# Sparse Subgraphs for 2-Connectivity in Directed Graphs

Loukas Georgiadis[1], Giuseppe F. Italiano[2], Aikaterini Karanasiou[1], Charis Papadopoulos[1], and Nikos Parotsidis[2]

**Abstract.** Let $G$ be a strongly connected directed graph. We consider the problem of computing the smallest strongly connected spanning subgraph of $G$ that maintains the pairwise 2-vertex-connectivity of $G$, i.e., the 2-vertex-connected blocks of $G$ ($\mathsf{2VC\text{-}B}$). We provide linear-time approximation algorithms for this problem that achieve an approximation ratio of 6. Based on these algorithms, we show how to approximate, in linear time, within a factor of 6 the smallest strongly connected spanning subgraph of $G$ that maintains respectively: both the 2-vertex-connected blocks and the 2-vertex-connected components of $G$ ($\mathsf{2VC\text{-}B\text{-}C}$); all the 2-connectivity relations of $G$ ($\mathsf{2C}$), i.e., both the 2-vertex- and the 2-edge-connected components and blocks. 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. $G$ 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. A vertex (resp., an edge) of $G$ is a *strong articulation point* (resp., a *strong bridge*) if its removal increases the number of strongly connected components. A digraph $G$ is 2*-vertex-connected* if it has at least three vertices and no strong articulation points; $G$ is *2-edge-connected* if it has no strong bridges. The 2*-vertex-* (resp., *2-edge-*) *connected components* of $G$ are its maximal 2-vertex- (resp., 2-edge-) connected subgraphs. Let $v$ and $w$ be two distinct vertices: $v$ and $w$ are 2*-vertex-connected* (resp., 2*-edge-connected*), denoted by $v \leftrightarrow_{2\mathrm{v}} w$ (resp., $v \leftrightarrow_{2\mathrm{e}} w$), if there are two internally vertex-disjoint (resp., two edge-disjoint) directed paths from $v$ to $w$ and two internally vertex-disjoint (resp., two edge-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*-vertex-connected block* (resp., 2*-edge-connected block*) of a digraph $G = (V, E)$

[1] University of Ioannina, Greece. E-mail: `{loukas,akaranas,charis}@cs.uoi.gr`.

[2] Università di Roma "Tor Vergata", Italy. E-mail: `{giuseppe.italiano,nikos.parotsidis}@uniroma2.it`. Partially supported by MIUR under Project AMANDA.

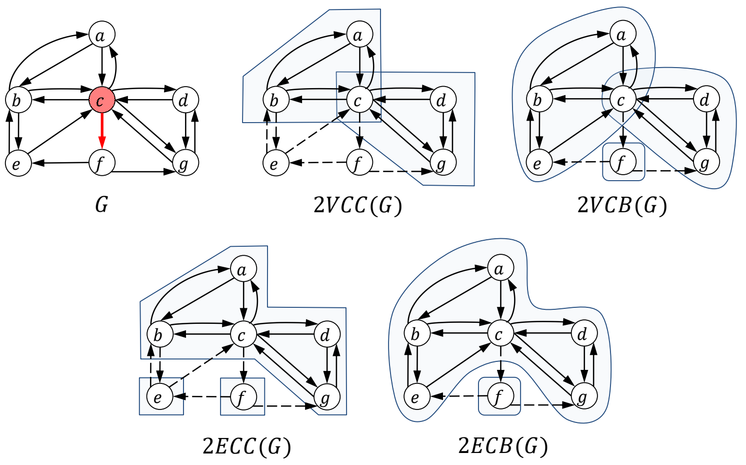

**Fig. 1.** A strongly connected digraph $G$ with a strong bridge $(c, f)$ and a strong articulation point $c$ shown in red (better viewed in color), the 2-vertex-connected components and blocks of $G$, and the 2-edge-connected components and blocks of $G$. Vertex $f$ forms a trivial 2-edge-connected and 2-vertex-connected block.

is a maximal subset $B \subseteq V$ such that $u \leftrightarrow_{2\mathrm{v}} v$ (resp., $u \leftrightarrow_{2\mathrm{e}} v$) for all $u, v \in B$. Note that, as a (degenerate) special case, a 2-vertex- (resp., 2-edge-) connected block might consist of a singleton vertex only: we denote this as a *trivial 2-vertex-* (resp., *2-edge-*) *connected block*. In the following, we will consider only non-trivial 2-vertex- and 2-edge- connected blocks. Since there is no danger of ambiguity, we will call them simply 2-vertex- and 2-edge-connected blocks.

Differently from undirected graphs, in digraphs 2-vertex and 2-edge 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 2-vertex- (resp., 2-edge-) connected blocks can be different from the 2-vertex- (resp., 2-edge-) connected components, i.e., two vertices may be 2-vertex- (resp., 2-edge-) connected but lie in different 2-vertex- (resp., 2-edge-) connected components (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 [25]. In the case of digraphs, however, it was shown only recently how to compute the 2-edge- and 2-vertex- connected blocks in linear time [11, 12], and the best current bound for computing the 2-edge- and the 2-vertex- connected components in digraphs is not even linear, but it is $O(n^2)$ [16].

