Strong Articulation Points and Strong Bridges in Large Scale Graphs

Conference Paper *in* Algorithmica · June 2012

DOI: 10.1007/978-3-642-30850-5_18

5 authors, including:

Giuseppe F. Italiano
University of Rome Tor Vergata
155 PUBLICATIONS **3,511** CITATIONS
SEE PROFILE

Luigi Laura
Sapienza University of Rome
114 PUBLICATIONS **1,272** CITATIONS
SEE PROFILE

Alessio Orlandi
Università di Pisa
8 PUBLICATIONS **103** CITATIONS
SEE PROFILE

Federico Santaroni
University of Rome Tor Vergata
9 PUBLICATIONS **127** CITATIONS
SEE PROFILE

Computing Strong Articulation Points and Strong Bridges in Large Scale Graphs[⋆]

Donatella Firmani[1], Giuseppe F. Italiano[2], Luigi Laura[1], Alessio Orlandi[3], and Federico Santaroni[2]

[1] Dept. of Computer Science and Systems, Sapienza Univ. of Rome
via Ariosto, 25 - 00185 Roma, Italy. {firmani,laura}@dis.uniroma1.it
[2] Dept. of Computer Science, Systems and Production, Univ. of Rome "Tor Vergata"
via del Politecnico 1 - 00133 Roma, Italy. {italiano,santaroni}@disp.uniroma2.it
[3] Dept. of Computer Science, Univ. of Pisa
Largo Bruno Pontecorvo 3, 56127 Pisa, Italy. aorlandi@di.unipi.it

Abstract. Let $G = (V, E)$ be a directed graph. A vertex $v \in V$ (respectively an edge $e \in E$) is a *strong articulation point* (respectively a *strong bridge*) if its removal increases the number of strongly connected components of G. We implement and engineer the linear-time algorithms in [9] for computing all the strong articulation points and all the strong bridges of a directed graph. Our implementations are tested against real-world graphs taken from several application domains, including social networks, communication graphs, web graphs, peer2peer networks and product co-purchase graphs. The algorithms implemented turn out to be very efficient in practice, and are able to run on large scale graphs, i.e., on graphs with ten million vertices and half billion edges. Our experiments on such graphs highlight some properties of strong articulation points, which might be of independent interest.

Keywords: graph algorithms, strong connectivity, strong articulation points, strong bridges, large scale graphs.

1 Introduction

Let $G = (V, E)$ be a directed graph. A vertex $v \in V$ is a *strong articulation point* if its removal increases the number of strongly connected components of G. Similarly, an edge $e \in E$ is a *strong bridge* if its removal increases the number of strongly connected components of G (see Figure 1). Note that strong articulation points and strong bridges are related to the notion of 2-vertex and 2-edge connectivity of directed graphs. We recall that a strongly connected graph G is said to be 2-vertex-connected if the removal of any vertex leaves G strongly connected; similarly, a strongly connected graph G is said to be 2-edge-connected if the removal of any edge leaves G strongly connected. The strong articulation points are exactly the vertex cuts for 2-vertex connectivity, while the strong

[⋆] This work has been partially supported by the 7th Framework Programme of the EU (Network of Excellence "EuroNF: Anticipating the Network of the Future - From Theory to Design") and by MIUR, the Italian Ministry of Education, University and Research, under Project AlgoDEEP.

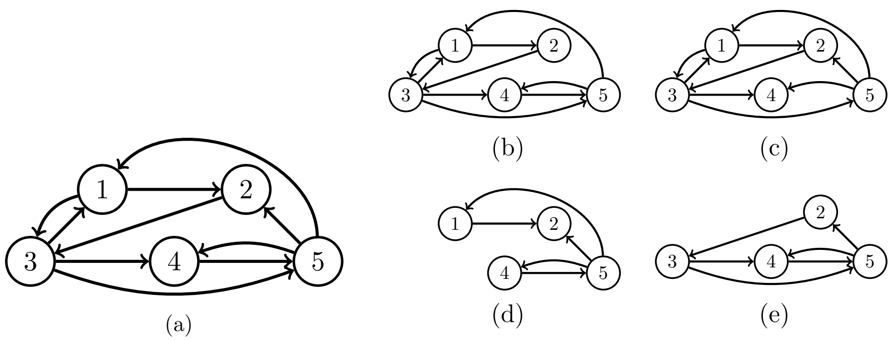

Fig. 1. (a) A strongly connected graph $G = (V, E)$. (b) Edge $(5, 2)$ is **not** a strong bridge, while (c) edge $(4, 5)$ is a strong bridge. (d) Vertex 3 is a strong articulation point, while (e) vertex 1 is **not** a strong articulation point. The strong articulation points in G are vertices 3 and 5, and the strong bridges in G are edges $(2, 3)$ and $(4, 5)$.

bridges are exactly the edge cuts for 2-edge connectivity: G is 2-vertex-connected (respectively 2-edge-connected) if and only if G does not contain any strong articulation point (respectively strong bridge). Surprisingly, the study of 2-vertex and 2-edge connectivity in directed graphs seems to have been overlooked and it only received attention quite recently. Although there is no specific linear-time algorithm in the literature, the 2-edge connectivity of a directed graph can be tested in $O(m+n)$: one can check whether a directed graph is 2-edge-connected by using Tarjan's algorithm [14] to compute two edge-disjoint spanning trees in combination with the disjoint set-union algorithm of Gabow and Tarjan [6]. For testing 2-vertex connectivity, there is a very recent linear-time algorithm of Georgiadis [7]. Note that none of the above algorithms finds *all* the strong articulation points or *all* the strong bridges of a directed graph.

