Approximating the Smallest Spanning Subgraph for 2-Edge-Connectivity in Directed Graphs

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

Abstract. Let G be a strongly connected directed graph. We consider the following three problems, where we wish to compute the smallest strongly connected spanning subgraph of G that maintains respectively: the 2-edge-connected blocks of G (2EC-B); the 2-edge-connected components of G (2EC-C); both the 2-edge-connected blocks and the 2-edge-connected components of G (2EC-B-C). All three problems are NP-hard, and thus we are interested in efficient approximation algorithms. For 2EC-C we can obtain a 3/2-approximation by combining previously known results. For 2EC-B and 2EC-B-C, we present new 4-approximation algorithms that run in linear time. We also propose various heuristics to 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. An edge of G is a *strong bridge* if its removal increases the number of strongly connected components of G. A digraph G is 2-edge-connected if it has no strong bridges. The 2-edge-connected components of G are its maximal 2-edge-connected subgraphs. Let v and w be two distinct vertices: v and w are 2-*edge-connected*, denoted by $v \leftrightarrow_{2e} w$, if there are two edge-disjoint directed paths from v to w and two edge-disjoint directed paths from w to v. (Note that a path from v to w and a path from w to v need not be edge-disjoint.) A 2-*edge-connected block* of $G = (V, E)$ is a maximal subset $B \subseteq V$ such that $u \leftrightarrow_{2e} v$ for all $u, v \in B$. Differently from undirected graphs, in digraphs 2-edge-connected blocks can be different from the 2-edge-connected components, i.e., two vertices may be 2-edge-connected but lie in different 2-edge-connected components. See Figure 1.

Computing a smallest spanning subgraph that maintains the same edge or vertex connectivity properties of the original graph is a fundamental problem in network design, with many practical applications [15]. In this paper we consider the problem of finding the smallest spanning subgraph of G that maintains certain 2-edge-connectivity requirements in addition to strong connectivity. Specifically, we distinguish three problems that we refer to as 2EC-B, 2EC-C and 2EC-B-C. In particular, we wish to compute the smallest strongly connected spanning

[1] University of Ioannina, Greece. E-mail: {loukas,charis,nparotsi}@cs.uoi.gr.
[2] Università di Roma "Tor Vergata", Italy. E-mail: giuseppe.italiano@uniroma2.it.
Partially supported by MIUR under Project AMANDA.

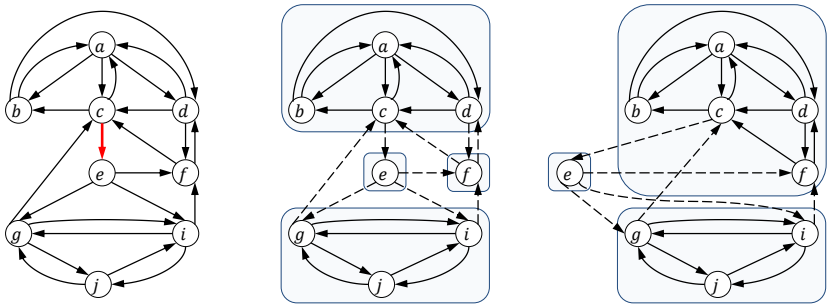

Fig. 1. From left-to-right we show: a strongly connected digraph G with a strong bridge (c, e), the 2-edge-connected components of G, and the 2-edge-connected blocks of G.

subgraph of a digraph G that maintains the following properties: the pairwise 2-edge-connectivity of G, i.e., the 2-edge-connected blocks of G (2EC-B); the 2-edge-connected components of G (2EC-C); both the 2-edge-connected blocks and the 2-edge-connected components of G (2EC-B-C). Since all those problems are NP-hard [7], we are interested in designing efficient approximation algorithms.

