanford	1.290	2.499	226.669	50153.480	1250.210	20.134	636.641
Amazon0601	4.768	7.659	1732.197	13067.429	2791.030	1725.333	2390.567
Web-Google	6.275	18.988	892.954	204990.718	15783.304	345.384	11714.605
Web-Berkstan	1.911	9.744	456.082	186129.463	5792.903	70.600	2552.911

Table 7: Running times in seconds of the algorithms for 2ECS and 2EC.

different from other networks. Indeed, road networks are very close to be "undirected": i.e., whenever there is an edge (x, y), there is also the reverse edge (y, x) (except for one-way roads). Roughly speaking, road networks mainly consist of the union of 2-edge-connected subgraphs, joined together by strong bridges, and their 2-edge-connected components coincide with their 2-edge-connected subgraphs. In this setting, a sparse strongly connected subgraph of the condensed graph will preserve both components and subgraphs. The second phenomenon is mainly due to the *trivial edge* heuristic described in Section 3.4.

Apart from the peculiarities of road networks, ZNI behaves as expected for 2ECS through its linear-time 2-approximation algorithm. Note that for both problems 2ECC and 2EC, all algorithms achieve quality ratio significantly smaller than our theoretical bound of 4. Regarding running times, we observe that the 2EC algorithms are faster than the 2ECC algorithms, sometimes significantly, as they take advantage of the condensed graph that seems to admit small size in real-world applications. In addition, our experiments highlight interesting tradeoffs between practical performance and quality of the obtained solutions. Indeed, the fastest (DST-2ECC and DST–2ECC original for problem 2ECC; DST-2EC for 2EC) and the slowest algorithms (Test2ECC and Hybrid for 2ECC; Test2ECC-2EC and Hybrid-2EC for 2EC) tend to produce respectively the worst and the best approximations. Note that DST-2ECC improves the quality of the solution of DST-2ECC original at the price of slightly higher running times, while Hybrid (resp., Hybrid-2EC) produces the same solutions

