Incremental 2-Edge-Connectivity In Directed Graphs

Loukas Georgiadis
University of Ioannina
Greece

Giuseppe F. Italiano
University of Rome
Tor Vergata
Italy

Nikos Parotsidis
University of Rome
Tor Vergata
Italy
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problem definition
  - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
Outline

 Definitions
  • 2-edge-connectivity in undirected graphs
  • 2-edge-connectivity in directed graphs
  • Problem definition
  • Known algorithm and our result

 High-level idea

 Basic ingredients
  • Dominators
  • Auxiliary components

 Tools

 Conclusion
Let $G = (V, E)$ be a undirected graph.

- $G$ is connected if there is a path between any two vertices.
- The connected components of $G$ are its maximal connected subgraphs.
Let $G = (V, E)$ be a connected undirected graph.

- An edge is a **bridge**, if its removal increases the number of connected components.
Let $G = (V, E)$ be a connected undirected graph.  
- An edge is a bridge, if its removal increases the number of connected components.
Undirected: Connected components

By Menger’s theorem, two vertices are 2-edge-connected iff the removal of any bridge leaves them in the same connected component.
By Menger’s theorem, two vertices are **2-edge-connected** iff the removal of any **bridge** leaves them in the same **connected component**.
By Menger’s theorem, two vertices are 2-edge-connected iff the removal of any bridge leaves them in the same connected component.

Two vertices are 2-edge-connected if there are two edge-disjoint paths between them.
By Menger’s theorem, two vertices are 2-edge-connected iff the removal of any bridge leaves them in the same connected component.

Two vertices are 2-edge-connected if there are two edge-disjoint paths between them.
By Menger’s theorem, two vertices are 2-edge-connected iff the removal of any bridge leaves them in the same connected component.

Two vertices are 2-edge-connected if there are two edge-disjoint paths between them.

The 2-edge-connected blocks of G are its maximal subsets $B \subseteq V$ s.t. $u$ and $v$ are 2-edge-connected $\forall u, v \in B$
By Menger’s theorem, two vertices are 2-edge-connected iff the removal of any bridge leaves them in the same connected component.

Two vertices are 2-edge-connected if there are two edge-disjoint paths between them.

The 2-edge-connected blocks of $G$ are its maximal subsets $B \subseteq V$ s.t. $u$ and $v$ are 2-edge-connected $\forall u, v \in B$

$O(m + n)$ time algorithm [Tarjan 1972]
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problem definition
  - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
Let $G = (V, E)$ be a directed graph.

- $G$ is strongly connected if there is a directed path from each vertex to every other vertex.
- The strongly connected components of $G$ are its maximal strongly connected subgraphs.
Let $G = (V, E)$ be a directed graph.

- $G$ is strongly connected if there is a directed path from each vertex to every other vertex.
- The strongly connected components of $G$ are its maximal strongly connected subgraphs.
Let $G = (V, E)$ be a strongly connected directed graph.

- An edge $e \in E$ is a strong bridge if its removal increases the strongly connected components of $G$. 
Directed: 2-edge-connectivity

Let $G = (V, E)$ be a strongly connected directed graph.

- An edge $e \in E$ is a strong bridge if its removal increases the strongly connected components of $G$. 
Let $G = (V, E)$ be a strongly connected directed graph.

- An edge $e \in E$ is a **strong bridge** if its removal increases the strongly connected components of $G$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices $u$ and $v$ are 2-edge connected if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are 2-edge connected if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$. 
Directed: 2-edge-connectivity

By Menger's Theorem, vertices $u$ and $v$ are 2-edge connected if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are 2-edge connected if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices $u$ and $v$ are **2-edge connected** if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are **2-edge connected** if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices $u$ and $v$ are 2-edge connected if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are 2-edge connected if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices $u$ and $v$ are 2-edge connected if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are 2-edge connected if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices $u$ and $v$ are 2-edge connected if and only if the removal of any strong bridge leaves them in same strongly connected component.

Vertices $u$ and $v$ are 2-edge connected if there are two edge-disjoint paths from $u$ to $v$ and two edge-disjoint paths from $v$ to $u$.