In this paper we investigate problems where we wish to find a smallest spanning subgraph of $G$ (i.e., with minimum number of edges) that maintains certain 2-connectivity requirements in addition to strong connectivity. Problems of this nature are fundamental in network design, and have several practical applications [24]. Specifically, we consider computing a smallest strongly connected spanning subgraph of a digraph $G$ that maintains the following properties: the pairwise 2-vertex-connectivity of $G$, i.e., the 2-vertex-connected blocks of $G$ (2VC-B); the 2-vertex-connected components of $G$ (2VC-C); both the 2-vertex-connected blocks and components of $G$ (2VC-B-C). This complements our previous study of the edge-connectivity versions of these problems [13], that we refer to as 2EC-C (maintaining 2-edge-connected components), 2EC-B (maintaining 2-edge-connected blocks), and 2EC-B-C (maintaining 2-edge-connected blocks and components). Finally, we also consider computing a smallest spanning subgraph of $G$ that maintains all the 2-connectivity relations of $G$ (2C), that is, simultaneously the 2-vertex-connected and the 2-edge-connected components and blocks. Note that all these problems are NP-hard [9, 13], 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, since algorithms with higher running times might be practically infeasible on today's architectures.

**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) [9]. 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. [20], who provided a polynomial-time algorithm with an approximation guarantee of 1.64. This was improved to 1.61 by the same authors [21]. Later on, Vetta announced a further improvement to 3/2 [27], and Zhao et al. [28] 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. [23] gave a randomized $(1+1/k)$-approximation algorithm. For the smallest $k$-vertex-connected spanning subgraph (kVCSS), Cheriyan and Thurimella [4], 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 [10]. 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., [6], and the survey [22]. The previous results on kECSS and kVCSS immediately imply an approximation ratio smaller than 2 for 2EC-C and 2VC-C [13, 19]. While there has been substantial progress for 2EC-C and 2VC-C, problems 2EC-B and 2VC-B (i.e., computing sparse subgraphs with the same pairwise 2-edge or 2-vertex connectivity) seem substantially harder. Jaberi [18] was the first to consider several optimization problems related to 2EC-B and 2VC-B and proposed approximation algorithms. The approximation ratio in his algorithms, however, is linear in the number of strong bridges for 2EC-B and in the number of strong articulation points for 2VC-B, and hence $O(n)$ in the worst case. In [13], linear-time 4-approximation algorithms for 2EC-B and 2EC-B-C were presented. It seems thus natural to ask whether one can design linear-time algorithms which achieve small approximation guarantees for 2VC-B, 2VC-B-C and 2C.

**Our Results.** In this paper we address this question by presenting practical approximation algorithms for the 2VC-B, 2VC-B-C and 2C problems. We stress that the approach in this paper is substantially different from [13], since vertex connectivity is typically more involved than edge connectivity and requires several novel ideas and non-trivial techniques. In particular, differently from [13], our starting point in this paper is the recent framework for strong connectivity and 2-connectivity problems in digraphs [14], combined with the notions of *divergent spanning trees* and *low-high orders* [15] (defined below). Building on this new framework, we can obtain *sparse certificates* also for the 2-vertex-connected blocks. 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-vertex-connected blocks of $G$. We show that our constructions achieve a 6-approximation for 2VC-B in linear time. Then, we extend our algorithms so that they compute a 6-approximation for 2VC-B-C and 2C. These algorithms also run in linear time once the 2-vertex and the 2-edge-connected components of $G$ are available; if not, the current best running time for computing them is $O(n^2)$ [16]. Then we provide efficient implementations of these algorithms that run very fast in practice. We also present several heuristics that improve the quality (i.e., the number of edges) of the computed spanning subgraphs. Finally, we assess how all these algorithms perform in practical scenarios by conducting a thorough experimental study, and report its main findings.

## 2 Preliminaries

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 $u$; $u$ is a *proper dominator* of $w$ if $u$ dominates $w$ and $u \neq w$. 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 $D(s)$. 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. [17] 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 [26].

[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 $G$.

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 in linear-time by an algorithm of Tarjan [26], using the disjoint set union data structure of Gabow and Tarjan [8]. 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* $\delta$ *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 $v$ ($u <_\delta v$), $v$ is less than $w$ ($v <_\delta w$), and $w$ is not a descendant of $v$ in $D(s)$. 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 [15].

Let $T$ be a dfs tree of a digraph $G$ rooted at $s$. 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 *loop nesting tree* $L$ of a strongly connected digraph $G$ with respect to $T$, is the tree in which the parent of any vertex $v \neq s$ is the nearest proper ancestor $u$ of $v$ such that $v \in loop(u)$. The loop nesting tree can be computed in linear time [3, 26].

## 3 Approximation algorithms and heuristics for 2VC-B

Let $G = (V, E)$ be the input strongly connected digraph. In problem 2VC-B, we wish to compute a strongly connected spanning subgraph $G'$ of $G$ that has the same 2-vertex-connected blocks of $G$, with as few edges as possible. We consider the following approach. Start with the empty graph $G' = (V, \emptyset)$, and add as few edges as possible until $G'$ is guaranteed to have the same 2-vertex-connected blocks as $G$. We consider three linear-time algorithms that apply this approach. The first two are based on the sparse certificates for 2-vertex-connected blocks from [12, 14], which use divergent spanning trees. The third is a new algorithm that selects the edges of $G'$ with the help of low-high orders.