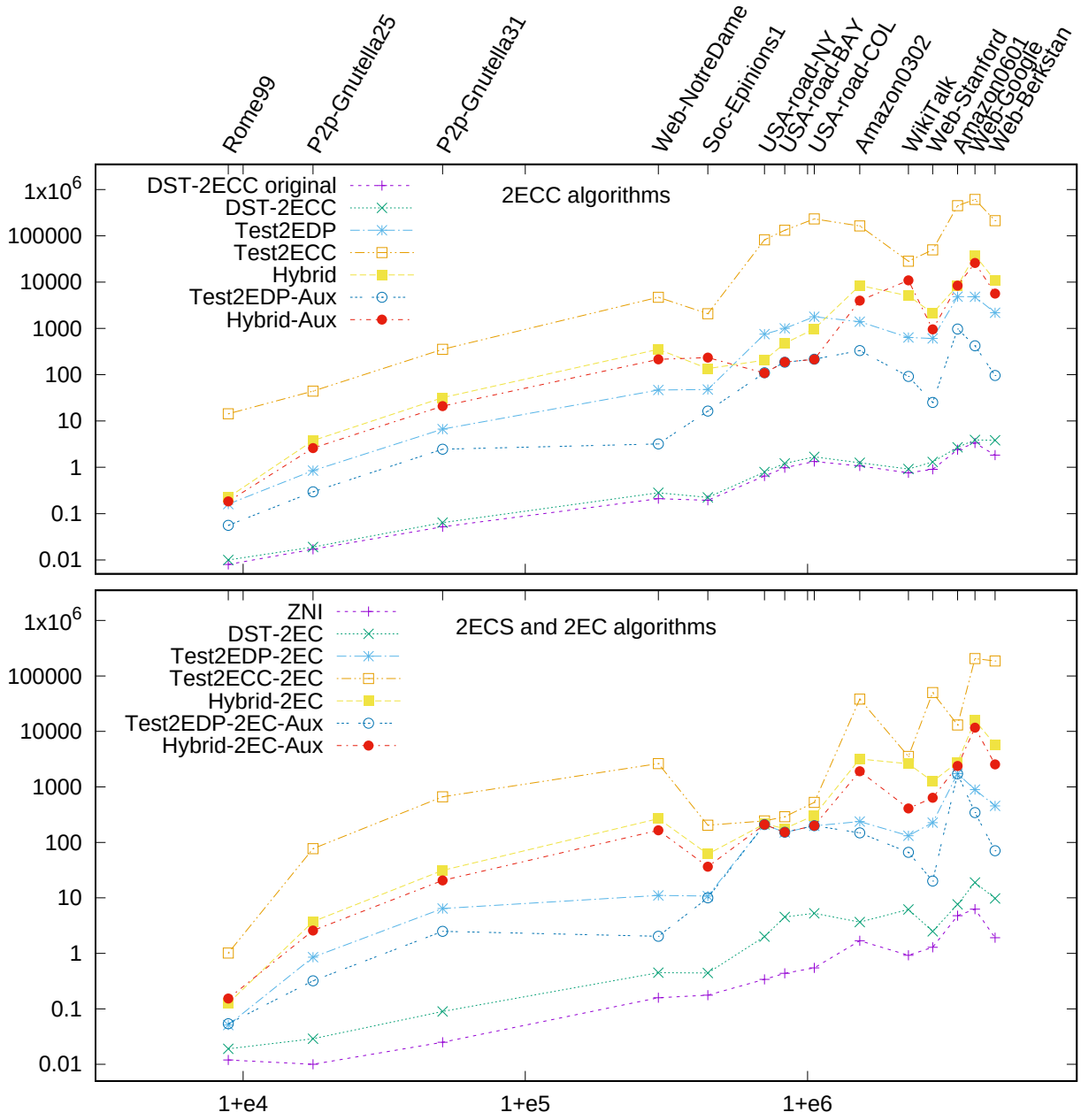


Figure 13: Running times in seconds with respect to the number of edges (in log-log scale) taken by Tables 6 and 7 for 2ECC, 2ECS, and 2EC.

as Test2ECC (resp., Test2ECC-2EC) with rather impressive speedups. Running an algorithm on the second-level auxiliary graphs seems to produce substantial performance benefits at the price of a slightly worse approximation (Test2EDP-Aux, Hybrid-Aux, Test2EDP-2EC-Aux and Hybrid-2EC-Aux versus Test2EDP, Hybrid, Test2EDP-2EC and Hybrid-2EC). Overall, in our experiments Test2EDP-Aux and Test2EDP-2EC-Aux seem to provide good quality solutions for the problems considered without being penalized too much by a substantial performance degradation.

Dataset	DST-2VCC original (%)	DST-2VCC (%)	DLN (%)	LHL (%)	2VDP (%)
Rome99	1.384 (92.1)	1.363 (90.7)	1.432 (95.3)	1.347 (89.6)	1.170 (77.8)
P2p-Gnutella25	1.726 (80.7)	1.602 (74.9)	1.713 (80.0)	1.467 (68.6)	1.234 (57.7)
P2p-Gnutella31	1.717 (74.5)	1.647 (71.5)	1.732 (75.2)	1.506 (65.3)	1.273 (55.2)
Web-NotreDame	2.072 (56.6)	2.067 (56.5)	2.108 (57.6)	1.769 (48.3)	1.588 (43.4)
Soc-Epinions1	2.082 (23.7)	1.964 (22.3)	2.213 (25.2)	1.755 (20.0)	1.475 (16.8)
USA-road-NY	1.255 (81.5)	1.251 (81.3)	1.371 (89.1)	1.332 (86.6)	1.168 (75.9)
USA-road-BAY	1.315 (89.6)	1.311 (89.3)	1.374 (93.7)	1.350 (92.0)	1.242 (84.6)
USA-road-COL	1.308 (90.8)	1.307 (90.8)	1.354 (94.0)	1.336 (92.8)	1.249 (86.7)
Amazon0302	1.918 (71.3)	1.791 (66.7)	1.849 (68.8)	1.548 (57.6)	1.245 (46.3)
WikiTalk	2.145 (23.5)	2.126 (23.3)	2.281 (25.0)	1.991 (21.9)	1.796 (19.7)
Web-Stanford	2.115 (31.9)	2.019 (30.5)	2.130 (32.2)	1.764 (26.6)	1.572 (23.7)
Amazon0601	1.926 (42.2)	1.793 (39.2)	1.959 (42.9)	1.576 (34.5)	1.196 (26.2)
Web-Google	2.052 (41.5)	2.004 (40.5)	2.083 (42.1)	1.746 (35.3)	1.485 (30.0)
Web-Berkstan	2.302 (25.8)	2.233 (25.0)	2.290 (25.7)	1.934 (21.7)	1.692 (19.0)

Table 8: Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2VCC. Within the parentheses we report the size (number of edges) of the computed solutions as a percentage of the size of the original input graphs.

