Approximating the Smallest 2-Vertex Connected Spanning Subgraph of a Directed Graph[⋆]

Loukas Georgiadis

Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece. E-mail: `lgeorg@uowm.gr`

Abstract. We consider the problem of approximating the smallest 2-vertex connected spanning subgraph (2-VCSS) of a 2-vertex connected directed graph, and explore the efficiency of fast heuristics. First, we present a linear-time heuristic that gives a 3-approximation of the smallest 2-VCSS. Then we show that this heuristic can be combined with an algorithm of Cheriyan and Thurimella that achieves a $(1 + 1/k)$-approximation of the smallest k-VCSS. The combined algorithm preserves the 1.5 approximation guarantee of the Cheriyan-Thurimella algorithm for $k = 2$ and improves its running time from $O(m^2)$ to $O(m\sqrt{n} + n^2)$, for a digraph with n vertices and m arcs. Finally, we present an experimental evaluation of the above algorithms for a variety of input data. The experimental results show that our linear-time heuristic achieves in practice a much better approximation ratio than 3, suggesting that a tighter analysis may be possible. Furthermore, the experiments show that the combined algorithm not only improves the running time of the Cheriyan-Thurimella algorithm, but it may also compute a smaller 2-VCSS.

1 Introduction

A directed (undirected) graph is *k-vertex connected* if it has at least $k+1$ vertices and the removal of any set of at most $k-1$ vertices leaves the graph strongly connected (connected). The computation of a smallest (i.e., with minimum number of edges) k-vertex connected spanning subgraph (k-VCSS) of a given k-vertex connected graph is a fundamental problem in network design with many practical applications [11]. This problem is NP-complete for $k \geq 2$ for undirected graphs, and for $k \geq 1$ for directed graphs [10]. Recently, the more general problem of approximating minimum-cost subgraphs that satisfy certain connectivity requirements has also received a lot of attention. See, e.g., [8, 21].

Here we consider the problem of approximating the smallest k-VCSS of a directed graph (digraph) for $k = 2$. The key contribution of this work is a linear-time heuristic that provides a 3-approximation of the smallest 2-VCSS. Our new heuristic is based on recent results on independent spanning trees [14], testing 2-vertex connectivity [12], and computing strong articulation points [18] in

⋆ Research funded by the John S. Latsis Public Benefit Foundation. The sole responsibility for the content of this paper lies with its authors.

digraphs. The approximation bound of the new heuristic is by a factor of two worse compared to the currently best known bound of 1.5. The latter is achieved by the algorithm of Cheriyan and Thurimella [4], which computes a $(1 + 1/k)$-approximation of the smallest k-VCSS and runs in $O(km^2)$ time for a k-vertex connected digraph with m arcs. However, our experimental results show that in practice the linear-time heuristic performs much better than the 3 approximation bound suggests, which implies that a tighter analysis may be possible. Furthermore, we provide a simplified version of the Cheriyan-Thurimella algorithm for $k = 2$, and present a combination of our linear-time heuristic with this algorithm. The combined algorithm achieves the same approximation guarantee of 1.5 as the Cheriyan-Thurimella algorithm, and it runs in $O(m\sqrt{n} + n^2)$ time. Finally, we present some experimental results for all the above algorithms on artificial and synthetic graphs. The experiments show that the combined algorithm, besides improving the running time of the Cheriyan-Thurimella algorithm, can also compute a smaller 2-VCSS.

1.1 Definitions and Notation

We denote the vertex set and the arc (edge) set of a directed (undirected) graph G by $V(G)$ and $A(G)$ $(E(G))$, respectively.

For an undirected graph G and a subset of edges $M \subseteq E(G)$, we let $\deg_M(v)$ denote the degree of vertex v in M, i.e., the number of edges in M adjacent to v. The subset M is a *matching* if for all vertices v, $\deg_M(v) \leq 1$. A vertex v is *free* (with respect to M) if $\deg_M(v) = 0$. If for all vertices v, $\deg_M(v) \geq k$, then we call M a $(\geq k)$*-matching.* Given a function $b : V \mapsto \mathbf{Z}$ the *b-matching* problem is to compute a maximum-size subgraph $G' = (V, E(G'))$ of G such that $\deg_{E(G')}(v) \leq b(v)$ for all v. This problem is also referred to as the *degree-constrained subgraph problem* [9].

