Datastream computation of graph biconnectivity: Articulation Points, Bridges, and Biconnected Components

G. Ausiello  D. Firmani  L. Laura

Dipartimento di Informatica e Sistemistica
Sapienza University of Rome
Via Ariosto, 25. 00185 Rome, Italy.

April 16, 2010
The connectivity is the basis of the structural analysis of a graph.

In the traditional offline setting the problem dates back to the 70s. In the on-line setting, the first algorithms have been addressed in 1989.

We propose the first algorithm that computes all the (bi)connectivity properties of an undirected graph, in the streaming model.
Statement of the problem

We can solve the following problem...

**Problem**

Given a streaming graph $G$, represented by a stream of its edges $S = e_1, e_2 \ldots e_m$ (in any order), the goal is to compute all its (bi)connectivity properties: connected components (CCs), articulation points, bridges, and biconnected components (BCCs).

- **INPUT.** a stream of edges;
- **OUTPUT.** CCs, APs, Bridges, BCCs.
Statement of the problem

...in the datastream framework.

Definition

In the **datastream framework**, as in the on-line framework, the items arrive one after the other, but there are stricter requirements concerning the memory occupation and the allowed per item processing time (PIPT), that should be small enough to allow real-time processing.

- your working memory cannot contain the input;
- if an item takes too time, you can miss the following one.
Definition of (bi)connectivity properties

**Definition**

Given a graph $G = (V, E)$, we define:

- **CC.** $V' \subseteq V$ s.t. at least one path joining $u, v \in V'$ exists;
- **bridge.** $e \in E$ s.t. its removal increases number of CCs;
- **articulation point.** $v \in V$ s.t. its removal increases number of CCs;
- **BCC.** a subgraph $G''$, induced by $V'' \subseteq V$, such that i) $G''$ is a CC, and ii) $G''$ is a CC also if any single vertex is removed from it.

**Figure:** A graph with 2 CCs, 4 APs, 2 BRs, and 4 BCCs.
Outline

1. Introduction
2. Preliminaries and Statement of the Problem
3. Related Work
4. The Algorithm: At First Look (AFL)
5. Complexity
6. Experimental Results
7. Conclusions
Related works: Datastreaming

Related streaming models are:

- **classical streaming model.** Munro and Paterson [7]:
  - memory $O(\log n)$ (with respect to the length $n$ of the stream);
  - too strict for basic graph problems such as connectivity.

  ⇒ **semi-streaming model.** Feigenbaum [5] and Muthukrishnan [8]:
  - memory $O(n \cdot \log n)$ (allows to store nodes but not edges);
  - works on $t$-spanners [2, 4, 6] and articulation points [5];

Other models are:

- **stream-sort model.** Aggarwal et al. [1].
- **w-stream model.** Demetrescu et al. [3];
Related works: Biconnectivity

Algorithms by Westbrook and Tarjan [9] to find on-line bridge-connected and biconnected components:

- both optimal time $O(n + m\alpha(m, n))$;
- sophisticated data structure, called link/condense tree;
- missing an experimental study.

We propose a different solution inspired by the problem to find bridges and APs in the ASes;

- the first requirement was to make a query on a link and respond in $O(1)$;
- the second requirement was having a simple sketch to track properties.
Outline

1. Introduction
2. Preliminaries and Statement of the Problem
3. Related Work
4. The Algorithm: At First Look (AFL)
5. Complexity
6. Experimental Results
7. Conclusions
The main idea behind the AFL algorithm is to keep in memory an object that we call *navigational sketch* (NS) of a graph G.

A NS is a graph (forest) $\text{NS} = (V_{ns}, E_{ns})$, where:

- the set of **nodes** contains all the nodes of $G$;
- the set of **edges** contains two types of edges:
  1. **solid edges.** Real edges of the graph $G$;
  2. **coloured edges.** Representative of *biconnected components*.
- the following property holds.

![Diagram](image)

**Figure:** A navigational sketch of the example graph.
The following correspondences between $G$ and NS hold true:

1. **CCs.** Maximal trees in the $NS$;
2. **bridges.** Solid edges of the $NS$;
3. **BCCs.** Subtree, inside a tree in the $NS$, with one father and $b - 1$ children (where $b$ is the cardinality of the biconnected component), where all the edges are of the same color, and this color is unique inside the $NS$.