Dataset	DLN-2VC (%)	LHL-2VC (%)	Test2VDP-2VC (%)	DLN-2C (%)	LHL-2C (%)	Test2VDP-2C (%)
Rome99	1.462 (97.3)	1.459 (97.1)	1.199 (79.8)	1.462 (97.3)	1.459 (97.1)	1.198 (79.8)
P2p-Gnutella25	1.712 (80.0)	1.560 (72.9)	1.234 (57.7)	1.712 (80.0)	1.560 (72.9)	1.234 (57.7)
P2p-Gnutella31	1.732 (75.2)	1.578 (68.5)	1.273 (55.3)	1.732 (75.2)	1.578 (68.5)	1.273 (55.3)
Web-NotreDame	2.232 (61.6)	1.995 (55.5)	1.628 (44.9)	2.250 (61.0)	2.028 (54.5)	1.638 (44.5)
Soc-Epinions1	2.474 (28.2)	2.346 (26.8)	1.572 (17.9)	2.474 (28.1)	2.351 (26.7)	1.573 (17.9)
USA-road-NY	1.376 (89.4)	1.396 (90.7)	1.175 (76.3)	1.376 (89.4)	1.396 (90.7)	1.175 (76.3)
USA-road-BAY	1.375 (93.7)	1.392 (94.8)	1.246 (84.9)	1.375 (93.7)	1.392 (94.8)	1.246 (84.9)
USA-road-COL	1.357 (94.2)	1.369 (95.0)	1.252 (86.9)	1.357 (94.2)	1.369 (95.0)	1.252 (86.9)
Amazon0302	2.020 (75.6)	1.904 (71.4)	1.386 (52.1)	2.032 (75.1)	1.920 (70.8)	1.399 (51.5)
WikiTalk	2.454 (26.9)	2.385 (26.1)	1.863 (20.4)	2.454 (26.9)	2.385 (26.2)	1.863 (20.5)
Web-Stanford	2.287 (34.7)	2.042 (31.0)	1.622 (24.6)	2.238 (34.5)	1.997 (30.8)	1.584 (24.5)
Amazon0601	2.241 (49.0)	2.174 (47.6)	1.278 (28.0)	2.242 (49.0)	2.176 (47.6)	1.279 (28.0)
Web-Google	2.306 (47.3)	2.144 (44.2)	1.585 (32.4)	2.338 (46.6)	2.183 (43.3)	1.602 (32.0)
Web-Berkstan	2.472 (27.9)	2.427 (25.7)	1.767 (19.9)	2.410 (27.7)	2.219 (26.1)	1.717 (19.8)

Table 9: Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2VC and 2C. Within the parentheses we report the size (number of edges) of the computed solutions as a percentage of the size of the original input graphs.

6.4 Evaluation of the experimental results for 2VC

We observe that all our algorithms perform well in terms of the quality of the solution they compute. Indeed, in contrast with the theoretical predictions (i.e., an approximation ratio of 6 in the worst case) the quality ratio is less than 2.5 for all algorithms and inputs. Our improved version of DST-2VCC performs consistently better than the original version. Also in all cases, LHL computed a higher quality solution than DLN. For most inputs, DST-2VCC computes a sparser graph than LHL, which is somewhat surprising given the fact that we do not have a good theoretical bound for the (constant) approximation ratio of DST-2VCC. On the other hand, LHL is faster than DST-2VCC