There are certain applications, however, when one is interested in computing *all* the strong articulation points or *all* the strong bridges of a directed graph. This includes the identification of cores in directed social networks [11], filtering algorithms for the tree constraint in constrained programming [2], and verifying the restricted edge connectivity of strongly connected graphs [15]. Linear-time algorithms for computing all the strong bridges and all the strong articulation points of a directed graph were recently proposed in [9]. In this paper, we implement and engineer the algorithms in [9] and test our implementations against real-world graphs taken from several application domains, including social networks, communication graphs (such as email graphs), web graphs, peer2peer networks and product co-purchase graphs. Our implementations appear to be fast and memory-efficient in practice, and this makes it possible to compute all the strong articulation points and the strong bridges of large scale graphs, namely graphs up to twenty million vertices and half billion edges, in a dozen minutes. In addition, our experiments highlight some properties of strong articulation points, which might help to further characterize the structure of large scale real-world graphs, and in particular to identify cores in social networks. We next describe briefly some of the observations that arise from our experiments.

First, strong articulation points appear frequently in real-world graphs: indeed, most of the graphs in our datasets have quite a high number of strong articulation points. As an example, between 15% and 25% of the vertices in product co-purchase graphs are strong articulation points, while in social graphs this figure ranges between 11% and 18%. The relative number of strong bridges is much smaller, with the only notable exception of email graphs, where strong bridges are about 10% of the total number of edges. Another interesting property is that in our dataset the vast majority of the strong articulation points and the strong bridges tend to be inside the largest strongly connected component. As a further indication of their importance, we mention that the indegree, outdegree and PageRank of the strong articulation points tend to be much higher than average in communication graphs and substantially higher than average in social and web graphs.

2 Graph Terminology

We assume that the reader is familiar with the standard graph terminology, as contained for instance in [5]. Let $G = (V, E)$ be a directed graph, with m edges and n vertices. A *directed path* in G is a sequence of vertices v_1, v_2, $\ldots$, v_k, such that edge $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k-1$. A directed graph G is *strongly connected* if there is a directed path from each vertex in the graph to every other vertex. The *strongly connected components* of G are its maximal strongly connected subgraphs. A directed graph G^t is said to be a *transitive reduction* of G if (i) G^t has a directed path from vertex u to vertex v if and only if G has a directed path from vertex u to vertex v, and (ii) there is no graph with fewer edges than G^t satisfying condition (i). Given a directed graph $G = (V, E)$, its *reversal graph* $G^R = (V, E^R)$ is defined by reversing all edges of G: namely, G^R has the same vertex set as G and for each edge (u, v) in G there is an edge (v, u) in G^R. We say that the edge (v, u) in G^R is the *reversal* of edge (u, v) in G.

A flowgraph $G(s) = (V, E, s)$ is a directed graph with a *start vertex* $s \in V$ such that every vertex in V is reachable from s. The *dominance relation* in $G(s)$ is defined as follows: a vertex u is a *dominator* of vertex v if every path from vertex s to vertex v contains vertex u. Let $dom(v)$ be the set of dominators of v. Clearly, $dom(s) = \{s\}$ and for any $v \neq s$ we have that $\{s, v\} \subseteq dom(v)$: we say that s and v are the *trivial dominators* of v in the flowgraph $G(s)$. The dominance relation is transitive and its transitive reduction is referred to as the *dominator tree* $DT(s)$. Note that the dominator tree $DT(s)$ is rooted at vertex s. Furthermore, vertex u dominates vertex v if and only if u is an ancestor of v in $DT(s)$. We say that u is an *immediate dominator* of v if u is a dominator of v, and every other non-trivial dominator of v also dominates u. It is known that if a vertex v has any non-trivial dominators, then v has a unique immediate dominator: the immediate dominator of v is the parent of v in the dominator tree $DT(s)$. In the following, we denote by $D(s)$ the set of non-trivial dominators in $G(s)$. Let $G^R = (V, E^R)$ be the reversal graph of G, and let $G^R(s) = (V, E^R, s)$ be the flowgraph with start vertex s: we denote by $D^R(s)$ the set of non-trivial dominators in $G^R(s)$.

Similarly, we say that an edge (u, v) is an *edge dominator* of vertex w if every path from vertex s to vertex w contains edge (u, v). Furthermore, if (u, v) is an edge dominator of w, and every other edge dominator of u also dominates w, we say that (u, v) is an *immediate edge dominator* of w. Similar to the notion of dominators, if a vertex has any edge dominators, then it has a unique immediate edge dominator. As before, we denote by $DE(s)$ the set of edge dominators in $G(s)$ and by $DE^R(s)$ the set of edge dominators in $G^R(s)$.