A 2-edge-connected block of $G$ is a maximal subset $B \subseteq V$ s.t. $u$ and $v$ are 2-edge connected for all $u, v \in B$. 
Directed: 2-edge-connectivity

By Menger’s Theorem, vertices \( u \) and \( v \) are 2-edge connected if and only if the removal of any strong bridge leaves them in the same strongly connected component.

Vertices \( u \) and \( v \) are 2-edge connected if there are two edge-disjoint paths from \( u \) to \( v \) and two edge-disjoint paths from \( v \) to \( u \).

A 2-edge-connected block of \( G \) is a maximal subset \( B \subseteq V \) s. t. \( u \) and \( v \) are 2-edge connected for all \( u, v \in B \).

- \( O(m + n) \) time algorithm
  [Georgiadis, Italiano, Laura, P. 2015]
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problem definition
    - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
Problem definition

Graph + data structure
Problem definition

Graph + data structure

Are $u$ and $v$ 2-edge-connected
Problem definition

Are $u$ and $v$ 2-edge-connected

Yes/No

Graph + data structure

ICALP 2016, Rome
Problem definition

Graph + data structure

What are the 2-edge-connected blocks in $G$
Problem definition

Graph + data structure

What are the 2-edge-connected blocks in $G$

$\{a, b, d\}, \{c, e\}, \{f\}$
Problem definition

Graph + data structure
Problem definition

Graph + data structure

Insert \((x, y)\)
Problem definition

Graph + data structure

Insert (x,y)
Problem definition

Graph + data structure

Are $u$ and $v$ 2-edge-connected
Problem definition

Graph + data structure

Are $u$ and $v$ 2-edge-connected

Yes/No
Problem definition

Graph + data structure

Goal:
Update time faster than recomputing
Fast query time
## Dynamic graph algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>Undirected graphs</th>
<th>Directed graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectivity/Transitive closure</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Connected components/Strongly connected components</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>APSP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DFS tree</td>
<td>Yes</td>
<td>(only on DAGs)</td>
</tr>
<tr>
<td>MST</td>
<td>Yes</td>
<td>?</td>
</tr>
<tr>
<td>2-edge-connectivity</td>
<td>Yes</td>
<td>?</td>
</tr>
<tr>
<td>2-vertex-connectivity</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>
Incremental 2-edge-connectivity
Undirected VS Directed

Block tree structure
Incremental 2-edge-connectivity
Undirected VS Directed

Block tree structure
Incremental 2-edge-connectivity
Undirected VS Directed

Block tree structure
Incremental 2-edge-connectivity
Undirected VS Directed

Block tree structure

No tree structure is possible
Incremental 2-edge-connectivity
Undirected VS Directed

Block tree structure

No tree structure is possible
Incremental 2-edge-connectivity
Undirected VS Directed

- Block tree structure
- No tree structure is possible
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problems definition
  - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
## Simple-minded solutions

<table>
<thead>
<tr>
<th></th>
<th>Update time</th>
<th>Query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never update</td>
<td>$O(1)$ per insertion</td>
<td>$O(m + n)$</td>
</tr>
<tr>
<td>Always update</td>
<td>$O(m + n)$ per insertion</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
## Our algorithm

<table>
<thead>
<tr>
<th></th>
<th>Update time</th>
<th>Query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never update</td>
<td>$O(1)$ per insertion</td>
<td>$O(m + n)$</td>
</tr>
<tr>
<td>Always update</td>
<td>$O(m + n)$ per insertion</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Our algorithm</td>
<td>$O(mn)$ total time</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problems definition
  - Known algorithm and our result

- High-level idea

- Basic ingredients
  - Dominators
  - Auxiliary components

- Tools

- Conclusion
High-level idea

Dominator tree

$O(n)$ time
Labeling algorithm

2-edge-connected blocks

Auxiliary components
Dominators

Flow graph $G(s) = (V, A, s)$: all vertices are reachable from start vertex $s$

$v$ dominates $w$ if all paths from $s$ to $w$ contain $v$

$dom(w) =$ set of vertices that dominate $w$
\( \nu \) dominates \( \omega \) if all paths from \( s \) to \( \omega \) contain \( \nu \)

\[ D(s) = \text{dominator tree of } G(s) \]