Dataset	DST-2VCC original	DST-2VCC	DLN	LHL	Test2VDP
Rome99	0.014	0.018	0.004	0.005	0.264
P2p-Gnutella25	0.027	0.032	0.008	0.007	1.587
P2p-Gnutella31	0.070	0.094	0.024	0.027	13.325
Web-NotreDame	0.335	0.486	0.059	0.080	97.355
Soc-Epinions1	0.258	0.309	0.089	0.110	92.812
USA-road-NY	1.095	1.402	0.261	0.360	2546.484
USA-road-BAY	1.659	2.152	0.316	0.435	4089.389
USA-road-COL	2.439	3.050	0.438	0.603	7739.256
Amazon0302	2.101	2.410	0.517	0.675	3503.910
WikiTalk	1.777	2.125	0.355	0.473	1158.855
Web-Stanford	1.756	2.395	0.429	0.564	1174.984
Amazon0601	3.532	3.924	1.363	1.605	15349.126
Web-Google	4.837	5.467	1.533	1.968	26299.714
Web-Berkstan	3.239	5.261	0.690	0.869	6301.410

Table 10: Running times in seconds of the algorithms for 2VCC.

Dataset	DLN-2VC	LHL-2VC	Test2VDP-2VC	DLN-2C	LHL-2C	Test2VDP-2C
Rome99	0.032	0.034	0.122	0.034	0.036	0.122
P2p-Gnutella25	0.042	0.042	0.729	0.051	0.053	0.725
P2p-Gnutella31	0.119	0.119	5.613	0.143	0.149	5.422
Web-NotreDame	0.491	0.521	27.091	0.573	0.600	27.746
Soc-Epinions1	0.606	0.621	54.559	0.602	0.664	54.548
USA-road-NY	2.227	2.337	991.092	2.153	2.415	995.913
USA-road-BAY	2.153	2.298	1429.443	2.296	2.476	1447.318
USA-road-COL	3.770	3.969	3093.258	3.938	4.228	3064.297
Amazon0302	4.708	5.017	2244.856	5.135	5.509	2094.263
WikiTalk	2.179	2.133	943.690	2.203	2.513	924.810
Web-Stanford	2.037	2.313	279.236	2.561	2.487	317.115
Amazon0601	9.793	10.038	8065.680	11.669	11.397	8696.212
Web-Google	9.789	10.172	5095.600	11.535	12.979	5128.337
Web-Berkstan	4.670	4.872	1595.033	5.178	5.601	1546.041

Table 11: Running times in seconds of the algorithms for 2VC and 2C.

by a factor of 4.15 on average and has the additional benefit of maintaining both the 2-vertex and the 2-edge-connected components. The Test2VDP filter provides substantial improvements of the solution, since all algorithms that apply this heuristic have consistently better quality ratios (1.38 on average and always less than 1.87). However, this is paid with much higher running times, as those algorithms can be even 5 orders of magnitude slower than the other algorithms.

In addition, our experiments highlight interesting tradeoffs between practical performance and quality of the obtained solutions. In particular, the fastest algorithms for the 2VCC problem are the ones based on loop-nesting trees (DLN and LHL), with LHL achieving consistently better solutions than DLN.

Once again, all algorithms achieve consistently better approximations for road networks than for most of the other graphs in our data set. The macroscopic structure of road networks has been explained earlier in the 2EC evaluation which also applies here for 2VC. Notice that a gain on the solution for the road networks is balanced at the cost of their additional running time.