For a directed graph (digraph) G and a subset of arcs $M \subseteq A(G)$, we let $\deg^-_M(v)$ denote the in-degree of vertex v in M, i.e., the number of arcs in M entering v, and let $\deg^+_M(v)$ denote the out-degree of vertex v in M, i.e., the number of arcs in M leaving v. If for all vertices v, $\deg^+_M(v) \geq k$ and $\deg^-_M(v) \geq k$, then we call M a $(\geq k)$*-matching.*

2 Approximation Algorithms

In this section we describe the three main algorithms and their combinations that we consider in our experimental study. First, we describe a simple heuristic that computes a minimal k-VCSS (Section 2.1). It considers one arc of the digraph at a time and decides whether to include it in the computed k-VCSS or not. This heuristic computes a 2-approximation of the smallest k-VCSS. Next we consider the algorithm of Cheriyan and Thurimella that computes a $(1 + 1/k)$-approximation of the smallest k-VCSS, and present a simplified version of this algorithm for the $k = 2$ case (Section 2.2). Despite the vast literature on connectivity and network design problems, the above two algorithms are, to the best of our knowledge, the only previously known algorithms for approximating the 2-VCSS of a digraph. (We refer to [4] for results on the undirected or edge-connected versions of the problem.) Then, we present our linear-time heuristic (Section 2.3) and two hybrid algorithms that are derived by combining the linear-time heuristic with either the minimal k-VCSS or the Cheriyan-Thurimella algorithm (Section 2.4).

The quality of the k-VCSS computed by any algorithm can be bounded by the following simple fact: In any k-vertex (or arc) connected digraph each vertex has outdegree $\geq k$. Thus any k-vertex (arc) connected digraph has at least kn arcs. We use this to bound the approximation ratio of the fast heuristic of Section 2.3. We note, however, that the approximation ratio achieved by the algorithm of Cheriyan and Thurimella [4] is based on deeper lower bounds on the number of arcs in k-connected digraphs.

2.1 Minimal *k*-VCSS

Results of Edmonds [7] and Mader [20] imply that every minimal k-VCSS has at most $2kn$ arcs [4]. This fact implies that the following simple heuristic guarantees a 2-approximation of the smallest k-VCSS. We process the arcs in an arbitrary order, and while doing so we maintain a current subgraph $\widehat{G}$ of G. Initially $\widehat{G} = G$. When we process an arc (x, y) we test if $\widehat{G}$ contains at least $k+1$ vertex-disjoint paths from x to y (including the arc (x, y)). If this is the case then we can set $\widehat{G} \leftarrow \widehat{G} - (x, y)$. At the end of this procedure $\widehat{G}$ is a minimal k-VCSS of G, i.e., for any arc $(x, y) \in A(\widehat{G})$, $\widehat{G} - (x, y)$ is not k-vertex connected.

Testing if a digraph G has $k + 1$ vertex-disjoint paths from x to y can be carried out efficiently, for constant k, by computing $k + 1$ arc-disjoint paths in a derived digraph G_{der} using a flow-augmenting algorithm [1]. The vertex set $V(G_{\text{der}})$ contains a pair of vertices v_- and v_+ for each vertex $v \in V(G)$. The arc set $A(G_{\text{der}})$ contains the arcs (v_-, v_+) corresponding to all $v \in V(G)$. Also, for each arc (v, w) in $A(G)$, $A(G_{\text{der}})$ contains the arc (v_+, w_-). A single flow-augmentation step in G_{der} takes $O(m)$ time, therefore the $k + 1$ arc-disjoint x-y paths are constructed in $O(km)$, and the overall running time of the algorithm is $O(km^2)$.

In the $k = 2$ case, if we wish to avoid the construction of the derived graph, we can replace the flow-augmenting algorithm with an algorithm that computes dominators in $G - (x, y)$ with x as the source vertex. The arc (x, y) can be removed if and only if y is not dominated by any vertex other than x and y. (See Section 2.3 and also [14].) We refer to the resulting algorithm as MINIMAL.