**Divergent Spanning Trees.** We can compute a sparse certificate $C(G)$ for the 2-vertex-connected blocks of a strongly connected digraph $G$ using the algorithm of [12], which is based on a linear-time construction of two divergent spanning trees of a flow graph [15]. We refer to this algorithm as DST-B. 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.

Algorithm DST-B 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. In the first phase, we insert into $C(G)$ the edges of two maximally edge-disjoint divergent spanning trees, $T_1(G(s))$ and $T_2(G(s))$ of $G(s)$. 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 $T_1(H^R(r))$ and $T_2(H^R(r))$ 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 $G$. 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 2VC-B. 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 [12].) We propose a modification of DST-B, that we call DST-B modified: For each auxiliary graph, we do not select in $C(G)$ the edges of its two divergent spanning trees that have only auxiliary descendants. Also, for every second-level auxiliary graph, during the computation of its strongly connected components we include the chosen edges that already form a strongly connected component.

More precisely, algorithm DST-B modified works as follows. In the first two phases, we try reuse as many edges as possible when we build the divergent spanning trees of $G(s)$ and of its auxiliary graphs. In the third phase of the construction we need to solve the smallest SCSS problem for each strongly connected component $S$ in the second-level auxiliary graphs $H_q$ after the deletion of the root vertex $q$. We do this by running a modified version of the linear-time 5/3-

approximation algorithm of Zhao et al. [28]. The algorithm of Zhao et al. a SCSS of a strongly connected graph by performing a depth-first search traversal of the input graph. 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 follows. The algorithms of [15] for computing two divergent spanning trees of a flow graph use 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 $C(G)$.

**Divergent Spanning Trees and Loop Nesting Trees.** An alternative linear-time algorithm to compute a sparse certificate $C(G)$ for the 2-vertex-connected blocks can be obtained via loop nesting trees, as described in [14]. As in algorithm DST-B, 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 a dfs tree of $G(s)$, and at most $n-1$ additional edges that are required to define the loops of $G(s)$. (See [15, 26] for the details.) Then, we repeat the same process in the reverse direction, i.e., for $G^R(s)$. As shown in [14], 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 blocks as $G$. We refer to this algorithm as DLN-B.

**Theorem 1.** *Algorithm DLN-B achieves an approximation ratio of* 6*, in linear time, for problem 2VC-B.*

*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 [15, 26], 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-B selects at most $6(n-1)$ edges. Since the resulting subgraph must be strongly connected, any valid solution to problem 2VC-B has at least $n$ edges, so DLN-B achieves a 6-approximation. By [15, 26], both the computation of a pair of divergent spanning trees and of a loop nesting tree can be done in linear time, hence DLN-B also runs in linear time. □

**Low-High Orders and Loop Nesting Trees.** Now we introduce a new linear-time construction of a sparse certificate, via low-high orders, that we refer to as LHL-B. 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-B. 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 $\delta$ be a low-high order on $G(s)$. Subgraph $C(G)$ satisfies the low-high order $\delta$ 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 $u \neq d(v)$.

**Theorem 2.** *Algorithm LHL-B is correct and achieves an approximation ratio of* 6 *for problem 2VC-B, in linear time.*

*Proof.* By construction, the sparse certificate $C(G)$ computed by LHL-B satisfies a low-high order $\delta$ of $G(s)$. This implies that $C(G)$ contains two divergent spanning trees $T_1$ and $T_2$ of $G(s)$ [15]. 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-B follows from the fact that DLN-B is correct.

Next we bound the approximation ratio of LHL-B. 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. $\square$

We note that both DLN-B and LHN-B also maintain the 2-edge-connected blocks of the input digraph. We use this fact in Section 4, where we compute a sparse subgraph that maintains all 2-connectivity relations. We can improve the solution computed by the above algorithms by using the following filter.

**Two Vertex-Disjoint Paths Test.** We test if $G' \setminus (x, y)$ contains two vertex-disjoint paths from $x$ to $y$. If this is the case, then we remove edge $(x, y)$; otherwise, we keep the edge $(x, y)$ in $G'$ and proceed with the next edge. For doing so, we define the 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 $(u, w)$ in $G'$ by $(u^+, v^-)$ in $G''$, so $v^-$ has the edges entering $v$ and $v^+$ has the edges leaving $v$. Now we can test if $G'$ still has two vertex-disjoint paths from $x$ to $y$ after deleting $(x, y)$ by running two iterations of the Ford-Fulkerson augmenting paths algorithm [7] for finding two edge-disjoint paths on $G''$ by treating $x^+$ as the source and $y^-$ as the sink. 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 $G''$. Since $G'$ has $O(n)$ edges, this test takes $O(n)$ time per edge, so the total running time is $O(n^2)$. We refer to this filter as 2VDP.

In our implementations we applied 2VDP on the outcome of DLN-B in order to assess our algorithms with a solution close to minimum. For the 2VC-B problem the algorithm obtained after applying such a filter is called 2VDP-B. In order to improve the running time of 2VDP in practice, we apply a speed-up heuristic for *trivial edges* $(x, y)$: if $x$ belongs to a 2-vertex-connected block and has outdegree two or $y$ belongs to a 2-vertex-connected block and has indegree two, then $(x, y)$ must be included in the solution.