3 Experimental Setup

Test Environment. All our experiments were performed on a machine with a CPU Intel Xeon X5650 with 6 cores, running at 2.67GHz, with 12MB of cache and 32GB RAM DDR3 at 1GHz. The operating system was Linux Red Hat 4.1.2-46, with kernel version 2.6.18, Java Virtual Machine version 1.6.0_16 (64-Bit) and WebGraph library version 3.0.1. Our implementations have been written in Java, in order to exploit the features offered by the WebGraph library [3], which has been designed especially to deal with large graphs. Our code is available upon request. All the running times reported in our experiments were averaged over ten different runs.

Datasets. In our experiments we considered several large-scale real-world graphs, with up to ten million vertices and half billion edges. Our graphs come from different application areas, including web graphs, communication networks, peer-to-peer networks, social networks and product co-purchase graphs. Vertices of a co-purchase graph correspond to products, while edges connect commonly co-purchased products (i.e., there is a directed edge from x to y when users who bought product x also bought product y). The graphs considered in our experiments, together with their type, information about the repository where the graph was taken, number of vertices (n) and edges (m), average vertex degree (δ_{avg}), are listed in Table 1. As already mentioned, we represent the graphs using the WebGraph file format, storing both the original and the reversal of a graph in order to access the adjacency lists in both directions. All the graphs taken from other repositories (such as the SNAP graphs) were converted into this format before performing the experiments. For lack of space, we refer the interested reader to the repositories SNAP [12] and WebGraph [16] for an explanation and more information about those graph datasets and their file formats.

4 Algorithms for Strong Articulation Points and Strong Bridges

In this section we describe algorithms for computing strong bridges and strong articulation points for strongly connected graphs. This is without loss of generality, since the strong bridges (respectively strong articulation points) of a directed graph G are given by the union of the strong bridges (respectively strong articulation points) of the strongly connected components of G.

Graph	**Type**	**Repository**	n	m	δ_{avg}
`p2p-Gnutella04`	Peer2peer	SNAP	11 K	40 K	3.7
`wiki-Vote`	Social	SNAP	7 K	103 K	14.5
`enron`	Communication	WebGraph	69 K	276 K	4.0
`email-EuAll`	Communication	SNAP	265 K	420 K	1.58
`soc-Epinions1`	Social	SNAP	76 K	509 K	30.5
`soc-Slashdot0811`	Social	SNAP	77 K	905 K	11.7
`soc-Slashdot0902`	Social	SNAP	82 K	948 K	11.5
`amazon0302`	Product co-purchase	SNAP	262 K	1.2 M	4.7
`web-NotreDame`	Web	WebGraph	325 K	1.4M	4.6
`uk-2007-05@100K`	Web	WebGraph	100 K	3 M	30.5
`cnr-2000`	Web	WebGraph	325 K	3.2 M	9.8
`amazon0312`	Product co-purchase	SNAP	400 K	3.2 M	8.0
`amazon-2008`	Product co-purchase	SNAP	735 K	5.1 M	7.0
`wiki-Talk`	Communication	SNAP	2.3 M	5.0 M	2.1
`web-Google`	Web	SNAP	875 K	5.1 M	5.8
`web-BerkStan`	Web	SNAP	685 K	7.6 M	11.0
`in-2004`	Web	WebGraph	1.3 M	16.9 M	12.2
`eu-2005`	Web	WebGraph	862 K	19.2 M	22.3
`uk-2007-05@1M`	Web	WebGraph	1 M	41.2 M	41.2
`soc-LiveJournal1`	Social	SNAP	4.8 M	68.9 M	14.2
`ljournal-2008`	Social	SNAP	5.3 M	79.0 M	14.7
`indochina-2004`	Web	WebGraph	7.4 M	194 M	26.1
`uk-2002`	Web	WebGraph	18.5M	298 M	16.1
`arabic-2005`	Web	WebGraph	22.7 M	640 M	28.1

Table 1. Real-world graphs in our experiments, sorted by number of edges (m).

There are trivial algorithms that compute all the strong articulation points and all the strong bridges of a graph G in $O(n(m+n))$ time. To compute whether a vertex v is a strong articulation point, it is enough to check whether the graph $G \setminus v$ is strongly connected. This yields an $O(n(m+n))$ time algorithm for computing all strong articulation points. To compute all strong bridges in $O(n(m+n))$ time is less immediate, and we describe next how to accomplish this task. Fix any vertex v of G. Let $T^+(v)$ be an out-branching rooted at v, i.e., a directed spanning tree rooted at v with all edges directed away from v. Similarly, let $T^-(v)$ be an in-branching rooted at v, i.e., a directed spanning tree rooted at v with all edges directed towards v. Note that $G' = T^+(v) \cup T^-(v)$ is not unique. However, as shown in [9], every graph G' produced in this way contains at most $(2n-2)$ edges, can be computed in $O(m+n)$ time and includes all the strong bridges of G. This implies that all the strong bridges of G can be computed in $O(n(m+n))$ time by simply computing a graph $G' = T^+(v) \cup T^-(v)$ from G and checking for each edge e of G' whether the graph $G \setminus e$ is strongly connected.