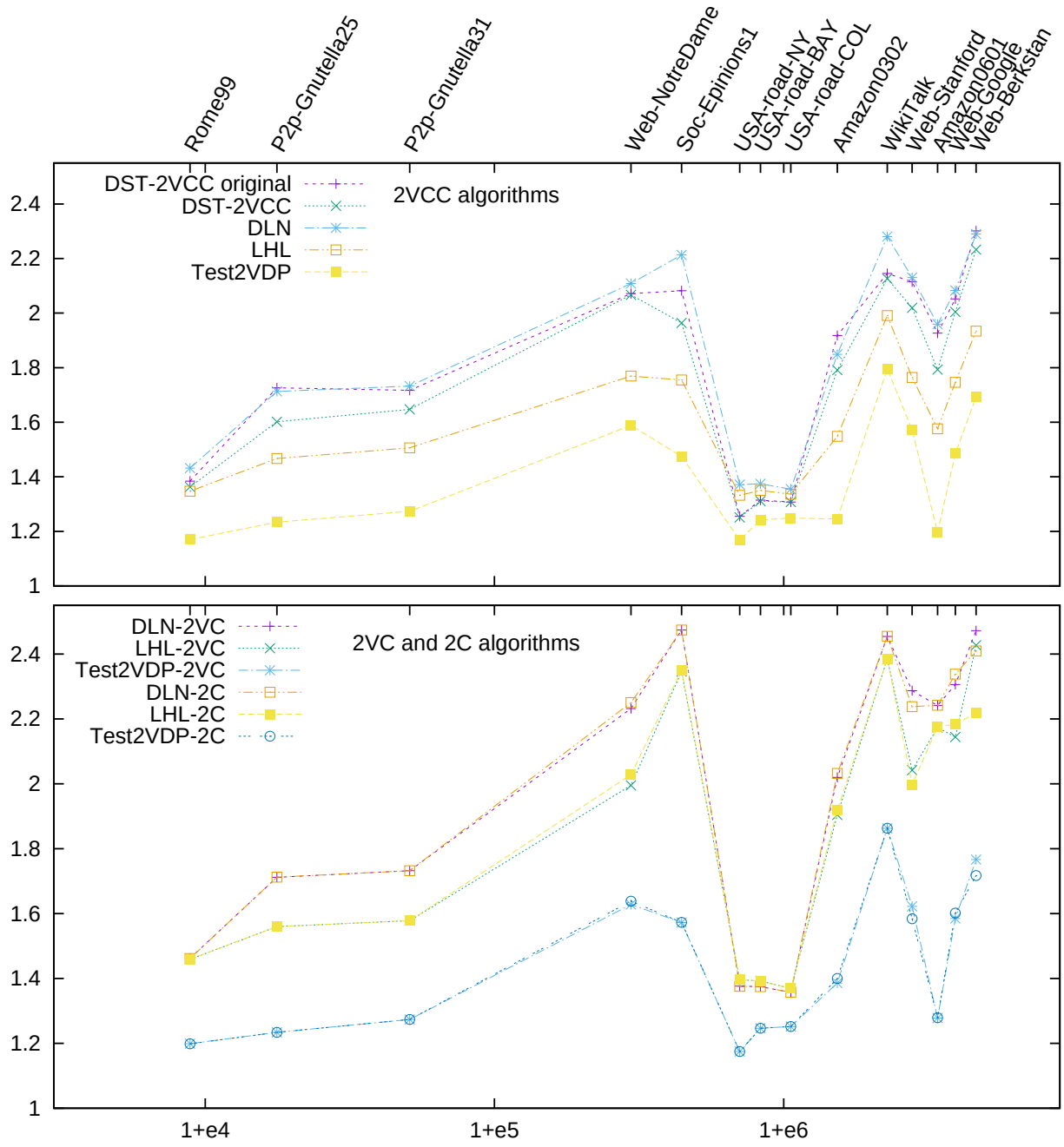


Figure 14: The plotted quality ratios taken by Tables 8 and 9.

6.5 Experiments with very large datasets

In order to assess the practicality of our algorithms for very large graphs, we also conducted experiments with the datasets shown in Table 12. We report only the results of our two linear-time algorithms, DLN and LHL for problem 2CC, which are more memory efficient than the algorithms of the DST family. Also, we note that running quadratic-time algorithms (based on the two edge-disjoint paths test or the 2-connected components test) for such large graphs is infeasible in our architecture.

The experimental results are given in Table 13. Again, we observe that both algorithms are