## 4 Approximation algorithms and heuristics for 2C

To get an approximate solution for problem 2C, we combine our algorithms for 2VC-B with algorithms that approximate 2VCSS [4, 10]. We also take advantage of the fact that every 2-vertex-connected component is contained in a 2-edge-connected component. This property suggests the following approach for 2C. First, we compute the 2-vertex-connected components of $G$ and solve the 2VCSS problem independently for each such component. Then, we apply one of the algorithms DLN-B or LHL-B for 2VC-B on $G$. Since the sparse certificate from DLN-B or LHL-B also maintain the 2-edge-connected blocks, it remains to include edges that maintain the 2-edge-connected components of $G$. We can find these edges in a *condensed graph* $\breve{G}$ defined as follows. Digraph $\breve{G}$ is formed from $G$ by contracting each 2-vertex-connected component of $G$ into a single supervertex. Note that any two 2-vertex-connected components may have at most one vertex in common: if two such components share a vertex, they are contracted into the same supervertex. The resulting digraph $\breve{G}$ is a multigraph since the contractions can create loops and parallel edges. For any vertex $v$ of $G$, we denote by $\breve{v}$ the supervertex of $\breve{G}$ that contains $v$. Every edge $(\breve{u}, \breve{v})$ of $\breve{G}$ 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 2-vertex-connected components. Solve independently the 2VCSS problem for each such component, using the linear-time algorithm of [10].
2. Form the condensed multigraph $\breve{G}$, and compute its 2-edge-connected components. Solve independently the 2ECSS problem for each such component, using edge-disjoint spanning trees [13].
3. Execute the DNL-B or LHL-B algorithms on the original graph $G$ and compute a sparse certificate for the 2-edge- and the 2-vertex-connected blocks.

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 components of size two because such a component may correspond to the union of 2-vertex-connected components of the original graph. We consider two versions of our algorithm, DLN-2C and LHL-2C, depending on the algorithm for the 2VC-B problem used in Step 3.

**Theorem 3.** *Algorithms DLN-2C and LHL-2C compute a 6-approximation for problem 2C. Moreover, if the 2-edge- and the 2-vertex- connected components 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 component of $G$. Also, let $\breve{n}$ be the number of vertices in $\breve{G}$, and let $\breve{n}_e$ be the number of vertices of $\breve{G}$ that belong to some 2-edge-connected component of $\breve{G}$. By the analysis in the proof of Theorem 4, the algorithm for 2VC-B-C selects less than $6(n + n_v)$ edges. For the 2ECSS problems, we can compute a 2-approximate solution in linear-time as in [13], using edge-disjoint spanning trees [5, 26]. Let $\breve{C}$ be a 2-edge-connected component of $\breve{G}$. We select an arbitrary vertex $\breve{v} \in \breve{C}$ as a root and compute two edge-disjoint spanning trees in the flow graph $\breve{C}(\breve{v})$ and two edge-disjoint spanning trees in the reverse flow graph $\breve{C}^R(\breve{v})$. 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 2-edge-connected components 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 $\breve{G}$ that do not belong in a 2-edge-connected component of $\breve{G}$. 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 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 2-edge- and the 2-vertex- connected components of $G$. This is immediate for Steps 1 and 3. In Step 2, we do not need to compute the 2-edge-connected components of $\breve{G}$ from scratch, but we can form them from the 2-edge-connected components of $G$ using contractions. Let $C$ be a 2-edge-connected component of $G$. We contract each 2-vertex-connected component of $G$ contained in $C$ into a single supervertex. Then, the resulting digraph $\breve{C}$ is a 2-edge-connected component of $\breve{G}$. □

If we wish to improve the quality of the computed solution $G'$, we can apply the 2VDP filter, and the analogous 2-edge-disjoint paths filter 2EDP, as follows. In Step 1, we run the 2VDP filter for the edges computed by the linear-time algorithm of [10]. This produces a minimal solution for 2VCSS in each 2-vertex-connected component of $G$. Similarly, in Step 2, we run the 2EDP filter for the edges of the edge-disjoint spanning trees computed in each 2-edge-connected component of $\breve{G}$. This produces a minimal solution for 2ECSS in each 2-edge-connected component of $\breve{G}$. Finally, we run the 2VDP 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 components. We implemented this algorithm, using DLN-B for Step 3, and refer to it as 2VDP-2C.

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

**Theorem 4.** *There is a polynomial-time algorithm for 2VC-B-C that achieves an approximation ratio of* 6*. Moreover, if the* 2*-vertex-connected components of* $G$ *are available, then the algorithm runs in linear time.*

*Proof.* A result in [10] shows that, given a 2-vertex-connected digraph with $\nu$ vertices, we can compute in linear time a 2-vertex-connected spanning subgraph that has less than $6\nu$ edges. Hence, if $n_c$ is the number of vertices that belong in a 2-vertex-connected component of $G$, then applying this algorithm to each 2-vertex-connected component selects less than $6n_c$ edges. Finally, we apply the construction of a sparse certificate for the 2-vertex-connected blocks which selects at most $6(n-1)$ edges by Theorems 1 or 2. Hence, the subgraph computed by the algorithm has less than $6(n+n_c)$. One the other hand, any solution to 2VC-B-C has to include at least $2n_c$ edges for the 2-vertex-connected components 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 2VC-B and 2C problems, we can improve the quality of the computed solution by applying the 2VDP filter for the edges that connect different 2-vertex-connected components. We implemented this algorithm, using DLN-B for Step 3, and refer to it as 2VDP-B-C.