Throughout this paper, we refer to both trivial $O(n(m+n))$ algorithms as `Naive`, and in particular we refer to the naive algorithm for computing strong articulation points (respectively strong bridges) as `Naive(SAP)` (respectively `Naive(SB)`). To obtain their linear-time algorithms, Italiano et al. [9] exploited

```
1. Choose arbitrarily a vertex $s \in V$ in $G$, and test whether $s$ is a strong articulation
   point in $G$. If $s$ is a strong articulation point, output $s$.
2. Compute $D(s)$, the set of non-trivial dominators in the flowgraph $G(s) = (V, E, s)$.
3. Compute the reversal graph $G^R = (V, E^R)$.
4. Compute $D^R(s)$, the set of non-trivial dominators in the flowgraph $G^R(s) = (V, E^R, s)$.
5. Output $D(s) \cup D^R(s)$.
```

Fig. 2. Algorithm `ILS(SAP)` for computing all strong articulation points of G

```
1. Choose arbitrarily a vertex $s \in V$ in $G$.
2. Compute $DE(s)$, the set of edge dominators in the flowgraph $G(s) = (V, E, s)$.
3. Compute the reversal graph $G^R = (V, E^R)$.
4. Compute $DE^R(s)$, the set of edge dominators in the flowgraph $G^R(s) = (V, E^R, s)$.
5. Output $DE(s) \cup DE^R(s)$.
```

Fig. 3. Algorithm `ILS(SB)` for computing all strong bridges of G

the relationship between strong articulation points, strong bridges and dominators in flowgraphs stated in the following theorem:

Theorem 1. [9] *Let $G = (V, E)$ be a strongly connected graph, and let $s \in V$ be any vertex in G. Then vertex $v \neq s$ is a strong articulation point in G if and only if $v \in D(s) \cup D^R(s)$. Furthermore, edge (u, v) is a strong bridge in G if and only if $(u, v) \in DE(s)$ or $(v, u) \in DE^R(s)$.*

We implemented the algorithms for computing all strong articulation points and all strong bridges which stem directly from Theorem 1, as described respectively in Figures 2 and 3. We call both algorithms `ILS`, and we refer to the algorithm for computing strong articulation points (respectively strong bridges) as `ILS(SAP)` (respectively `ILS(SB)`). We next show how to compute dominators and edge dominators, which are the building blocks for the `ILS` algorithms.

Computing Dominators. Several algorithms for computing dominators in flowgraphs have been proposed in the literature. The algorithm by Lengauer and Tarjan [10], which we refer to as `LT`, runs in $O(m\alpha(m, n))$ worst-case time, where $\alpha(m, n)$ is a slowly growing inverse Ackermann function. Later on, Alstrup et al. [1] designed linear-time solutions for computing dominators. As observed in the experimental study of Georgiadis et al. [8], this linear-time solution is "significantly more complex and thus unlikely to be faster than `LT` in practice". Georgiadis et al. [8] proposed a hybrid algorithm, dubbed `semi-NCA`, which despite being slower from the theoretical viewpoint (it runs in $O(n^2 + m \log n)$ time), appears to be very efficient in practice. In our study, we rewrote the implementations of `LT` and `semi-NCA` within the WebGraph framework [3] and performed some experiments in order to select the best method for computing dominators in our framework. We confirm on a larger scale the experimental findings of Georgiadis et al. [8], as the running times of `LT` and `semi-NCA` are

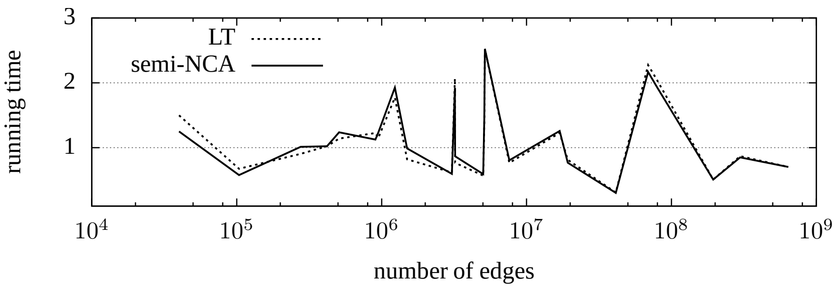

Fig. 4. Experimental comparison of `LT` and `semi-NCA`. The number of edges is shown in logarithmic scale and the running times (in microsecs) are normalized to the number of edges.

very close in practice. The result of one such experiment on graphs in our dataset is illustrated in Figure 4. Since both algorithms appear to be comparable, and there is no clear winner, in the remainder of the paper we will only show the results of experiments where the dominator trees are computed by one of them, in particular `LT`.