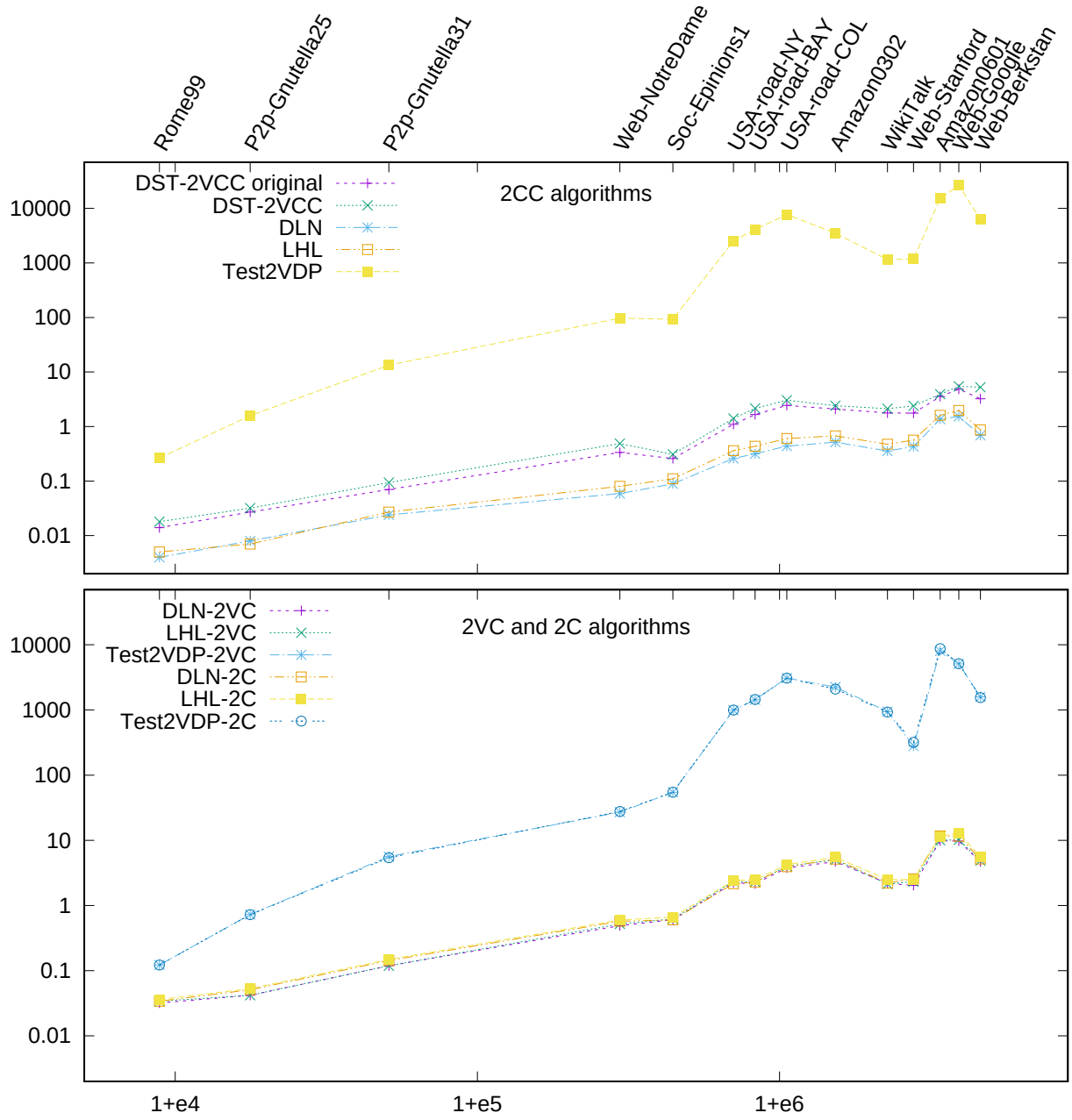


Figure 15: Running times in seconds with respect to the number of edges (in log-log scale) taken by Tables 10 and 11 for 2VCC, 2VC, and 2C.

able to compute good approximate solutions reasonably fast, with LHL performing consistently better with a slight increase in its running time. Notice that for all graphs the solutions that are computed by our algorithms reduce considerably their size. In real-world applications that deal with large-scale networks such a considerable reduction is highly appreciated.

Dataset	n	m	file size	δ_{avg}	b^*	a^*	$\delta_{avg}^{\mathsf{2ECC}}$
Flickr	1605184	30338513	539MB	18.90	1689976	311611	1.41
LiveJournal	4846609	65349587	1,2GB	13.48	1385512	649125	1.40
UK-2002	12090163	227480826	4,2GB	18.81	6334990	1885677	1.52
Webbase-2001	53891939	612895365	12,1GB	11.37	32153024	8636383	1.51
UK-2005	25711307	692875781	13,4GB	26.95	12645456	3254786	1.54
It-2004	29855421	925643838	18,8GB	31.00	15780944	4570241	1.53
Twitter	33479734	1394440635	27,7GB	41.65	7834054	3175909	1.79

Table 12: Real-world large graphs sorted by file size of their largest SCC.