2.2 The Cheriyan-Thurimella Algorithm

Cheriyan and Thurimella [4] proposed an elegant algorithm that achieves an $(1 + 1/k)$-approximation of the smallest k-VCSS. The algorithm consists of the following two phases:

Phase 1. This phase computes a minimum $(\geq k-1)-$matching M of the input digraph G. This is transformed to a b-matching problem on a bipartite graph B associated with G. The bipartite graph is constructed as follows. For each vertex $v \in V$ there is a pair of vertices v_- and v_+ in $V(B)$. For each arc (v, w) in $A(G)$ there is an edge $\{v_+, w_-\}$ in $E(B)$. The problem of computing M in G is equivalent to computing a minimum $(\geq k-1)$-matching M_B in B. This, in turn, is equivalent to computing a maximum b-matching M'_B in B, where $b(v) = \deg_{E(B)}(v) - (k-1)$, since $M_B = E(B) \setminus M'_B$. A b-matching problem on a graph with n vertices and m edges can be solved in $O(\sqrt{m\alpha(m,m)\log m}\;\; m\log m)$ time [9].

Phase 2. The second phase runs the minimal k-VCSS algorithm of Section 2.1 but only for the arcs in $A(G) \setminus M$. The algorithm maintains a current graph $\widehat{G}$. Initially we set $A(\widehat{G}) = A$, and at the end of this phase $A(\widehat{G}) = M \cup F$, where $F \subseteq A(G) \setminus M$ is a minimal subset of arcs such that $\widehat{G}$ is k-vertex connected. Using the algorithm of Section 2.1, phase 2 takes $O(k|E|^2)$ time, which is also the asymptotic running time of the whole algorithm.

Simplification for $k = 2$. Although the Cheriyan-Thurimella algorithm is conceptually simple (but its analysis is intricate), it is challenging to provide an efficient implementation, especially for phase 1. For the case $k = 2$ we propose the following simpler implementation of phase 1. First we compute a maximum matching M in B. Then we augment this matching to a set M_2 by adding an edge incident to each free vertex. Next we show that M_2 is indeed a minimum (≥ 1)-matching in B.

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

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

Consider now $X \neq \emptyset$. Let x be any vertex in X. Then, for any edge $\{x, y\}$ in M', $\deg_{M'}(y) = 1$. Otherwise, $M' - \{x, y\}$ is a (≥ 1)-matching in B which contradicts the fact that M' is minimum. Therefore, there is no edge $\{x, y\} \in M'$ such that both x and y are in X. Let N be a subset of M' that is left after removing $\deg_{M'}(x) - 1$ edges for each $x \in X$. Suppose that ℓ edges are removed from M' to form N. Then $|N| = |M'| - \ell$ and B has ℓ free vertices with respect to N. We show that N is a maximum matching in B. Suppose, for contradiction, that it is not. Let M be a maximum matching. Then, for some $\ell' \geq 1$ we have $|M| = |N| + \ell' = |M'| + (\ell' - \ell)$. Next note that there are $\ell - 2\ell'$ free vertices with respect to M. Therefore $|M_2| \leq |M| + (\ell - 2\ell') = |M'| + \ell' - \ell + \ell - 2\ell' = |M'| - \ell' < |M'|$, a contradiction. So $|M| = |N|$ which implies $|M_2| = |M'|$. $\square$

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

2.3 Linear-Time Algorithm

Now we present a fast heuristic which is based on recent work on independent spanning trees [14], testing 2-vertex connectivity [12], and computing strong articulation points [18] in digraphs. Our algorithm uses the concept of dominators and semi-dominators. Semi-dominators were introduced by Lengauer and Tarjan [19] in their fast algorithm for computing dominators in flowgraphs.

A flowgraph $G(s) = (V, A, s)$ is a directed graph with a distinguished source vertex $s \in V$ such that every vertex is reachable from s. A vertex w *dominates* a vertex v if every path from s to v includes w. By Menger's theorem (see, e.g., [6]), it follows that if G is 2-vertex connected then, for any $x \in V$, $G(x)$ has only trivial dominators, meaning that any vertex in $G(x)$ is dominated only by itself and x.