\( G(s) \)

\( D(s) \)
Dominators

\( v \) dominates \( w \) if all paths from \( s \) to \( w \) contain \( v \)

\[ G(s) \]

\[ D(s) = \text{dominator tree of } G(s) \]
Dominators

\( \nu \) dominates \( \omega \) if all paths from \( s \) to \( \omega \) contain \( \nu \)

\[ G(s) \]

\[ D(s) = \text{dominator tree of } G(s) \]

ICALP 2016, Rome
Dominators

\[ \nu \text{ dominates } \omega \text{ if all paths from } s \text{ to } \omega \text{ contain } \nu \]

\[ D(s) = \text{dominator tree of } G(s) \]

\[ G(s) \]

\[ D(s) \]

ICALP 2016, Rome
𝑣 dominates 𝑤 if all paths from 𝑠 to 𝑤 contain 𝑣

\[ G(s) \]

\[ D(s) = \text{dominator tree of } G(s) \]

ICALP 2016, Rome
**Dominators**

$v$ dominates $w$ if all paths from $s$ to $w$ contain $v$

$G(s)$

$D(s) = \text{dominator tree of } G(s)$

$v$ dominates $w$ if all paths from $s$ to $w$ contain $v$

$G(s)$

$D(s) = \text{dominator tree of } G(s)$

$\mathcal{O}(ma(m, n))$-time algorithm: [Lengauer and Tarjan ’79]

$\mathcal{O}(m + n)$-time algorithms:

[Alstrup, Harel, Lauridsen, and Thorup ‘97]
Exploiting dominator tree
Exploiting dominator tree

• All paths from $s$ to $c$ contain $l, f, d$
Exploiting dominator tree

- All paths from $s$ to $c$ contain $l, f, d$
- All paths from $s$ to $c$ contain the strong bridges $(s, l), (f, d)$
Exploiting dominator tree

- All paths from $s$ to $c$ contain $l, f, d$
- All paths from $s$ to $c$ contain the strong bridges $(s, l), (f, d)$
- A strong bridge is the only incoming edge to the vertices of its subtree
The **dominator tree** of the graph provides only **partial information**.
Exploiting dominator tree

The **dominator tree** of the graph provides only **partial information**. The **dominator tree of the reverse graph** provides **other** partial information.
Lemma [Georgiadis, Italiano, P.]: Two vertices $u$ and $v$ are 2-edge-connected iff

- Their nearest bridge $e$ in $D$ is common and they are not separated in $G \setminus e$
- Their nearest bridge $e$ in $D^R$ is common and they are not separated in $G \setminus e$
Lemma [Georgiadis, Italiano, P.]: Two vertices $u$ and $v$ are 2-edge-connected iff
- Their nearest bridge $e$ in $D$ is common and they are not separated in $G \setminus e$
- Their nearest bridge $e$ in $D^R$ is common and they are not separated in $G \setminus e$

$c$ and $g$ are not 2-edge-connected since they have distinct nearest bridges.
Exploiting dominator tree

\[ c \text{ and } e \text{ are 2-edge-connected iff they are strongly connected in } G \setminus (f, d) \]

Lemma [Georgiadis, Italiano, P.]: Two vertices \( u \) and \( v \) are 2-edge-connected iff
- Their nearest bridge \( e \) in \( D \) is common and they are not separated in \( G \setminus e \)
- Their nearest bridge \( e \) in \( D^R \) is common and they are not separated in \( G \setminus e \)
Exploiting dominator tree

Lemma [Georgiadis, Italiano, P.]: Two vertices $u$ and $v$ are 2-edge-connected iff

- Their nearest bridge $e$ in $D$ is common and they are not separated in $G \setminus e$
- Their nearest bridge $e$ in $D^R$ is common and they are not separated in $G \setminus e$
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problems definition
  - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
**Auxiliary components**

**Bridge decomposition:** The forest obtained by removing the strong bridges from the dominator tree.

**Lemma:** two vertices are 2-edge-connected **only if** they are in the same tree of the bridge decomposition.
**Bridge decomposition:** The forest obtained by removing the strong bridges from the dominator tree.