Computing Edge Dominators. Our algorithm for computing edge dominators in flowgraphs hinges on the following lemma by Tarjan:

Lemma 1. [13] *Let $G = (V, E, s)$ be a flowgraph and let T be a DFS tree of G with start vertex s. Edge (v, w) is an edge dominator in G if and only if all of the following conditions are met: (v, w) is a tree edge, w has no entering forward edge or cross edge, and there is no back edge (x, w) such that w does not dominate x.*

From the algorithmic viewpoint, the only non-trivial part of this lemma is to check whether w dominates x. To do this, it is enough to test whether w is an ancestor of x in the dominator tree $DT(s)$. We can accomplish this task in constant time, once $DT(s)$ is available together with the preordering and postordering numbers produced by a depth-first search visit of $DT(s)$. Recall that the preordering number $\mathit{first}(v)$ is given by the order in which vertex v was *first* visited, while the postordering number $\mathit{last}(v)$ is given by the order in which vertex v was *last* visited. Now, it is easy to see that w dominates x if and only if $\mathit{first}(w) < \mathit{first}(x) < \mathit{last}(x) < \mathit{last}(w)$. Our algorithm for computing edge dominators requires all the extra work implied by Lemma 1 in addition to the computation of dominator trees. Thus, we expect that in practice `ILS(SB)` will be slightly slower than `ILS(SAP)`. This was confirmed by our experiments.

5 Experimental Results

In this section we report the results of our experimental evaluation. To check when they become faster than the simple-minded `Naive` algorithms, we performed some experiments with small graphs. As it can be seen from Table 2, the `ILS` algorithms appear to be superior to the `Naive` algorithms even for very small graphs (few hundred vertices and thousand edges), and they become substantially faster for larger graphs. This shows that the ILS implementations seem to be the method of choice even in the case of very small graphs.

Table 3 illustrates the results of another experiment with large scale graphs, where the Naive algorithms are omitted due to their extremely high running times. In our experiments the ILS algorithms appeared to be very fast in practice: quite surprisingly, they were able to handle large graphs (hundred million edges) only in about a dozen minutes. As expected, ILS(SAP) was slightly faster than ILS(SB), since computing edge dominators is slightly more complicated than computing dominators (see Lemma 1).

The efficiency of our ILS implementations makes it possible to compute the strong articulation points and the strong bridges of large graphs, and allows us to try to characterize some of their main properties, as shown in Table 3. The data collected in our experiments suggest that in the graphs considered most of the strong articulation points are in the largest strongly connected component. This seems to be true especially for peer2peer, social and product co-purchase graphs. For instance in p2p-Gnutella04 all the strong articulation points and all strong bridges are inside the largest strongly connected component, even if this component contains only about 40% of the vertices and 47% of the edges. For web graphs, however, there are few instances (such as uk-2007-05@100k and web-BerkStan) where a susbstantial fraction of the strong articulation points lie outside of the largest strongly connected component. Similar properties hold for strong bridges, with the only exception of few instances of social graphs, namely soc-Slashdot0811 and soc-Slashdot0902, where, differently from strong articulation points, more than 90% of the strong bridges lie outside of the largest strongly connected component.

In general, our experiments show that strong articulation points appear frequently in our datasets. Their actual frequency seems to depend on the type of the graph considered: product co-purchase graphs seem to have the highest percentage of strong articulation points (between 15% and 25% of their vertices are strong articulation points), followed closely by social graphs (where between 11% and 18% of the vertices are strong articulation points). Another interesting aspect is that in our dataset only the email graphs (such as email-EuAll) seem

Graph	n	m	$\#sap$	$\#sb$	ILS(SAP)	Naive(SAP)	ILS(SB)	Naive(SB)
c.elegans	0.3 K	2.3 K	36	45	0.097	0.231	0.067	0.313
political_blogs1	0.6 K	2.7 K	43	84	0.049	0.214	0.064	0.296
political_blogs2	1.0 K	8.2 K	80	154	0.050	0.822	0.068	1.479
political_blogs3	1.4 K	19 K	115	216	0.101	2.049	0.099	4.238
p2p-Gnutella04	11 K	40 K	1344	1674	0.150	29.22	0.180	42.73
wiki-Vote	7 K	103 K	143	152	0.180	12.57	0.180	24.77
enron	69 K	276 K	796	4967	0.580	371.39	0.690	700.37

Table 2. ILS and Naive running times, measured in seconds. We use $\#sap$ and $\#sb$ to denote respectively the number of strong articulation points and of strong bridges in the graph.