## 5 Experimental Analysis

We implemented the algorithms previously described: 5 for 2VC-B, 3 for 2VC-B-C, and 3 for 2C, as summarized in Table 1. 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 2. 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 $\delta^{A}_{avg}$ is the average vertex indegree of the 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 2VC-B and 2VC-B-C we define $\delta^{B}_{avg} = (n+k)/n$, where $n$ is the total number of vertices of the input digraph and $k$ is the number of vertices that belong in (nontrivial) 2-vertex-connected blocks [2]. We set a similar lower bound $\delta^{C}_{avg}$ for 2C, with

[2] This follows from the fact that in the sparse subgraph the $k$ vertices in blocks 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.

| Algorithm | Problem | Technique | Time |
|---|---|---|---|
| DST-B | 2VC-B | Original sparse certificate from [12] based on divergent spanning trees | $O(m+n)$ |
| DST-B modified | 2VC-B | Modified sparse certificate from [12] | $O(m+n)$ |
| DLN-B | 2VC-B | Sparse certificate from [14] based on divergent spanning trees and loop nesting trees | $O(m+n)$ |
| LHL-B | 2VC-B | New sparse certificate based on low-high orders and loop nesting trees | $O(m+n)$ |
| 2VDP-B | 2VC-B | 2VDP filter applied on the digraph produced by DLN-B | $O(n^2)$ |
| DLN-B-C | 2VC-B-C | DST-B combined with the linear-time 2VCSS algorithm of [10] | $O(m+n)^\dagger$ |
| LHL-B-C | 2VC-B-C | LHL-B combined with the linear-time 2VCSS algorithm of [10] | $O(m+n)^\dagger$ |
| 2VDP-B-C | 2VC-B-C | 2VDP filter applied on the digraph produced by DLN-B-C | $O(n^2)$ |
| DLN-2C | 2C | DLN-B-C combined with the linear-time 2ECSS algorithm using edge-disjoint spanning trees | $O(m+n)^\ddagger$ |
| LHL-2C | 2C | LHL-B-C combined with the linear-time 2ECSS algorithm using edge-disjoint spanning trees | $O(m+n)^\ddagger$ |
| 2VDP-2C | 2C | 2VDP and 2EDP filters applied on the digraph produced by DLN-2C | $O(n^2)$ |

**Table 1.** The algorithms considered in our experimental study. The worst-case bounds refer to a digraph with $n$ vertices and $m$ edges. Running times indicated by † assume that the 2-vertex-connected components of the input digraph are available; running times indicated by ‡ assume that also the 2-edge-connected components are available.

| Dataset | $n$ | $m$ | file size | $\delta_{avg}$ | $s^*$ | $\delta_{avg}^B$ | $\delta_{avg}^C$ | type |
|---|---|---|---|---|---|---|---|---|
| Rome99 | 3353 | 8859 | 100KB | 2.64 | 789 | 1.76 | 1.76 | road network |
| P2p-Gnutella25 | 5153 | 17695 | 203KB | 3.43 | 1840 | 1.60 | 1.60 | peer2peer |
| P2p-Gnutella31 | 14149 | 50916 | 621KB | 3.59 | 5357 | 1.56 | 1.56 | peer2peer |
| Web-NotreDame | 53968 | 296228 | 3,9MB | 5.48 | 9629 | 1.50 | 1.50 | web graph |
| Soc-Epinions1 | 32223 | 443506 | 5,3MB | 13.76 | 8194 | 1.56 | 1.56 | social network |
| USA-road-NY | 264346 | 733846 | 11MB | 2.77 | 46476 | 1.80 | 1.80 | road network |
| USA-road-BAY | 321270 | 800172 | 12MB | 2.49 | 84627 | 1.69 | 1.69 | road network |
| USA-road-COL | 435666 | 1057066 | 16MB | 2.42 | 120142 | 1.68 | 1.68 | road network |
| Amazon0302 | 241761 | 1131217 | 16MB | 4.67 | 69616 | 1.74 | 1.74 | prod. co-purchase |
| WikiTalk | 111881 | 1477893 | 18MB | 13.20 | 14801 | 1.45 | 1.45 | social network |
| Web-Stanford | 150532 | 1576314 | 22MB | 10.47 | 14801 | 1.62 | 1.58 | web graph |
| Amazon0601 | 395234 | 3301092 | 49MB | 8.35 | 69387 | 1.82 | 1.82 | prod. co-purchase |
| Web-Google | 434818 | 3419124 | 50MB | 7.86 | 89838 | 1.59 | 1.58 | web graph |
| Web-Berkstan | 334857 | 4523232 | 68MB | 13.50 | 53666 | 1.56 | 1.51 | web graph |

**Table 2.** Real-world graphs sorted by file size of their largest SCC; $n$ is the number of vertices, $m$ the number of edges, and $\delta_{avg}$ is the average vertex indegree; $s^*$ is the number of strong articulation points; $\delta_{avg}^B$ and $\delta_{avg}^C$ are lower bounds on the average vertex indegree of an optimal solution to 2VC-B and 2C, respectively.

the only difference that $k$ is the number of vertices that belong in (nontrivial) 2-edge-connected blocks, since every 2-vertex-connected component or block is contained in a 2-edge-connected block. Note that the quality ratio is an upper bound of the actual approximation ratio. 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 $\mathcal{P}$.