Related Work. Finding a smallest k-edge-connected (resp. k-vertex-connected) spanning subgraph of a given k-edge-connected (resp. k-vertex-connected) digraph is NP-hard for $k \geq 2$ for undirected graphs, and for $k \geq 1$ for digraphs [7]. Problems of this type, together with more general variants of approximating minimum-cost subgraphs that satisfy certain connectivity requirements, have received a lot of attention, and several important results have been obtained. See, e.g., the survey [13]. Currently, the best approximation ratio for computing the smallest strongly connected spanning subgraph (SCSS) is 3/2 achieved by Vetta [17]. A linear-time algorithm that achieves a 5/3-approximation was given by Zhao et al. [18]. For the smallest k-edge-connected spanning subgraph (kECSS), Laehanukit et al. [14] gave a randomized $(1+1/k)$-approximation algorithm. Regarding hardness of approximation, Gabow et al. [5] showed that there exists an absolute constant $c > 0$ such that for any integer $k \geq 1$, approximating the smallest kECSS on directed multigraphs to within a factor $1 + c/k$ in polynomial time implies P = NP. Jaberi [12] considered various optimization problems related to 2EC-B and proposed corresponding approximation algorithms. The approximation ratio in Jaberi's algorithms, however, is linear in the number of strong bridges, and hence $O(n)$ in the worst case.

Our Results. In this paper we provide both theoretical and experimental contributions to the 2EC-B, 2EC-C and 2EC-B-C problems. A 3/2-approximation for 2EC-C can be obtained by carefully combining the 2ECSS randomized algorithm of Laehanukit et al. [14] and the SCSS algorithm of Vetta [17]. A faster and deterministic 2-approximation algorithm for 2EC-C can be obtained by combining techniques based on edge-disjoint spanning trees [4, 16] with the SCSS algorithm of Zhao et al. [18]. We remark that the other two problems considered here, 2EC-B and 2EC-B-C, seem harder to approximate. The only known result is the *sparse certificate* for 2-edge-connected blocks of [8], which implies a linear-time $O(1)$-approximation algorithm for 2EC-B. 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 2EC-B and 2EC-B-C that run in linear time (the algorithm for 2EC-B-C runs in linear time once the 2-edge-connected components of G are available; if not, they can be computed in $O(n^2)$ time [10]).

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 our algorithms 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. For lack of space, proofs and some details are omitted and will be given in the full paper.

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 is reflexive and transitive. Its transitive reduction is a rooted tree, the *dominator tree* $D(s)$: u dominates w if and only if u is an ancestor of w in $D(s)$. If $w \neq s$, $d(w)$, the parent of w in $D(s)$, is the *immediate dominator* of w: it is the unique proper dominator of w that is dominated by all proper dominators of w. The dominator tree of a flow graph can be computed in linear time, see, e.g., [1, 2]. An edge (u, w) is a *bridge* in $G(s)$ if all paths from s to w include (u, w).[1] Italiano et al. [11] showed that the strong bridges of G can be computed from the bridges of the 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 B and R 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 [16]. The 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 [16], using the disjoint set union data structure of Gabow and Tarjan [6]. Two spanning trees B and R rooted at s are *independent* if for all vertices v,

[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.

the paths from s to v in B and R share only the dominators of v. Every flow graph $G(s)$ has two such spanning trees, computable in linear time [9] which are maximally edge-disjoint.

3 Approximation algorithms and heuristics

We describe our main approaches for solving problem 2EC-B. Let $G = (V, E)$ be the input directed graph. The first two algorithms 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 $G' = G$, and the order in which the edges are processed is arbitrary. The third algorithm starts with the empty graph $G' = (V, \emptyset)$, and adds the edges of spanning trees of certain subgraphs of G until the resulting digraph is strongly connected and has the same 2-edge-connected blocks as G.

Two Edge-Disjoint Paths Test. We test if $G' \setminus (x, y)$ contains two edge-disjoint paths from x to y. If this is the case, then we remove edge (x, y). This test takes $O(m)$ time per edge, so the total running time is $O(m^2)$. We refer to this algorithm as Test2EDP-B. Note that Test2EDP-B computes a minimal 2-approximate solution for the 2ECSS problem [3], which is not necessarily minimal for the 2EC-B problem.

2-Edge-Connected Blocks Test. If (x, y) is not a strong bridge in G', we test if $G' \setminus (x, y)$ has the same 2-edge-connected blocks as G'. If this is the case then we remove edge (x, y). We refer to this algorithm as Test2ECB-B. Since the 2-edge-connected blocks of a graph can be computed in linear time [8], Test2ECB-B runs in $O(m^2)$ time. Test2ECB-B computes a minimal solution for 2EC-B and achieves an approximation ratio of 4.