Let D be a depth-first search tree of $G(s)$, rooted at s. We assign to each vertex a preorder number with respect to D and identify the vertices by their preorder numbers. Then, $u < v$ means that u was visited before v during the depth-first search. A path $P = (u = v_0, v_1, \dots, v_{k-1}, v_k = v)$ is a *semi-dominator path* (abbreviated sdom-path) if $v < v_i$ for $1 \le i \le k-1$. The *semi-dominator* of vertex v is defined by

$$\mathit{sdom}(v) = \min\{u \mid \text{there is an sdom-path from } u \text{ to } v\}.$$

From the properties of depth-first search it follows that, for every $v \neq s$, $\mathit{sdom}(v)$ is a proper ancestor of v in D [19].

For any vertex $v \neq s$, we define $t(v)$ to be a predecessor of v that belongs to an sdom-path from $\mathit{sdom}(v)$ to v. Such vertices can be found easily during the computation of semi-dominators [14]. Therefore, by [3], we can compute $t(v)$ for all $v \neq s$ in linear time. Next, we define the *minimal equivalent sub-flowgraph* of $G(s)$, $\min(G(s)) = (V, A(\min(G(s))), s)$, where

$$A(\min(G(s))) = \big\{(p(v), v) \mid v \in V - s\big\} \cup \big\{(t(v), v) \mid v \in V - s\big\}.$$

The following lemma from [14] shows that $G(s)$ and $\min(G(s))$ are equivalent with respect to the dominance relation.

Lemma 2 ([14]). *The flowgraphs $G(s)$ and $\min(G(s))$ have the same dominators.*

Let $G_{\not s}$ denote the graph that remains after deleting s from G. Since G is 2-vertex connected then $G_{\not s}$ is strongly connected. For any digraph H, let H^r denote the digraph that is formed after reversing all arc directions in H; we apply the same notation for arc sets. In particular, the notation $\min(G^r(s))$ refers to the minimal equivalent sub-flowgraph of $G^r(s)$.

The algorithm constructs a subgraph $G^* = (V, A^*)$ of G as follows. First, it chooses two arbitrary vertices $s, s' \in V$, where $s' \neq s$. Then, it computes $\min(G(s))$, $\min(G^r(s))$, a spanning out-tree $T_{\not s}$ of $G_{\not s}(s')$ (e.g., a depth-first search tree of $G_{\not s}$, rooted at s') and a spanning out-tree $T'_{\not s}$ of $G^r_{\not s}(s')$. Then, it sets

$$A^* = A(\min(G(s))) \cup A^r(\min(G^r(s))) \cup A(T_{\not s}) \cup A^r(T'_{\not s}).$$

Theorem 1. *G^* is 2-vertex connected.*

Proof. From [12, 18], we have that a digraph H is 2-vertex connected if and only if it satisfies the following property: $H(s)$ and $H^r(s)$ have trivial dominators only, and $H_{\not s}$ is strongly connected, where $s \in V(H)$ is arbitrary. Lemma 2, implies that G^* satisfies the above property by construction. Therefore, G^* is 2-vertex connected. □

An alternative way to express the construction of G^* is via independent spanning trees [14]. Two spanning trees, T_1 and T_2 of a flowgraph $G(s)$ are independent if, for all vertices $x \neq s$, the paths from s to x in T_1 and T_2 have only s and x in common. In [14] it is shown that $\min(G(s))$ is formed by two independent spanning trees of $G(s)$. Therefore, G^* is formed by two independent spanning trees of $G(s)$, two independent spanning trees of $G^r(s)$ (with their arcs reversed), a spanning tree of $G_{\not s}(s')$, and a spanning tree of $G^r_{\not s}(s')$ (with its arcs reversed).