We now report the results of our experiments with all the algorithms considered for problems 2VC-B and 2C. For the 2VC-B problem, the quality ratio of the spanning subgraphs computed by the different algorithms is shown in Table 3 (left) and Figure 2 (top), while their running times are given and plotted in

| Dataset | DST-B | DST-B modified | DLN-B | LHL-B | 2VDP-B | DLN-B-C | LHL-B-C | 2VDP-B-C | DLN-2C | LHL-2C | 2VDP-2C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Rome99 | 1.384 | 1.363 | 1.432 | 1.388 | 1.170 | 1.462 | 1.459 | 1.199 | 1.462 | 1.459 | 1.198 |
| P2p-Gnutella25 | 1.726 | 1.602 | 1.713 | 1.568 | 1.234 | 1.712 | 1.568 | 1.234 | 1.712 | 1.568 | 1.234 |
| P2p-Gnutella31 | 1.717 | 1.647 | 1.732 | 1.602 | 1.273 | 1.732 | 1.573 | 1.273 | 1.732 | 1.573 | 1.273 |
| Web-NotreDame | 2.072 | 2.067 | 2.108 | 2.085 | 1.588 | 2.232 | 2.149 | 1.628 | 2.250 | 2.180 | 1.638 |
| Soc-Epinions1 | 2.082 | 1.964 | 2.213 | 2.027 | 1.475 | 2.474 | 2.411 | 1.572 | 2.474 | 2.411 | 1.573 |
| USA-road-NY | 1.255 | 1.251 | 1.371 | 1.357 | 1.168 | 1.376 | 1.374 | 1.175 | 1.376 | 1.374 | 1.175 |
| USA-road-BAY | 1.315 | 1.311 | 1.374 | 1.365 | 1.242 | 1.375 | 1.379 | 1.246 | 1.375 | 1.379 | 1.246 |
| USA-road-COL | 1.308 | 1.307 | 1.354 | 1.348 | 1.249 | 1.357 | 1.357 | 1.252 | 1.357 | 1.357 | 1.252 |
| Amazon0302 | 1.918 | 1.791 | 1.849 | 1.719 | 1.245 | 2.020 | 1.928 | 1.386 | 2.032 | 1.944 | 1.399 |
| WikiTalk | 2.145 | 2.126 | 2.281 | 2.190 | 1.796 | 2.454 | 2.441 | 1.863 | 2.454 | 2.441 | 1.863 |
| Web-Stanford | 2.115 | 2.019 | 2.130 | 2.078 | 1.572 | 2.287 | 2.257 | 1.622 | 2.238 | 2.209 | 1.584 |
| Amazon0601 | 1.926 | 1.793 | 1.959 | 1.747 | 1.196 | 2.241 | 2.155 | 1.278 | 2.242 | 2.157 | 1.279 |
| Web-Google | 2.052 | 2.004 | 2.083 | 2.051 | 1.485 | 2.306 | 2.335 | 1.585 | 2.338 | 2.372 | 1.602 |
| Web-Berkstan | 2.302 | 2.233 | 2.290 | 2.275 | 1.692 | 2.472 | 2.492 | 1.767 | 2.410 | 2.431 | 1.717 |

**Table 3.** Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2VC-B, 2VC-B-C and 2C.

| Dataset | DST-B | DST-B modified | DLN-B | LHL-B | 2VDP-B | DLN-B-C | LHL-B-C | 2VDP-B-C | DLN-2C | LHL-2C | 2VDP-2C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Rome99 | 0.014 | 0.018 | 0.004 | 0.005 | 0.264 | 0.032 | 0.034 | 0.122 | 0.034 | 0.036 | 0.122 |
| P2p-Gnutella25 | 0.027 | 0.032 | 0.008 | 0.007 | 1.587 | 0.042 | 0.042 | 0.729 | 0.051 | 0.053 | 0.725 |
| P2p-Gnutella31 | 0.070 | 0.094 | 0.024 | 0.027 | 13.325 | 0.119 | 0.119 | 5.613 | 0.143 | 0.149 | 5.422 |
| Web-NotreDame | 0.335 | 0.486 | 0.059 | 0.080 | 97.355 | 0.491 | 0.521 | 27.091 | 0.573 | 0.600 | 27.746 |
| Soc-Epinions1 | 0.258 | 0.309 | 0.089 | 0.110 | 92.812 | 0.606 | 0.621 | 54.559 | 0.602 | 0.664 | 54.548 |
| USA-road-NY | 1.095 | 1.402 | 0.261 | 0.360 | 2546.484 | 2.227 | 2.337 | 991.092 | 2.153 | 2.415 | 995.913 |
| USA-road-BAY | 1.659 | 2.152 | 0.316 | 0.435 | 4089.389 | 2.153 | 2.298 | 1429.443 | 2.296 | 2.476 | 1447.318 |
| USA-road-COL | 2.439 | 3.050 | 0.438 | 0.603 | 7739.256 | 3.770 | 3.969 | 3093.258 | 3.938 | 4.228 | 3064.297 |
| Amazon0302 | 2.101 | 2.410 | 0.517 | 0.675 | 3503.910 | 4.708 | 5.017 | 2244.856 | 5.135 | 5.509 | 2094.263 |
| WikiTalk | 1.777 | 2.125 | 0.355 | 0.473 | 1158.855 | 2.179 | 2.133 | 943.690 | 2.203 | 2.513 | 924.810 |
| Web-Stanford | 1.756 | 2.395 | 0.429 | 0.564 | 1174.984 | 2.037 | 2.313 | 279.236 | 2.561 | 2.487 | 317.115 |
| Amazon0601 | 3.532 | 3.924 | 1.363 | 1.605 | 15349.126 | 9.793 | 10.038 | 8065.680 | 11.669 | 11.397 | 8696.212 |
| Web-Google | 4.837 | 5.467 | 1.533 | 1.968 | 26299.714 | 9.789 | 10.172 | 5095.600 | 11.535 | 12.979 | 5128.337 |
| Web-Berkstan | 3.239 | 5.261 | 0.690 | 0.869 | 6301.410 | 4.670 | 4.872 | 1595.033 | 5.178 | 5.601 | 1546.041 |