**Lemma:** two vertices are 2-edge-connected **only if** they are in the same tree of the bridge decomposition.
Idea: Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

Construction:
• Keep the paths using only vertices of the tree
• For every path using vertices outside the tree, replace the subpath outside with a shortcut edge
**Auxiliary components**

**Idea:** Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

**Construction:**
- Keep the paths using only vertices of the tree
- For every path using vertices outside the tree, replace the subpath outside with a shortcut edge
Auxiliary components

Idea: Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

Construction:
- Keep the paths using only vertices of the tree
- For every path using vertices outside the tree, replace the subpath outside with a shortcut edge
**Auxiliary components**

**Idea:** Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

**Construction:**
- Keep the paths using only vertices of the tree
- For every path using vertices outside the tree, replace the subpath outside with a shortcut edge

**Lemma:** Two vertices in the same tree (rooted at $r$) are disconnected by $(d(r), r)$ iff they are not strongly connected in the auxiliary graph of their tree (in the same auxiliary component).
**Idea:** Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

**Construction:**
- Keep the paths using only vertices of the tree
- For every path using vertices outside the tree, replace the subpath outside with a shortcut edge

**Lemma:** Two vertices in the same tree (rooted at $r$) are disconnected by $(d(r), r)$ iff they are not strongly connected in the auxiliary graph of their tree (in the same auxiliary component).

**Algorithm:** Two vertices $u$ and $v$ are 2-edge-connected iff they are in the same auxiliary component in $G$ and $G^R$. 
**Auxiliary components**

**Idea:** Encode all the paths that do not use the incoming strong bridge between vertices in the same tree of the bridge decomposition with an auxiliary graph.

**Construction:**
- Keep the paths using only vertices of the tree
- For every path using vertices outside the tree, replace the subpath outside with a shortcut edge

**Lemma:** Two vertices in the same tree (rooted at \( r \)) are disconnected by \((d(r), r)\) iff they are not strongly connected in the auxiliary graph of their tree (in the same auxiliary component).

**GOAL:** Incrementally maintain the bridge decomposition and the auxiliary components.
Outline

- Definitions
  - 2-edge-connectivity in undirected graphs
  - 2-edge-connectivity in directed graphs
  - Problems definition
  - Known algorithm and our result
- High-level idea
- Basic ingredients
  - Dominators
  - Auxiliary components
- Tools
- Conclusion
Tools

- Incremental dominator tree
  - 2012 – Georgiadis, Italiano, Laura, Santaroni
    - $O(m \min\{n, k\} + kn)$

- Incremental SCCs in each auxiliary graph
  - 2009 & 2016 – Bender, Fineman, Gilbert, Tarjan:
    - $O\left(m \min\{\sqrt{m}, n^{2/3}\}\right)$
Combining things...

- Many instances of the Incremental SCCs algorithm
- Vertices can move across auxiliary graphs
- Auxiliary graphs can merge
- ...

ICALP 2016, Rome
Concluding remarks

Results:

• **Incremental** $O(mn)$ **algorithm** for maintaining the pairwise **2-edge-connectivity** in directed graphs.

• **Answer queries in** $O(1)$ **time**, whether two vertices are 2-edge-connected. If the two vertices are not 2-edge-connected, we return an edge that separates them.

Open problems:

• Can we maintain incrementally the 2-vertex-connected blocks?

• **Decremental? Fully dynamic?**
Concluding remarks

Results:

• Incremental $O(mn)$ algorithm for maintaining the pairwise 2-edge-connectivity in directed graphs.

• Answer queries in $O(1)$ time, whether two vertices are 2-edge-connected. If the two vertices are not 2-edge-connected, we return an edge that separates them.

Open problems:

• Can we maintain incrementally the 2-vertex-connected blocks?

• Decremental? Fully dynamic?

Thank you!