Graph	n	m	$\#sap$	$\#sb$	n_{scc}	m_{scc}	$\#sap_{scc}$	$\#sb_{scc}$	`ILS(SAP)`	`ILS(SB)`
`p2p-Gnutella04`	11 K	40 K	1.3 K	1.6 K	4.3 K	18.7 K	1.3 K	1.6 K	0.15	0.18
`wiki-Vote`	7 K	103 K	143	152	1.3 K	39.4 K	143	152	0.18	0.18
`enron`	69 K	276 K	796	4.9 K	8.2 K	147 K	781	4.8 K	0.58	0.69
`email-EuAll`	265 K	420 K	962	46.0 K	34 K	151 K	960	46.0 K	1.32	1.80
`soc-Epinions1`	76 K	509 K	8.4 K	23.5 K	32 K	443 K	8.1 K	20.9 K	1.07	1.20
`soc-Slashdot0811`	77 K	905 K	14.0 K	417	70 K	888 K	13.0 K	3	1.76	2.19
`soc-Slashdot0902`	82 K	948 K	14.2 K	501	71 K	912 K	14.1 K	69	1.79	2.35
`amazon0302`	262 K	1.2 M	71.9 K	75.7 K	241 K	11.3 M	69.6 K	73.3 K	3.55	4.77
`web-NotreDame`	325 K	1.4 M	13.8 K	61.8 K	54 K	304 K	9.6 K	31.9 K	2.48	3.27
`uk-2007-05@100K`	100 K	3 M	9.4 K	47.0 K	53 K	1.6 M	2.8 K	16.8 K	3.00	3.46
`cnr-2000`	325 K	3.2 M	32.5 K	104 K	112 K	1.6 M	14.6 K	44.1 K	4.28	5.08
`amazon0312`	400 K	3.2 M	69.5 K	83.2 K	380 K	3.0 M	69.0 K	82.6 K	11.37	12.40
`amazon-2008`	735 K	5.1 M	103 K	159 K	627 K	4.7 M	102 K	156 K	25.81	21.89
`wiki-Talk`	2.3 M	5.0 M	14.8 K	86.7 K	111 K	14.7 M	14.8 K	85.5 K	19.02	18.58
`web-Google`	875 K	5.1 M	102 K	267 K	434 K	3.4 M	89.8 K	211 K	13.59	15.48
`web-BerkStan`	685 K	7.6 M	108 K	297 K	334 K	4.5 M	53.6 K	164 K	9.91	12.15
`in-2004`	1.3 M	16.9 M	82.0 K	421 K	480 K	7.8 M	33.5 K	216 K	32.39	39.02
`eu-2005`	862 K	19.2 M	104 K	160 K	752 K	17.9 M	99.3 K	146 K	23.95	27.67
`uk-2007-05@1M`	1 M	41.2 M	147 K	415 K	593 K	22.0 M	82.5 K	259 K	20.90	24.51
`soc-LiveJournal1`	4.8 M	68.9 M	654 K	1.3 M	3.8 M	65.8 M	649 K	1.3 M	260.03	273.60
`ljournal-2008`	5.3 M	79.0 M	734 K	1.3 M	4.1 M	74.9 M	727 K	1.3 M	275.53	299.57
`indochina-2004`	7.4 M	194 M	774 K	2.2 M	3.8 M	98.8 M	503 K	1.4 M	155.83	192.06
`uk-2002`	18.5 M	298 M	2.3 M	6.1 M	12.0 M	232 M	1.8 M	4.8 M	404.92	478.13
`arabic-2005`	22.7 M	640 M	2.7 M	6.7 M	15.1 M	473 M	2.2 M	5.2 M	681.47	837.89

Table 3. Analysis of strong articulation points and bridges, and corresponding `ILS` running time (in seconds), for graphs from 40 K to 640 M edges. We denote by n_{scc} (resp. m_{scc}) the number of vertices (resp. edges) in the largest strongly connected component of the graph, and $\#sap_{scc}$ and $\#sb_{scc}$ to denote the number of strong articulation points and of strong bridges in that component.

to have a large number of strong bridges, which are again mostly contained in the largest strongly connected component.

To further check what are the bottlenecks of the `ILS` algorithms and whether there could be room for further improvements, we broke down their running times into smaller subtasks. In particular, Figure 5 shows the results of such an experiment with `ILS(SAP)`, where we measured the running time of (i) loading the graph G into memory; (ii) computing the strongly connected components of G; (iii) computing the dominators with `LT` and (iv) testing whether the root r is a strong articulation point (by simply checking whether the strongly connected component containing r breaks down after the removal of r). Note that all the remaining running times are accounted for in a generic subtask called "other".

As shown in Figure 5, most of the graphs of the same type (peer2peer, communication, product co-purchase, social and web graphs) tend to share the same break down structure in their running times. As expected, for most of the graphs the bottleneck of `ILS(SAP)` appears to be the computation of dominators (via the `LT` algorithm), which nevertheless seems to be slower than the computation of the strongly connected components only by a small constant factor. On the other hand, checking whether the root is a strong articulation point is consistently faster than computing the strongly connected components of the entire graph. The most notable exception to this general behavior arises from the communication graphs, such as `enron`, `email-EuAll` and `wiki-Talk`. This happens since those graphs tend to have a much different structure from the other graphs

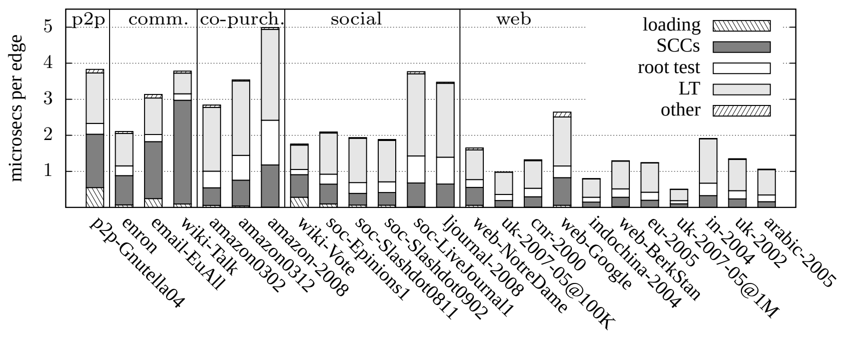

Fig. 5. Running times of `ILS(SAP)`, broken down in the following subtasks: 1) loading the graph G into memory, 2) computing the strongly connected components of G, 3) computing the dominators with `LT` and 4) testing whether the root r is a strong articulation point. All the remaining running times are accounted for in a generic subtask called "other". The input graphs are partitioned according to their type (peer2peer, communication, product co-purchase, social and web graphs).

in our dataset: indeed, they consist of one big strongly connected component, containing slightly more than 10% of the vertices, and a very large number of tiny strongly connected components (each having less than about 10 vertices). In such a case, `LT` and the root test run on much smaller inputs (i.e., the strongly connected components) than the entire graph.