Independent Spanning Trees. We can compute a sparse certificate for 2-edge-connected blocks as in [8], based on a linear-time construction of two independent spanning trees of a flow graph [9]. We refer to this algorithm as IST-B original. We will show later that a suitably modified construction, which we refer to as IST-B, yields a linear-time 4-approximation algorithm.

Test2EDP-B and Test2ECB-B can be combined into a hybrid algorithm (Hybrid-B), as follows: if the tested edge (x, y) connects vertices in the same block (i.e., $x \leftrightarrow_{2e} y$), then apply Test2EDP-B; otherwise, apply Test2ECB-B. One can show that Hybrid-B returns the same sparse subgraph as Test2ECB-B.

In algorithm Hybrid-B we also apply an additional speed-up heuristic for *trivial edges* (x, y): if x belongs to a nontrivial block (i.e., a block of size ≥ 2) and has outdegree two or y belongs to a nontrivial block and has indegree two, then (x, y) must be included in the solution. As we show later in our experiments, such a simple test can yield significant performance gains.

Note that Test2EDP-B, Test2ECB-B and Hybrid-B produce a 4-approximation for 2EC-B in $O(n^2)$ time if they are run on the sparse subgraph computed by IST-B instead of the original digraph. We observed experimentally that this also improves the quality of the computed solutions in practice. Therefore, we applied this idea in all our implementations. See Table 1 in Section 4.

Although all the above algorithms do not maintain the 2-edge-connected components of the original graph, we can still apply them to get an approximation for 2EC-B-C, as follows. First, we compute the 2-edge-connected components of G and solve the 2ECSS problem independently for each such component. Then, we can apply any of the algorithms for 2EC-B (Test2EDP-B, Test2ECB-B, Hybrid-B or IST-B) for the edges that connect different components. To speed them up, we apply them to a *condensed graph* H that is formed from G by contracting each 2-edge-connected component of G into a single supervertex. Note that H is a multigraph since the contractions can create loops and parallel edges. For any vertex v of G, we denote by $h(v)$ the supervertex of H that contains v. Every edge $(h(u), h(v))$ of H is associated with the corresponding original edge (u, v) of G. Algorithms Test2EDP-BC, Test2ECB-BC, Hybrid-BC or IST-BC are obtained by applying to graph H the corresponding algorithm for 2EC-B. Let H' be the obtained subgraph of H, and let G' be the digraph that is obtained after we expand back each supervertex of H with its 2-edge-connected sparse subgraph computed before. Then, G' is a valid solution to the 2EC-B-C problem.

As a special case of applying Test2EDP-B to H, we can immediately remove loops and parallel edges $(h(u), h(v))$ if H has more than two edges directed from $h(u)$ to $h(v)$. To obtain faster implementations, we solve the 2ECSS problems in linear-time using edge-disjoint spanning trees [4, 16]. Let C be a 2-edge-connected component 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 2ECSS on C. Moreover, as in 2EC-B, we can apply algorithms Test2EDP-BC, Test2ECB-BC and Hybrid-BC on the sparse subgraph computed by IST-BC. Then, these algorithms produce a 4-approximation for 2EC-B-C in $O(n^2)$ time. Furthermore, for these $O(n^2)$-time algorithms, we can improve the approximate solution C' for 2ECSS on each 2-edge-connected component 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.

We can also use the condensed graph in order to obtain an efficient approximation algorithm for 2EC-C. To that end, we can apply the algorithm of Laehanukit et al. [14] and get a 3/2-approximation of the 2ECSS problem independently for each 2-edge-connected component of G. Then, since we only need to preserve the strong connectivity of H, we can run the algorithm of Vetta [17] on a digraph $\tilde{H}$ that results from H after removing all loops and parallel edges. This computes a spanning subgraph H' of $\tilde{H}$ that is a 3/2-approximation for SCSS in H. The corresponding expanded graph G', where we substitute each supervertex $h(v)$ of H with the approximate smallest 2ECSS, gives a 3/2-approximation for 2EC-C. A faster and deterministic 2-approximation algorithm for 2EC-C can be obtained as follows. For the 2ECSS problems we use the edge-disjoint spanning trees 2-approximation algorithm described above. Then, we solve SCSS on $\tilde{H}$ by applying the linear-time algorithm of Zhao et al. [18]. This yields a 2-approximation algorithm for 2EC-C that runs in linear time once the 2-edge-

connected components of G are available (if not, they can be computed in $O(n^2)$ time [10]). We refer to this algorithm as ZNI-C.