**Table 4.** Running times in seconds of the algorithms for 2VC-B, 2VC-B-C and 2C.

Table 4 (left) and Figure 3 (left), respectively. Similarly, for the 2VC-B-C and 2C problems, the quality ratio of the spanning subgraphs computed by the different algorithms is shown in Table 3 (right) and Figure 2 (bottom), while their running times are given and plotted in Table 4 (right) and Figure 3 (right), respectively.

We observe that all our algorithms perform well in terms of the quality of the solution they compute. Indeed, the quality ratio is less than 2.5 for all algorithms and inputs. Our modified version of DST-B performs consistently better than the original version. Also in all cases, LHL-B computed a higher quality solution than DLN-B. For most inputs, DST-B modified computes a sparser graph than LHL-B, which is somewhat surprising given the fact that we do not have a good bound for the (constant) approximation ratio of DST-B modified. On the other hand, LHL-B is faster than DST-B modified by a factor of 4.15 on average and has the additional benefit of maintaining both the 2-vertex and the 2-edge-connected blocks. The 2VDP 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

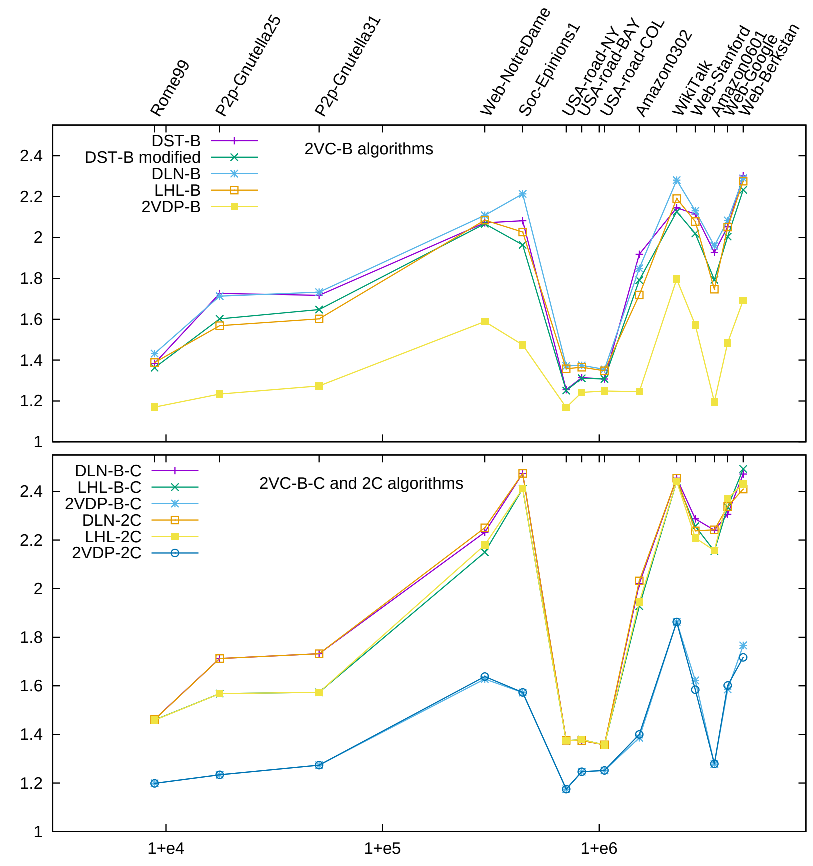

**Fig. 2.** The plotted quality ratios taken by Table 3.

higher running times, as those algorithms can be even 5 orders of magnitude slower than the other algorithms.

From the analysis of our experimental data, all algorithms achieve consistently better approximations for road networks than for most of the other graphs in our data set. This can be explained by taking into account the macroscopic structure of road networks, which is rather 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-vertex-connected components, joined together by strong bridges, and their 2-vertex-connected blocks coincide with their 2-vertex-connected components. In this setting, a sparse strongly connected subgraph of the condensed graph will preserve both blocks and components. On the other hand, such a gain on the solution for the road networks is balanced at the cost of their additional running time.