**Figure:** A navigational sketch of the example graph.
The Navigational Sketch

We define articulation points in $NS$ with the colour degree of a node $i$:

- $d_c(i)$ is the number of incident solid edges plus the number of distinct colours of incident coloured edges.

**Property**

The following correspondence between $G$ and $NS$ holds true:

1. **APs.** Nodes $i$ for which it holds $d_c(i) > 1$.

**Figure:** A navigational sketch of the example graph.
How to build and maintain the $NS$

At each step the algorithm looks at the current edge $(u,v)$ from the stream and, at first look, it decides the corresponding action to be executed on $NS$:

1. **it joins two trees.** Unite them with a *solid edge*;
2. **it joins nodes in the same tree.** Another path besides the one in $NS$.

In case (2) we look at the edges in the (unique) path in in $NS$ joining $u$ and $v$ and update the tree:

1. **same coloured edges.** Drop the edge;
2. **solid or different coloured edges.** Unite the BCCs “touched” by the path.
Example

<table>
<thead>
<tr>
<th>CASE</th>
<th>CURRENT NS</th>
<th>EDGE</th>
<th>UPDATED NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
<tr>
<td>2(a)</td>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
<tr>
<td>2(b)</td>
<td><img src="image7.png" alt="Diagram" /></td>
<td><img src="image8.png" alt="Diagram" /></td>
<td><img src="image9.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Figure:** Example of the three cases of Algorithm AFL.
We proved the correctness of the AFL algorithm, demonstrating that it builds a valid NS for the graph as seen until the current item.

The (bi)connectivity properties of the graph therefore represent invariants of the AFL algorithm.
Per item processing time

Logic operations on $F$:
1. find nodes in the same tree;
2. join trees;
3. find same coloured edges;
4. join sets of edges;
5. find paths.

Basic operations:
- 1 and 2 $\rightarrow$ union-find over trees, i.e. CCs;
- 3 and 4 $\rightarrow$ union-find over edge type, i.e. BCCs;
- 5 $\rightarrow$ Least Common Ancestor (LCA).

Theorem
The amortized per item processing time of the algorithm AFL is $O(find + \frac{n-1}{m} union + \frac{n-1}{m} LCA)$. 
### Table: Array data relative to the navigational sketch.

<table>
<thead>
<tr>
<th>Node</th>
<th>Father</th>
<th>BCC Rep</th>
<th>Left Brother</th>
<th>Right Brother</th>
<th>CC Rep.</th>
<th>BCC Size</th>
<th>CC Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Figure: NS and a graphical (pointer) view of the first 4 columns of array data.
Overall processing time

We use the following approaches:

- sequence of at most $n-1$ union and $m$ find $\rightarrow O(n + m\alpha(m, n))$;
- joining $n-1$ CC needs to evert the smaller tree $\rightarrow O(n \log n)$;
- for LCA we go up from nodes marking every visited node $\rightarrow O(d)$.

**Corollary**

The processing time of the algorithm AFL on the entire stream sequence is $O(n \log n + m\alpha(m, n))$.

**Corollary**

Optimal if average degree $\frac{m}{n}$ greater or equal than $\log n$. 
The space needed to store $F$ corresponds to the typical space complexity of the semi-streaming model.

**Lemma**

The space occupation of the algorithm AFL is $O(n \log n)$.

It is the “sweet spot” for graph streaming problems [Muthukrishnan 01], and for the (bi)connectivity problem it is just its space complexity.

**Lemma**

The space occupation of the algorithm AFL is tight.

**Hint.** There are graph instance like trees in which bridges are $n - 1$, if $n$ is the number of nodes...
Outline

1. Introduction
2. Preliminaries and Statement of the Problem
3. Related Work
4. The Algorithm: At First Look (AFL)
5. Complexity
6. Experimental Results
7. Conclusions
Let’s briefly recall AFL algorithm features:

- it takes a graph stream as input;
- it has the following complexity bounds:
  - space \( O(n \log n) \) on a graph with \( n \) nodes;
  - time \( O(m\alpha(m, n) + n \log n) \),
    \( \alpha \) is a functional inverse of Ackermann’s function;
  - PIPT \( O(\alpha(m, n) + \frac{n}{m} \log n) \), (almost) constant amortized.
Testing environment

We tested a C implementation of AFL on a laptop Dell XPS M1330 (4Gb RAM, Intel Core2 Duo T8100 2.1GHz).