Theorem 1. *There is a polynomial-time algorithm for 2EC-C that achieves an approximation ratio of 3/2. Moreover, if the 2-edge-connected components of G are available, then we can compute a 2-approximate 2EC-C in linear time.*

3.1 Independent Spanning Trees

Here we present our new algorithm IST-B and prove that it gives a linear-time 4-approximation for 2EC-B and 2EC-B-C. Since IST-B is a modified version of the sparse certificate $C(G)$ for the 2-edge-connected blocks of a digraph G [8] (IST-B original), let us review IST-B original first.

Let s be an arbitrarily chosen start vertex of the strongly connected digraph G. The *canonical 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 $T(v)$ denote the tree containing vertex v in this decomposition. We refer to the subtree roots in the canonical 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 of all the vertices in $T(r)$, referred to as *ordinary* vertices, and a set of *auxiliary* vertices, which are obtained by contracting vertices in $V \setminus T(r)$, as follows. Let v be a vertex in $T(r)$. We say that v is a *boundary vertex in* $T(r)$ 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 $V \setminus T(r)$ that are not descendants of r are contracted into $d(r)$ ($r \neq s$ if any such vertex exists). 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 [8], 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 blocks of G are a refinement of the vertex sets in the trees of the canonical decomposition. The sparse certificate of [8] is constructed in three phases. We maintain a list (multiset) L of the edges to be added in $C(G)$; initially $L = \emptyset$. The same edge may be inserted into L multiple times, but the total number of insertions will be $O(n)$. So the edges of $C(G)$ can be obtained from L 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 this case we insert the associated original edge instead.

Phase 1. We insert into L the edges of two independent spanning trees, $B(G(s))$ and $R(G(s))$ 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 independent spanning trees $B(H^R(r))$

and $R(H^R(r))$ for the corresponding reverse flow graph $H^R(r)$ with start vertex r. We insert into L the edges of these two spanning trees. We note that L 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 L.

The above construction inserts $O(n)$ edges into $C(G)$, and therefore achieves a constant approximation ratio for 2EC-B. 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 applying two levels of the canonical decomposition. In the next section we analyze a modified version of the sparse certificate construction, and show that it achieves a 4-approximation for 2EC-B. Then we show that we also achieve a 4-approximation for 2EC-B-C by applying this sparse certificate on the condensed graph H.

The new algorithm IST-B. The main idea behind IST-B 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 $G(s)$.

Consider Phase 2. Let $H = G_r$ be a first-level auxiliary graph. In the sparse certificate we include two independent spanning trees, $B(H^R(r))$ and $R(H^R(r))$, 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 $B(H^R(r))$ and $R(H^R(r))$. 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. 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 $B(H^R(r))$. Then $z = d(x)$ is the unique child of x in $B(H^R(r))$, and the reverse of the edge (x, z) of $B(H^R(r))$ is already included in $C(G)$ by Phase 1. Therefore, in all cases, we can charge to x at most one new edge.

Now we consider 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. 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 $C(G)$. If x is ordinary in H but a second-level auxiliary vertex in H_q then it has a unique leaving edge (x, z), which is a bridge in $H^R(r)$ and $C(G)$ already contains a corresponding original edge. 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 the $2b$ bound follows. Using the above construction we obtain the following result (see the full paper for the complete proof):

Theorem 2. *There is a linear-time approximation algorithm for the 2EC-B problem that achieves an approximation ratio of* 4*. Moreover, if the* 2*-edge-connected components of the input digraph are known in advance, we can compute a* 4*-approximation for the 2EC-B-C problem in linear time.*

Heuristics applied on auxiliary graphs. To speed up algorithms from the Test2EDP and Hybrid families, we applied them to the first-level and second-level 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 variants as Test2EDP-B-Aux, Hybrid-B-Aux, Test2EDP-BC-Aux, Hybrid-BC-Aux, depending on the algorithm (Test2EDP or Hybrid) and problem (2EC-B or 2EC-B-C) considered.