Theorem 2. *G^* is a 3-approximation of the smallest 2VCSS.*

Proof. Since G^* is formed by 4 spanning trees on n vertices and 2 spanning trees on $n-1$ vertices, we have $|A(G^*)| \leq 4(n-1) + 2(n-2)$. (The inequality holds because the spanning trees may have some arcs in common.) In any 2-vertex connected digraph each vertex has outdegree at least 2, so a 2-vertex connected digraph has at least $2n$ arcs. These facts imply an approximation ratio of at most $3 - \frac{4}{n}$. □

Practical Implementation. The experimental results presented in Section 3 show that in fact our algorithm performs much better than the bound of Theorem 2 suggests, especially after taking the following measures. First, when we have computed $\min(G(s))$ and $\min(G^r(s))$ the algorithms checks if $A(\min(G(s))) \cup A^r(\min(G^r(s)))$ already forms a 2-vertex connected digraph. If this is the case then it returns this graph instead of G^*, therefore achieving a 2-approximation of the smallest 2-VCSS. Another idea that helps in practice is to run the algorithm from several different source vertices and return the best result. For our experiments we used five sources, chosen at random. In all of our experiments, the returned 2-VCSS was formed only by the set $A(\min(G(s))) \cup A^r(\min(G^r(s)))$ for one of the five randomly chosen sources s. (We note that for some choices of s the above set did not suffice to produce a 2-VCSS.) We leave as open question whether there is a tighter bound on the approximation ratio of our heuristic, or whether some variant of the heuristic can achieve better than a 2-approximation.

The above algorithm runs in $O(m+n)$ time using a linear-time algorithm to compute semi-dominators [2, 3, 13], or in $O(m\alpha(m,n))$ time using the Lengauer-Tarjan algorithm [19]. (Here $\alpha(m,n)$ is a very slow-growing functional inverse of Ackermann's function.) Previous experimental studies (e.g., [15]) have shown that in practice the simple version of the Lengauer-Tarjan algorithm, with $O(m \log n)$ worst-case running time, outperforms the $O(m\alpha(m, n))$-time version. In view of these experimental results, our implementation of the above heuristic, referred to as FAST, was based on the simple version of Lengauer-Tarjan.

2.4 Hybrid Algorithms

We can get various hybrid algorithms by combining FAST with either MINIMAL or CT. A first idea is to run MINIMAL with the 2-VCSS returned by FAST as input. We call this algorithm FASTMINIMAL. Since this algorithm also computes a minimal 2-VCSS, it also achieves a 2-approximation. The running time is improved from $O(m^2)$ to $O(n^2)$.

The above idea does not work for CT, since FAST may filter out the wrong arcs and, therefore, the final 2-VCSS may not be a 1.5-approximation. To fix this problem we propose the following combination of FAST and CT, referred to as FASTCT, which runs FAST between the two phases of CT. Following the computation of the (≥ 1)-matching M, we run FAST with the initial digraph G as input. This returns a 2-VCSS G^* of G. The input to the second phase of CT is the digraph with arc set $A(G^*) \cup M$.

Theorem 3. *Algorithm* FASTCT *computes a* 1.5-*approximation of the smallest 2-VCSS in* $O(m\sqrt{n} + n^2)$ *time.*

Proof. After having computed a minimum $(\geq 1)-$matching M of the input digraph G, the Cheriyan-Thurimella algorithm can process the arcs in $A(G) \setminus M$ in an arbitrary order. So if it processes the arcs in $A(G) \setminus \big(A(G^*) \cup M\big)$ first, all these arcs will be removed from the current graph $\widehat{G}$ that is maintained during the second phase. Thus, the approximation guarantee of the Cheriyan-Thurimella algorithm is preserved.

Regarding the running time, by Lemma 1 we have that a minimum $(\geq 1)-$matching can be computed in $O(m\sqrt{n})$ time using the Hopcroft-Karp maximum bipartite matching algorithm [17]. Also, by Section 2.3, the computation of $A(G^*)$ takes linear time. Then, we are left with a 2-VCSS with $O(n)$ arcs, so the last phase of the algorithm runs in $O(n^2)$ time. $\square$

3 Experimental Results

3.1 Implementation and Experimental Setup.

For the first phase of CT and FASTCT, we used an implementation of the push-relabel maximum-flow algorithm of Goldberg and Tarjan [16] from [5]. This implementation does not use a dynamic tree data structure [22], which means that the worst-case bound for the first phase of these algorithms is $O(n^3)$. However, as we confirmed experimentally, the push-relabel algorithm runs very fast in practice. For the implementation of FAST, as well as for the last phase of CT, MINIMAL, and FASTCT, we adapted the implementation of the simple version of the Lengauer-Tarjan dominators algorithm [19] from [15]. The source code was compiled using `g++` v. 3.2.2 with full optimization (flag `-O4`). All tests were conducted on an Intel Xeon E5520 at 2.27GHz with 8192KB cache, running Ubuntu 9.04 Server Edition.