Our dataset is composed by:

- **Autonomous System graphs.** Collected from the Route Views project;
- **Web graphs.** Collected using the WebGraph framework.
- **other domain graphs.** Thanks to various provider.

Graphs with different density features, most of them worst-case:

\[ \frac{m}{n} < \log n. \]
Results on the data set

| Graph       | Type          | Disk Space | Number of Nodes: $n = |V|$ | Number of Edges: $m = |E|$ | Average Degree: $\frac{m}{n}$ | Density Factor: $\frac{n \log n}{m}$ | Max # touched edges per operation | Avg. # touched edges per operation | Overall Processing Time (t) | Amortized PIPT: $\frac{t}{m}$ | Edges processed per second: $\frac{m}{t}$ |
|-------------|---------------|------------|----------------------|----------------------|-------------------------------|----------------------------------|----------------------------------|-------------------------------|-------------------------------|-----------------------------|-------------------------------|
| AS          | A.Systems     | 670 Kb     | 57k                  | 57k                  | 0,88                          | 18,24                           | 4                               | 0,77                          | < 0.1                         | $\approx 8.17E-7$             | -                           |
| eatRS       | linguistic    | 3.8 Mb     | 23.2k                | 325.5k               | 14,02                         | 1,03                            | 9                               | 0,06                          | < 0.2                         | $\approx 3.87E-7$             | -                           |
| hep-th-new  | citation      | 4.2 Mb     | 27.7k                | 352.7k               | 12,70                         | 1,16                            | 7                               | 0,07                          | < 0.3                         | $\approx 6.20E-7$             | -                           |
| cnr-2000    | web           | 44.7 Mb    | 325k                 | 3.2M                  | 9,88                          | 1,85                            | 10                              | 0,06                          | < 3                           | $\approx 1.02E-6$             | $\approx 1M$                |
| eu-2005     | web           | 270.8 Mb   | 862k                 | 19.2M                 | 22,3                          | 0,88                            | 7                               | 0,04                          | < 20                          | $\approx 1.06E-6$             | $\approx 1M$                |
| indochina-2004 | web       | 3 Gb       | 7.4M                 | 194.1M                | 26,18                         | 0,87                            | 72                              | 0,03                          | < 200                         | $\approx 1.02E-6$             | $\approx 1M$                |
| uk-2002     | web           | 5 Gb       | 18.5M                | 298.1M                | 16,1                          | 1,50                            | 91                              | 0,05                          | < 300                         | $\approx 1.01E-6$             | $\approx 1M$                |
| it-2004     | web           | 20.5 Gb    | 41.2M                | 1.1G                  | 27,87                         | 0,91                            | 255                             | 0,07                          | < 600                         | $\approx 5.80E-7$             | $\approx 2M$                |

**Table:** Experimental results; time expressed in seconds.

We recall that operations are **amortized**: the execution of a very complex operation make easier the following ones (AFL doesn’t have to do it anymore).
Remarks

Hints

Values of $t/m$ suggest that:

- PIPT is in practice almost constant;
- an off-the-shelf laptop can process up to 1M edge per second.

Different performances depend on the structural properties of the graphs:

- if $F$ tends to “collapse” into a large and short tree, it helps LCA;
- if $BCC$s tend to be “discovered” quickly, many edges are “dropped”.
Outline

1. Introduction
2. Preliminaries and Statement of the Problem
3. Related Work
4. The Algorithm: At First Look (AFL)
5. Complexity
6. Experimental Results
7. Conclusions
A *coloured-tree forest* provides all the (bi)connectivity properties of the corresponding graph, and it can be seen as its “navigational sketch”.

Therefore our approach could be used as a building block in the development of more complex streaming graph algorithms.

THANKS FOR YOUR ATTENTION.
On the streaming model augmented with a sorting primitive.

Graph spanners in the streaming model: An experimental study.
*Algorithmica*, 2009.

C. Demetrescu, I. Finocchi, and A. Ribichini.
Trading off space for passes in graph streaming problems.
In *Proc. of SODA’06*, 2006.

M. Elkin.
Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners.

J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang.
On graph problems in a semi-streaming model.

J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang.
Graph distances in the streaming model: the value of space.
In *Proc. of SODA’05*, 2005.

I. Munro and M. Paterson.
Selection and sorting with limited storage.
S. Muthukrishnan.
Data streams: Algorithms and applications.

J. Westbrook and R.E. Tarjan.
Maintaining bridge-connected and biconnected components on-line.