4 Experimental Analysis

We implemented the algorithms previously described: 7 for 2EC-B, 6 for 2EC-B-C, and one for 2EC-C, 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

Algorithm	Problem	Technique	Time
ZNI-C	2EC-C	Zhao et al. [18] applied on the condensed graph	$O(m+n)^\dagger$
IST-B original	2EC-B	Original sparse certificate from [8]	$O(m+n)$
IST-B	2EC-B	Modified sparse certificate	$O(m+n)$
Test2EDP-B	2EC-B	Two edge-disjoint paths test on sparse certificate of input graph	$O(n^2)$
Test2ECB-B	2EC-B	2-edge-connected blocks test on sparse certificate of input graph	$O(n^2)$
Hybrid-B	2EC-B	Hybrid of two edge-disjoint paths and 2-edge-connected blocks test on sparse certificate of input graph	$O(n^2)$
Test2EDP-B-Aux	2EC-B	Test2EDP-B applied on second-level auxiliary graphs	$O(n^2)$
Hybrid-B-Aux	2EC-B	Hybrid-B applied on second-level auxiliary graphs	$O(n^2)$
IST-BC	2EC-B-C	Modified sparse certificate preserving 2-edge-connected components (applied on the condensed graph)	$O(m+n)^\dagger$
Test2EDP-BC	2EC-B-C	Two edge-disjoint paths test on sparse certificate of condensed graph	$O(n^2)$
Test2ECB-BC	2EC-B-C	2-edge-connected blocks test on sparse certificate of condensed graph	$O(n^2)$
Hybrid-BC	2EC-B-C	Hybrid of two edge-disjoint paths and 2-edge-connected blocks test on sparse certificate of condensed graph	$O(n^2)$
Test2EDP-BC-Aux	2EC-B-C	Test2EDP-BC applied on second-level auxiliary graphs	$O(n^2)$
Hybrid-BC-Aux	2EC-B-C	Hybrid-BC applied on second-level auxiliary graphs	$O(n^2)$

Table 1. The algorithms considered in our experimental study. The worst-case bounds refer to a digraph with n vertices and m edges. †These linear running times assume that the 2-edge-connected components of the input digraph are available.

Dataset	n	m	file size	δ_{avg}	b^*	δ^B_{avg}	δ^C_{avg}	type
Rome99	3353	8859	100KB	2.64	1474	1.75	1.67	road network
P2p-Gnutella25	5153	17695	203KB	3.43	2181	1.60	1.00	peer2peer
P2p-Gnutella31	14149	50916	621KB	3.59	6673	1.56	1.00	peer2peer
Web-NotreDame	53968	296228	3,9MB	5.48	34879	1.50	1.36	web graph
Soc-Epinions1	32223	443506	5,3MB	13.76	20975	1.56	1.55	social network
USA-road-NY	264346	733846	11MB	2.77	104618	1.80	1.80	road network
USA-road-BAY	321270	800172	12MB	2.49	196474	1.69	1.69	road network
USA-road-COL	435666	1057066	16MB	2.42	276602	1.68	1.68	road network
Amazon0302	241761	1131217	16MB	4.67	73361	1.74	1.64	prod. co-purchase
WikiTalk	111881	1477893	18MB	13.20	85503	1.45	1.44	social network
Web-Stanford	150532	1576314	22MB	10.47	64723	1.62	1.33	web graph
Amazon0601	395234	3301092	49MB	8.35	83995	1.82	1.82	prod. co-purchase
Web-Google	434818	3419124	50MB	7.86	211544	1.59	1.48	web graph
Web-Berkstan	334857	4523232	68MB	13.50	164779	1.56	1.39	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 δ_{avg} is the average vertex indegree; b^* is the number of strong bridges; δ^B_{avg} and δ^C_{avg} are lower bounds on the average vertex indegree of an optimal solution to 2EC-B and 2EC-C, respectively.