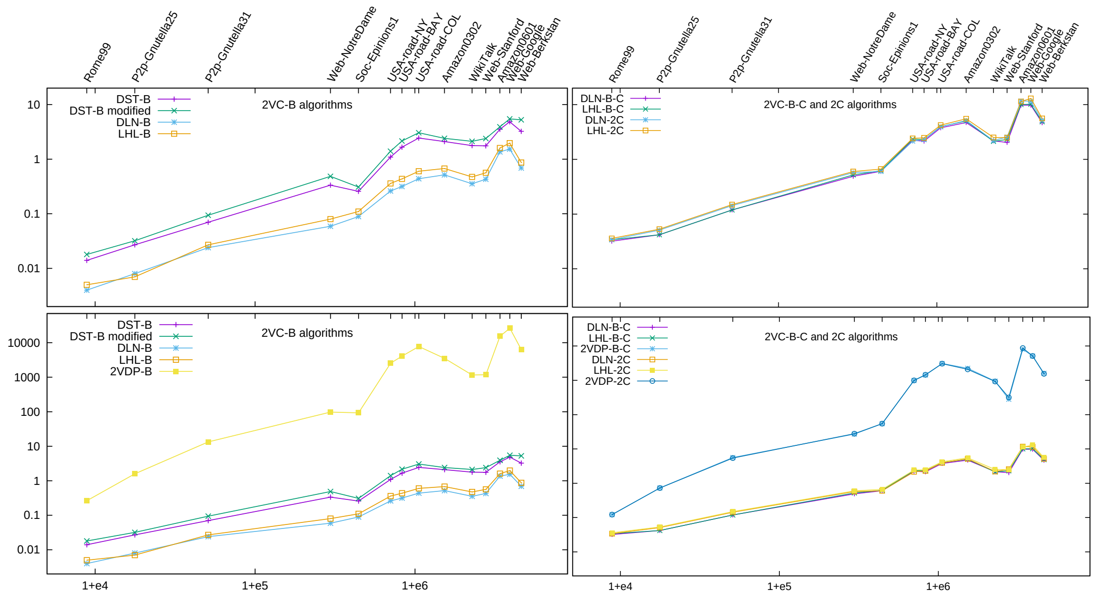

**Fig. 3.** Running times in seconds with respect to the number of edges (in log-log scale) taken by Table 4. The upper plots get a close-up view of the fastest algorithms by not considering 2VDP-B, 2VDP-B-C and 2VDP-2C.

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

## 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. J. Cheriyan and R. Thurimella. Approximating minimum-size $k$-connected spanning subgraphs via matching. *SIAM J. Comput.*, 30(2):528–560, 2000.
5. J. Edmonds. Edge-disjoint branchings. *Combinat. Algorithms*, pages 91–96, 1972.
6. J. Fakcharoenphol and B. Laekhanukit. An $o(\log^2 k)$-approximation algorithm for the $k$-vertex connected spanning subgraph problem. In *Proc. 40th ACM Symp. on Theory of Computing*, STOC '08, pages 153–158, New York, NY, USA, 2008. ACM.
7. L. R. Ford; D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

8. 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.
9. 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.
10. L. Georgiadis. Approximating the smallest 2-vertex connected spanning subgraph of a directed graph. In *Proc. 19th European Symposium on Algorithms*, pages 13–24, 2011.
11. L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-edge connectivity in directed graphs. In *SODA 2015*, pages 1988–2005, 2015.
12. L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-vertex connectivity in directed graphs. In *ICALP 2015*, pages 605–616, 2015.
13. L. Georgiadis, G. F. Italiano, C. Papadopoulos, and N. Parotsidis. Approximating the smallest spanning subgraph for 2-edge-connectivity in directed graphs. In *ESA 2015*, pages 582–594, 2015.
14. L. Georgiadis, G. F. Italiano, and N. Parotsidis. A new framework for strong connectivity and 2-connectivity in directed graphs. *CoRR*, arXiv:1511.02913, November 2015.
15. 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.
16. M. Henzinger, S. Krinninger, and V. Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *ICALP 2015*, pages 713–724, 2015.
17. 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.
18. R. Jaberi. Computing the 2-blocks of directed graphs. *RAIRO-Theor. Inf. Appl.*, 49(2):93–119, 2015.
19. R. Jaberi. On computing the 2-vertex-connected components of directed graphs. *Discrete Applied Mathematics*, 2015. To appear.
20. S. Khuller, B. Raghavachari, and N. E. Young. Approximating the minimum equivalent digraph. *SIAM J. Comput.*, 24(4):859–872, 1995. Announced at *SODA 1994*, 177-186.
21. S. Khuller, B. Raghavachari, and N. E. Young. On strongly connected digraphs with bounded cycle length. *Discrete Applied Mathematics*, 69(3):281–289, 1996.
22. G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. *Approximation Algorithms and Metaheuristics*, 2007.
23. B. Laekhanukit, S. O. Gharan, and M. Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *ICALP 2012*, pages 606–616, 2012.
24. H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition.
25. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
26. R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–85, 1976.
27. A. Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *SODA 2001*, pages 417–426, 2001.
28. 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.