The main focus of the experiments in this paper is the quality of the 2-VCSS returned by each algorithm. Therefore we did not put much effort in optimizing our source code. (Note, however, that the implementations in [5] and [15] are highly optimized.) For completeness we do mention the running times for a subset of the experiments, but leave a thorough experimental study of the running times for the full version of the paper.

3.2 Instances

We considered three types of instances: bicliques, internet peer-to-peer networks taken from the Stanford Large Network Dataset Collection [23], and random digraphs. Some of these graphs are not 2-vertex connected to begin with, so we need to apply a suitable transformation to make them 2-vertex connected. We tested two transformations: adding a bidirectional Hamiltonian cycle (indicated by the letter H in the tables below), and removing strong articulation points (indicated by the letter A in the tables below).

The term bidirectional Hamiltonian cycle refers to two oppositely directed Hamiltonian cycles. We create these cycles by taking a random permutation of the vertices of the input digraph, say $v_0, v_1, \ldots, v_{n-1}$, and adding the arcs (v_i, v_{i+1}) and (v_{i+1}, v_i), where the addition is computed mod n. Note that a bidirectional Hamiltonian cycle is by itself a minimum 2-VCSS with exactly $2n$ arcs, so in this case it is easy to assess how close to optimal are the 2-VCSS produced by the tested algorithms.

A strong articulation point of a strongly connected digraph is a vertex whose removal increases the number of strongly connected components [18]. To apply the second transformation method, we first introduce a directed Hamiltonian cycle to the input digraph if it is not strongly connected. Then we compute the strong articulation points of the resulting digraph and add appropriate arcs so that the final digraph is 2-vertex connected. More details are given below.

Bicliques. Graphs that consist of a complete bipartite graph (biclique) together with a Hamiltonian cycle were suggested in [4] as instances for which getting a better than a 2-approximation of the smallest k-VCSS may be hard. We let $\mathsf{bcH}(n_1, n_2)$ denote a graph that consists of a biclique K_{n_1,n_2} plus a Hamiltonian cycle; we convert this graph to a digraph by making all edges bidirectional, i.e. replace each edge $\{x, y\}$ with the arcs (x, y) and (y, x).

We remark that the order in which the arcs are read from the input file can affect the quality of the returned 2-VCSS significantly. To that end we considered the following example using $\mathsf{bcH}(1000, 2)$ as input. If the arcs of the Hamiltonian cycle are read first then FAST, running from an appropriately chosen source vertex, returns a 2-VCSS with 2004 arcs, which is the optimal value. On the other hand, MINIMAL returns 4000 arcs. The situation is reversed if the arcs of the Hamiltonian cycle are read last, i.e., FAST returns 4000 arcs and MINIMAL returns 2004 arcs. The cause of this effect is that FAST favors the arcs that are explored first during a depth-first search, while MINIMAL favors the arcs that are processed last. The number of arcs that FASTCT returns is 2005 for both graphs. Also, FASTMINIMAL returns 2004 arcs for the first graph and 2006 for the second.

In view of the above effect, we created several instances of each digraph bcH(n_1, n_2) used in our experiments, by permuting randomly its arcs. The results are shown in Table 1.

Table 1. Bicliques with embedded bidirectional Hamiltonian cycle. Average number of arcs and standard deviation in the computed 2-VCSS for 20 seeds. The best result in each row is marked in bold.