Dataset	quality ratio		running times		size of solution	
	DLN ($q(A,\mathcal{P})$)	LHL ($q(A,\mathcal{P})$)	DLN (sec.)	LHL (sec.)	DLN (%)	LHL (%)
Flickr	2.00	1.78	4.48	5.31	14.97	13.32
LiveJournal	2.06	1.77	19.71	22.26	21.41	18.36
UK-2002	2.28	2.02	13.19	16.94	18.41	16.31
Webbase-2001	2.13	1.91	58.04	77.03	28.23	25.26
UK-2005	2.34	2.00	44.29	55.58	13.40	11.46
It-2004	2.27	1.98	48.17	54.20	11.19	9.75
Twitter	2.36	1.84	542.46	680.39	10.15	7.91

Table 13: The quality ratio and the running times for problem 2CC of algorithms DLN and LHL on the large dataset. The last two columns indicate the actual size of the computed solution with respect to the size of the original considered graph.

7 Concluding remarks

In this paper we have conducted both theoretical and experimental study to the 2ECC and 2VCC problems, and their generalizations 2EC, 2VC, and 2C. We have provided new practical algorithms that achieve good approximations with fast running times.

Our work raises some new and perhaps intriguing questions. As already mentioned earlier, we do not know if the approximation ratios of 4 or 6 that we provided are tight. In light of our experimental results, it seems possible that the Hybrid algorithms always achieve a 2-approximation, but we have no proof. Moreover the concept of 2-edge-connected components or 2-vertex-connected components may well be generalized to k-edge disjoint paths or k-vertex disjoint paths, for $k \geq 2$. Keeping in mind that the underlying graph should remain strongly connected, it is natural to ask whether computing the smallest such spanning subgraph is able to achieve a better approximation ratio for $k > 2$. Such a phenomenon occurs in approximating the smallest spanning k-edge or k-vertex connected subgraph [5, 10].

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] 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.

[4] 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 *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms*, (SODA 2017), pages 1900–1918, 2017.

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

[6] W. Di Luigi, L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-connectivity in directed graphs: An experimental study. In *Proc. 17th Algorithm Engineering and Experiments*, (ALENEX 2015), pages 173–187, 2015.

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

[8] 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 2008), pages 153–158, New York, NY, USA, 2008. ACM.

[9] L. R. Ford; D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[10] H. N. Gabow, M. X. Goemans, E. Tardos, and D. P. Williamson. Approximating the smallest k-edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009.

[11] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–21, 1985.

[12] 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.

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

[14] 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.

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

[16] L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-edge connectivity in directed graphs. *ACM Transactions on Algorithms*, 13(1):9:1–9:24, 2016. Announced at SODA 2015.

[17] 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*, (ESA 2015), pages 582–594, 2015.

[18] L. Georgiadis, G. F. Italiano, and N. Parotsidis. Strong connectivity in directed graphs under failures, with applications. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms*, (SODA 2017), pages 1880–1899, 2017.

[19] 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.

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

[21] G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theor. Comput. Sci.*, 447(0):74–84, 2012.

[22] R. Jaberi. Computing the 2-blocks of directed graphs. *RAIRO-Theor. Inf. Appl.*, 49(2):93–119, 2015.

[23] S. Khuller, B. Raghavachari, and N. E. Young. Approximating the minimum equivalent digraph. *SIAM Journal on Computing*, 24(4):859–872, 1995. Announced at *SODA 1994*, 177-186.

[24] S. Khuller, B. Raghavachari, and N. E. Young. On strongly connected digraphs with bounded cycle length. *Discrete Applied Mathematics*, 69(3):281–289, 1996.

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

[26] B. Laekhanukit, S. O. Gharan, and M. Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *Proc. 39th Int'l. Coll. on Automata, Languages, and Programming*, (ICALP 2012), pages 606–616, 2012.

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

[28] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

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

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

[31] R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–85, 1976.

[32] A. Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, (SODA 2001), pages 417–426, 2001.

[33] 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.