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 δ^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 2EC-B and 2EC-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-edge-connected blocks.[2] We set a similar lower bound δ^C_{avg} for 2EC-C, with the only difference that k is the number of vertices that belong in nontrivial 2-edge-connected components. 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 2EC-B, 2EC-B-C and 2EC-C. As previously mentioned, for the sake of efficiency, all variants of Test2EDP, Test2ECB and Hybrid were run on the sparse certificate computed by either IST-B or IST-BC (depending on the problem at hand) instead of the original digraph. For the 2EC-B problem, the quality ratio of the spanning subgraphs computed by the different algorithms is shown in Table 3, while their running times are plotted in Figure 2 (top). Similarly, for the 2EC-C and 2EC-B-C problems, the quality ratio of the spanning subgraphs computed by the different algorithms is shown in Table 4, while their running times are plotted in Figure 2 (bottom).

There are two peculiarities related to road networks that emerge immediately from the analysis of our experimental data. First, all algorithms achieve consistently better approximations for road networks than for most of the other graphs in our data set. Second, for the 2EC-B problem the Hybrid algorithms (Hybrid-B and Hybrid-B-Aux) seem to achieve substantial speedups on road networks; for the 2EC-B-C problem, this is even true for Test2ECB-BC. The first phenomenon 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) (expect for one-way roads). Roughly speaking, road networks mainly consist of the union of 2-edge-connected components, joined together by strong bridges, and their 2-edge-connected blocks coincide with their 2-edge-connected components. In this setting, a sparse strongly connected subgraph of the condensed graph will preserve both blocks and components. The second phenomenon is mainly due to the *trivial edge* heuristic described in Section 3.

Apart from the peculiarities of road networks, ZNI-C behaves as expected for 2EC-C through its linear-time 2-approximation algorithm. Note that for both problems 2EC-B and 2EC-B-C, all algorithms achieve quality ratio significantly smaller than our theoretical bound of 4. Regarding running times, we observe that the 2EC-B-C algorithms are faster than the 2EC-B 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 (IST-B and IST-B original for problem 2EC-B; IST-BC for 2EC-B-C) and the slowest algorithms (Test2ECB-B and Hybrid-B for 2EC-B; Test2ECB-BC and Hybrid-BC for 2EC-B-C) tend to produce respectively the

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

Dataset	IST-B original	IST-B	Test2EDP-B	Test2ECB-B & Hybrid-B	Test2EDP-B-Aux	Hybrid-B-Aux
Rome99	1.389	1.363	1.171	1.167	1.177	1.174
P2p-Gnutella25	1.656	1.512	1.220	1.143	1.251	1.234
P2p-Gnutella31	1.682	1.541	1.251	1.169	1.291	1.274
Web-NotreDame	1.964	1.807	1.489	1.417	1.500	1.471
Soc-Epinions1	2.047	1.837	1.435	1.379	1.441	1.406
USA-road-NY	1.343	1.245	1.174	1.174	1.175	1.175
USA-road-BAY	1.361	1.307	1.245	1.246	1.246	1.246
USA-road-COL	1.354	1.304	1.251	1.252	1.252	1.252
Amazon0302	1.762	1.570	1.186	1.134	1.206	1.196
WikiTalk	2.181	2.050	1.788	1.588	1.792	1.615
Web-Stanford	1.907	1.688	1.409	1.365	1.418	1.406
Amazon0601	1.866	1.649	1.163	1.146	1.170	1.166
Web-Google	1.921	1.728	1.389	1.322	1.401	1.377
Web-Berkstan	2.048	1.775	1.480	1.427	1.489	1.469