Similar results are obtained for strong bridges. Figure 6 shows the results of such an experiment with `ILS(SB)`, where we measured the running time of (i) loading the graph G into memory; (ii) computing the strongly connected components of G and (iii) computing the edge dominators. As with strong articulation points, all the remaining running times are accounted for in a generic subtask called "other". The main difference is that now the computation of edge dominators requires more time than the computation of dominators, as implied by Lemma 1. All those experiments seem to suggest that in order to obtain substantially faster `ILS` implementations, one should be able to produce faster codes for computing dominators.

To investigate further properties of strong articulation points, we computed average indegree, outdegree and PageRank of strong articulation points and of vertices in our graph instances. The result of such experiment is shown in Table 4. While in co-purchase graphs strong articulation points seem to have vertex degrees and PageRank close to average, there are more striking differences for the other graphs in our dataset. In particular, the indegree, outdegree and PageRank of the strong articulation points tend to be much higher than average in communication graphs and higher than average in social and web graphs.

Another application where strong articulation points may be handy is the identification of cores in directed social networks. As defined in [11], a core is a minimal set of vertices which are necessary for the connectivity of the network, i.e., removing vertices in the core breaks the remainder of the vertices into many small, disconnected strongly connected components. In recent work, Mislove et al. [11], following an approximation commonly used in web graph analysis [4],

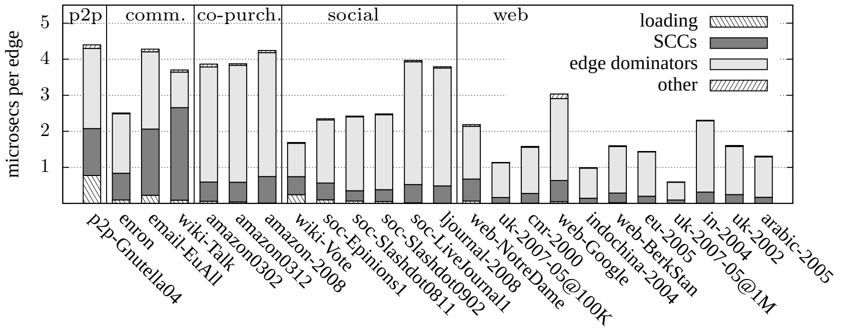

Fig. 6. Running times of `ILS(SB)`, broken down in the following subtasks: 1) loading the graph G into memory, 2) computing the strongly connected components of G and 3) computing the edge dominators. All the remaining running times are accounted for in a generic subtask called "other". The input graphs are partitioned according to their type (peer2peer, communication, product co-purchase, social and web graphs).