INSTANCE	CT	FASTCT	FAST	MINIMAL	FASTMINIMAL
bcH$(1000, 2)$	2664.75	**2045.65**	2458.00	2791.65	2061.90
	16.12	38.29	95.95	17.54	41.13
bcH$(1000, 10)$	2925.90	**2132.40**	2771.75	3558.50	2199.50
	7.85	37.77	57.65	14.78	36.27
bcH$(1000, 20)$	2977.00	**2159.55**	2916.80	3752.90	2230.90
	5.30	20.44	19.90	12.34	27.90
bcH$(1000, 40)$	3020.00	**2227.00**	3048.50	3871.40	2315.50
	4.27	20.25	23.91	11.07	14.97

Internet Peer-to-Peer Networks. We tested digraphs of the family p2p-Gnutella taken from the Stanford Large Network Dataset Collection [23], after performing some preprocessing. These graphs are internet peer-to-peer networks, and therefore relevant for problems concerning the design of resilient communication networks. Also, their size is small enough so that they could be processed within a few hours by the slower algorithms in our study.

The goal of preprocessing was to make these digraphs 2-vertex connected. (Most of these graphs are not strongly connected, and some are not even weakly connected.) To that end we applied the two transformations mentioned earlier, indicated in Table 2 by the letters H and A. The former indicates that a bidirectional Hamiltonian cycle was added to the digraph. The latter transformation first adds a unidirectional Hamiltonian cycle, producing a strongly connected digraph, say G'. Then it chooses two random source vertices r_1 and r_2 and computes strong articulation points (cut vertices) in G' using dominator and post-dominator (i.e., dominator in the reverse digraph) computations with r_1 and r_2 as roots. (We refer to [12, 18] for details.) A strong articulation point x is removed by adding one of the arcs (r_1, x), (x, r_1), (r_2, x) or (x, r_2) depending on which dominators calculation revealed the specific articulation point.

Table 2. Internet peer-to-peer networks; the number of vertices n and the average number of arcs $\overline{m}$ for each type of instances is shown. Average number of arcs and standard deviation in the computed 2-VCSS for 20 seeds. The best result in each row is marked in bold.

INSTANCE	CT	FASTCT	FAST	MINIMAL	FASTMINIMAL
p2p-Gnutella04-H	23461.80	23340.10	30024.10	23488.10	**23254.55**
$(n = 10876, \overline{m} = 61692)$	23.54	17.37	35.18	21.74	29.86
p2p-Gnutella04-A	28203.35	**27929.60**	33920.60	28131.05	28055.95
$(n = 10876, \overline{m} = 63882.50)$	304.09	38.64	414.36	288.35	102.18
p2p-Gnutella08-H	13561.30	13459.60	17048.15	13584.95	**13395.55**
$(n = 6301, \overline{m} = 33137)$	14.23	15.65	35.00	14.00	15.45
p2p-Gnutella08-A	16858.20	**16727.85**	19357.65	16859.20	16778.45
$(n = 6301, \overline{m} = 33913.35)$	140.13	15.82	198.89	153.26	36.96
p2p-Gnutella09-H	17410.75	17321.40	21922.30	17449.70	**17232.60**
$n = 8114, \overline{m} = 41977$	14.49	18.28	35.55	21.28	20.95
p2p-Gnutella09-A	21740.70	**21606.25**	25076.60	21714.80	21679.15
$n = 8114, \overline{m} = 44081.70$	141.43	14.44	293.33	134.11	42.52

Random Graphs. The last type of digraphs we consider are random directed graphs. We denote by rand(n, m) a random directed graph with parameters n and m; n is the number of vertices and m is the expected number of arcs, i.e., each of the $n(n-1)$ possible arcs is chosen with probability $\frac{m}{n(n-1)}$. Since we need to construct 2-vertex connected digraphs, we have to choose large values for m. In order to get a view of the situation for smaller values of m we also include the family of graphs randH(n, m), which contain a bidirectional Hamiltonian cycle. The results are shown in Table 3. Table 4 gives us an indication of the corresponding running times.

Table 3. Random directed graphs. Average number of arcs and standard deviation in the computed 2-VCSS for 20 seeds. The best result in each row is marked in bold.

INSTANCE	CT	FASTCT	FAST	MINIMAL	FASTMINIMAL
rand$(1000, 20000)$	2301.40	2167.25	2991.65	2222.65	**2164.00**
	7.41	9.57	13.03	11.09	10.23
rand$(1000, 40000)$	2336.85	2167.95	2997.10	2224.70	**2163.55**
	10.58	9.46	11.63	7.73	9.20
randH$(1000, 2000)$	2165.65	2149.75	2694.10	2167.00	**2145.95**
	7.04	7.80	12.94	9.22	5.35
randH$(1000, 4000)$	2205.80	**2159.25**	2851.45	2200.15	2159.30
	8.13	8.55	13.46	8.52	7.79

Table 4. Random directed graphs. Average running time (in seconds) and standard deviation for 20 seeds.