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

Dataset	ZNI-C	IST-BC	Test2EDP-BC	Test2ECB-BC & Hybrid-BC	Test2EDP-BC-Aux	Hybrid-BC-Aux
Rome99	1.360	1.371	1.197	1.187	1.197	1.195
P2p-Gnutella25	1.276	1.517	1.218	1.141	1.249	1.232
P2p-Gnutella31	1.312	1.537	1.251	1.170	1.290	1.273
Web-NotreDame	1.620	1.747	1.500	1.426	1.510	1.484
Soc-Epinions1	1.790	1.847	1.488	1.435	1.489	1.476
USA-road-NY	1.343	1.341	1.163	1.163	1.163	1.163
USA-road-BAY	1.360	1.357	1.237	1.237	1.237	1.237
USA-road-COL	1.343	1.339	1.242	1.242	1.242	1.242
Amazon0302	1.464	1.580	1.279	1.228	1.292	1.284
WikiTalk	1.891	2.099	1.837	1.630	1.838	1.827
Web-Stanford	1.560	1.679	1.430	1.390	1.436	1.427
Amazon0601	1.709	1.727	1.200	1.186	1.202	1.200
Web-Google	1.637	1.728	1.437	1.381	1.446	1.431
Web-Berkstan	1.637	1.753	1.516	1.472	1.523	1.511

Table 4. Quality ratio $q(A, \mathcal{P})$ of the solutions computed for 2EC-C and 2EC-B-C.

worst and the best approximations. Note that IST-B improves the quality of the solution of IST-B original at the price of slightly higher running times, while Hybrid-B (resp., Hybrid-BC) produces the same solutions as Test2ECB-B (resp., Test2ECB-BC) 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-B-Aux, Hybrid-B-Aux, Test2EDP-BC-Aux and Hybrid-BC-Aux versus Test2EDP-B, Hybrid-B, Test2EDP-BC and Hybrid-BC). Overall, in our experiments Test2EDP-B-Aux and Test2EDP-BC-Aux seem to provide good quality solutions for the problems considered without being penalized too much by a substantial performance degradation.

References

1. S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–32, 1999.
2. 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.
3. J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. Comput.*, 30(2):528–560, 2000.
4. J. Edmonds. Edge-disjoint branchings. *Combinat. Algorithms*, pages 91–96, 1972.
5. 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.

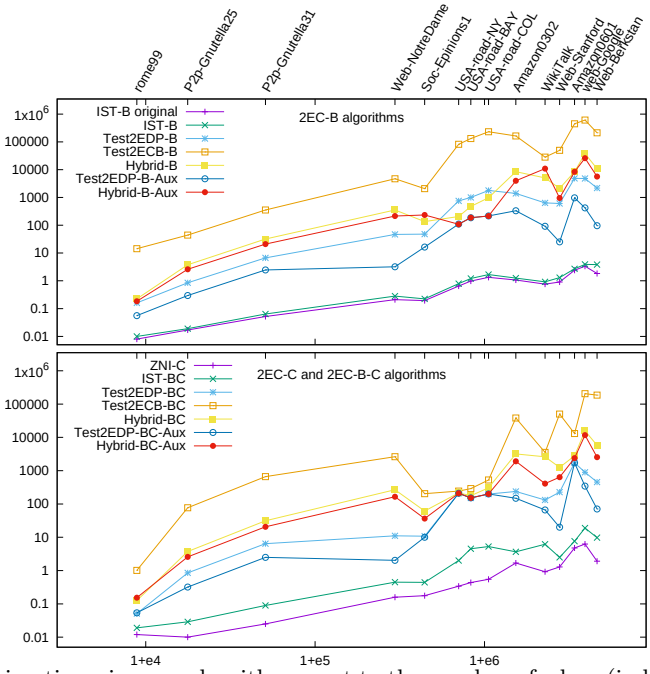

Fig. 2. Running times in seconds with respect to the number of edges (in log-log scale).

6. 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.
7. 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.
8. L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-edge connectivity in directed graphs. In *SODA 2015*, pages 1988–2005, 2015.
9. L. Georgiadis and R. E. Tarjan. Dominator tree verification and vertex-disjoint paths. In *SODA 2005*, pages 433–442, 2005.
10. 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.
11. 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.
12. R. Jaberi. Computing the 2-blocks of directed graphs. *CoRR*, abs/1407.6178, 2014.
13. G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. *Approximation Algorithms and Metaheuristics*, 2007.
14. 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.
15. H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition.
16. R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–85, 1976.
17. A. Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *SODA 2001*, pages 417–426, 2001.
18. 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.