	δ^-_{avg}		δ^+_{avg}		PR_{avg}		$\#sap_{scc}$	$\#sap_{scc}$ in	$\#sap_{scc}$ in
Graph	V	sap	V	sap	V	sap		$\delta^-(10\%)$	$\delta^+(10\%)$
`p2p-Gnutella04`	3.68	4.87	3.68	9.60	$9.19{\cdot}10^{-5}$	$1.12{\cdot}10^{-4}$	1.3 K	132	69
`enron`	3.99	62.79	3.99	103.48	$1.46{\cdot}10^{-5}$	$1.63{\cdot}10^{-4}$	781	24	8
`email-EuAll`	1.58	280.43	1.58	103.06	$3.80{\cdot}10^{-6}$	$4.29{\cdot}10^{-4}$	960	3	5
`wiki-Talk`	2.1	69.35	2.10	290.50	$4.18{\cdot}10^{-7}$	$2.86{\cdot}10^{-6}$	14.8 K	53	152
`amazon0302`	4.71	4.65	4.71	4.89	$3.82{\cdot}10^{-6}$	$3.42{\cdot}10^{-6}$	69.6 K	2497	4801
`amazon0312`	7.99	6.81	7.99	8.65	$2.50{\cdot}10^{-6}$	$2.06{\cdot}10^{-6}$	69.0 K	998	3628
`amazon-2008`	7.02	8.21	7.02	8.97	$1.36{\cdot}10^{-6}$	$2.05{\cdot}10^{-6}$	102 K	374	1825
`wiki-Vote`	12.5	79.29	12.50	68.91	$1.21{\cdot}10^{-4}$	$6.83{\cdot}10^{-4}$	143	3	11
`soc-Epinions1`	6.71	34.29	6.71	32.47	$1.32{\cdot}10^{-5}$	$5.83{\cdot}10^{-5}$	8.1 K	89	90
`soc-Slashdot0811`	11.7	38.73	11.70	39.90	$1.21{\cdot}10^{-5}$	$3.06{\cdot}10^{-5}$	13.0 K	1	3
`soc-Slashdot0902`	11.54	39.08	11.54	40.48	$1.16{\cdot}10^{-5}$	$2.92{\cdot}10^{-5}$	14.1 K	2	5
`soc-LiveJournal1`	14.23	35.89	14.23	35.47	$2.07{\cdot}10^{-7}$	$5.69{\cdot}10^{-7}$	649 K	3057	3113
`ljournal-2008`	14.73	38.46	14.73	37.56	$1.88{\cdot}10^{-7}$	$5.10{\cdot}10^{-7}$	727 K	3729	3666
`web-NotreDame`	4.6	14.13	4.60	13.63	$3.11{\cdot}10^{-6}$	$1.25{\cdot}10^{-5}$	9.6 K	797	478
`uk-2007-05@100K`	30.51	139.95	30.51	46.40	$1.01{\cdot}10^{-5}$	$4.32{\cdot}10^{-5}$	2.8 K	24	153
`cnr-2000`	9.88	27.59	9.88	16.79	$3.12{\cdot}10^{-6}$	$9.08{\cdot}10^{-6}$	14.6 K	1587	741
`web-BerkStan`	11.09	21.08	11.09	10.87	$1.46{\cdot}10^{-6}$	$3.36{\cdot}10^{-6}$	53.6 K	1640	4179
`web-Google`	5.57	17.80	5.57	9.24	$1.09{\cdot}10^{-6}$	$3.30{\cdot}10^{-6}$	89.8 K	5150	4037
`in-2004`	41.25	222.67	41.25	45.78	$1.01{\cdot}10^{-6}$	$5.06{\cdot}10^{-6}$	33.5 K	2641	2907
`eu-2005`	22.3	49.35	22.30	25.20	$1.16{\cdot}10^{-6}$	$2.99{\cdot}10^{-6}$	99.3 K	10441	12928
`uk-2007-05@1M`	12.23	41.71	12.23	19.61	$7.32{\cdot}10^{-7}$	$2.09{\cdot}10^{-6}$	82.5 K	5825	8360
`indochina-2004`	26.18	62.23	26.18	27.60	$1.37{\cdot}10^{-7}$	$4.09{\cdot}10^{-7}$	503 K	30252	38724
`uk-2002`	16.1	43.93	16.10	20.82	$5.48{\cdot}10^{-8}$	$1.43{\cdot}10^{-7}$	1.8 M	152773	183997
`arabic-2005`	28.14	82.68	28.14	34.96	$4.44{\cdot}10^{-8}$	$1.31{\cdot}10^{-7}$	2.2 M	139907	176214

Table 4. For each graph, in this table we report, distinguished between the vertices (V) and the strong articulation points (sap), the average values of indegree (δ^-_{avg}), outdegree (δ^+_{avg}) and PageRank (PR_{avg}); in the last three columns we can see, respectively, the number of the strong articulation points in the main strongly connected components (sap_{scc}) and, amongst these, the one that belongs to the top 10% vertices sorted by indegree ($\delta^-(10\%)$) and by outdegree ($\delta^+(10\%)$).

observed that after removing 10% of the highest indegree (or highest outdegree) vertices in a social graph, the largest strongly connected component will be split into smaller components. It seems thus natural to ask how many of the removed vertices are actually strong articulation points, i.e., how many of the removed vertices are really effective in splitting the largest strongly connected component. Table 4 reports the results of an experiment where we tried to answer this question. As shown in the last columns of Table 4, only few strong articulation points are selected by this process, which indeed seems to miss the vast majority of strong articulation points (roughly 95% on average). This gives some evidence that using strong articulation points in place of high degree vertices may provide a better approximation of the core of a directed graph.

Acknowledgments

We are indebted to Massimo Bernaschi for providing the computing infrastructure on which our experiments were run. We thank also Loukas Georgiadis for helpful discussions and for providing the source code of the implementation of `LT` and `semi-NCA` described in [8], and the anonymous referees for useful comments.

References

1. S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM J. Comput.*, 28(6):2117–2132, 1999.
2. N. Beldiceanu, P. Flener, and X. Lorca. The tree constraint. *Proc. CPAIOR 2005*, 64–78.
3. P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. 13th Int. World Wide Web Conference (WWW 2004)*, 595–601.
4. A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd edition.* MIT Press, 2009.
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 – 221, 1985.
7. L. Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. *Proc. 37th ICALP*, 433–442, 2010.
8. L. Georgiadis, R. E. Tarjan, and R. F. F. Werneck. Finding dominators in practice. *J. Graph Algorithms Appl.*, 10(1):69–94, 2006.
9. G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science.* To appear.
10. T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.
11. A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. *Proc. 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, 29–42, 2007.
12. SNAP: Stanford Network Analysis Project. `http://snap.stanford.edu/`.
13. R. E. Tarjan. Edge-disjoint spanning trees, dominators, and depth-first search. Technical report, Stanford, CA, USA, 1974.
14. R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Inf.*, 6:171–185, 1976.
15. L. Volkmann. Restricted arc-connectivity of digraphs. *Inf. Process. Lett.*, 103(6):234–239, 2007.
16. The WebGraph Framework Home Page. `http://webgraph.dsi.unimi.it/`.