INSTANCE	CT	FASTCT	FAST	MINIMAL	FASTMINIMAL
$\mathsf{rand}(1000, 20000)$	14.17	0.79	0.02	14.63	4.54
	0.25	0.02	0.01	0.19	0.09
$\mathsf{rand}(1000, 40000)$	52.59	0.77	0.02	54.52	8.89
	0.63	0.02	0.00	0.77	0.14
$\mathsf{randH}(1000, 2000)$	0.81	0.70	0.01	1.00	1.00
	0.03	0.02	0.00	0.03	0.03
$\mathsf{randH}(1000, 4000)$	1.65	0.77	0.01	1.79	1.45
	0.04	0.02	0.00	0.04	0.03

3.3 Evaluation

The above experimental results show that although our fast heuristic returned a larger 2-VCSS than the other algorithms in all tested cases (with the exception of bicliques), the approximation ratio of 3 seems pessimistic, at least in practice. Moreover, the hybrid algorithms FASTCT and FASTMINIMAL had the best performance in all of our tests. The sizes of the subgraphs returned by these two algorithms were very close to each other, with FASTCT being slightly better for bicliques, the A-instances of the peer-to-peer networks, and the larger H-instance of the random digraphs.

The comparison between FASTCT and FASTMINIMAL favors the former, since it guarantees a better approximation ratio, and, as Table 4 shows, it can run significantly faster. The reason for this difference in the running times is that the maximum-flow computation (used for bipartite maximum matching) is completed a lot faster than the time it takes FASTMINIMAL to process the arcs of the matching that are excluded from the corresponding processing during the last phase of FASTCT. We will investigate thoroughly the relative running times of different implementations in the full version of the paper.

Acknowledgements We would like to thank Renato Werneck for providing the implementation of the push-relabel maximum-flow algorithm from [5] and for several helpful discussions, and the anonymous referees for some useful comments.

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. D. Delling, A. V. Goldberg, I. Razenshteyn, and R. F. Werneck. Graph partitioning with natural cuts. In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, 2011.
6. R. Diestel. *Graph Theory*. Springer-Verlag, New York, second edition, 2000.
7. J. Edmonds. Edge-disjoint branchings. *Combinatorial Algorithms*, pages 91–96, 1972.
8. H. N. Gabow and S. Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, Proc. 19th ACM-SIAM Symp. on Discrete Algorithms, pages 550–559, 2008.
9. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38:815–853, 1991.
10. 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.
11. N. Garg, V. S. Santosh, and A. Singla. Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pages 103–111, 1993.
12. L. Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. In *Proc. 37th Int'l. Coll. on Automata, Languages, and Programming*, pages 738–749, 2010.
13. L. Georgiadis and R. E. Tarjan. Finding dominators revisited. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, pages 862–871, 2004.
14. L. Georgiadis and R. E. Tarjan. Dominator tree verification and vertex-disjoint paths. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms*, pages 433–442, 2005.
15. L. Georgiadis, R. E. Tarjan, and R. F. Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications (JGAA)*, 10(1):69–94, 2006.
16. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35:921–940, October 1988.
17. J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.
18. G. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. In *Combinatorial Optimization and Applications*, volume 6508 of *Lecture Notes in Computer Science*, pages 157–169. Springer Berlin / Heidelberg, 2010.
19. T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.
20. W. Mader. Minimal n-fach zusammenhängende digraphen. *Journal of Combinatorial Theory, Series B*, 38(2):102–117, 1985.
21. Z. Nutov. An almost $O(\log k)$-approximation for k-connected subgraphs. In *Proc. 20th ACM-SIAM Symp. on Discrete Algorithms*, SODA '09, pages 912–921, 2009.
22. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.
23. Stanford network analysis platform (snap). `http://snap.stanford.edu/`.