A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time

by Liam Roditty and Uri Zwick at STOC 04

presented by R. Inkulu
Reachability
Reachability Tree from a Vertex

- **in-tree (1):** SCCs which can reach 1 from vertex 1.
- **out-tree (1):** SCCs which can be reached from vertex 1.

Root vertex: 1
Strongly Connected Component (SCC)

Intra-component edge

Inter-component edge

A

B

C

D

courtesy: CLRS
Reachability Tree among SCCs

The diagram illustrates the reachability tree among strongly connected components (SCCs). The root vertex is labeled as 11, and the in-tree consists of components 1, 2, 3, 4, and 5. The out-tree (1) includes vertices 6, 9, 8, 10, and 4.

The tree structure shows the relationships between the vertices, with arrows indicating the direction of reachability.
SCCs under Arc Insertions

A doesn’t change

C & D merge
SCCs under Arc Deletions

A remains same

B splits into two
Reachability under Arc Insertions and Deletions

Deleting (5,6) causes C to be unreachable from any vertex in B

Inserting (5,8) causes vertices in D to be reachable from C
Overview of the Algorithm

• Fully Dynamic Strong Connectivity
  – Maintaining SCCs under arc insertions and deletions using Component Forest (side effect: persistency)
  – SCC splits/merges from Component Forest
• Decremental Maintenance of Reachability Trees
  – SCC splitting and updating the inter-component Arcs
  – Reconnecting the possibly unconnected tree
• Fully Dynamic Reachability Algorithm
Fully Dynamic Strong Connectivity
Component Forest
Maintaining SCCs under Insertions

G₀

G₁

G₂

G₃

Dynamic Edge Sets

H₁

H₂

H₃

H₄
Maintaining SCCs under Deletions

Dynamic Edge Sets

$G_0$  $G_1$  $G_2$  $G_3$

$H_1$  $H_2$  $H_3$  $H_4$
Finding SCCs in the given Graph

LL[1] = 1
LL[2] = 1
LL[3] = 1
LL[4] = 3
LL[5] = 4
LL[6] = 6
LL[7] = 7
LL[8] = 6

LL[v] – lowest dfs numbered vertex in the same SCC as ‘v’ reachable via a sequence of 0 or more tree Arcs and 0 or 1 cross or back Arcs

courtesy : The design and analysis of computer algorithms by Aho, Hopcroft, Ullman
Analysis of Fully Dynamic SCC Maintenance

(m : # of Arcs  n : # of vertices)

• O(mα(m, n)) worst-case insert operation
• O(mα(m, n)) amortized deletion operation
• O(1) to query whether two vertices belong to the same SCC in a given version of the graph (using LCA algorithm)
• O(|SCC|) to list the vertices of a SCC
• O(m+n) space
Decompositions of SCCs
Analysis of Component Forest

• $O(n)$ amortized time to maintain the pointers from the old to new forest
• $O(1)$ worst-case time to find the SCCs to which one SCC from the previous version got split
Decremental Maintenance of Reachability Trees
Reachability Tree from a Vertex

- **Root vertex:** 1
- **in-tree (1):** SCCs which can reach 1
  - 5
  - 7
- **out-tree (1):** SCCs which can be reached from 1
  - 2
  - 9
  - 8
  - 3

- Other vertices:
  - 4
  - 6
  - 11
  - 3
  - 2
  - 7
  - 10
  - 4
  - 5
  - 10

Numbers 10 and 20 are not part of the graph but are mentioned in the text.
Determining Reachability Out Tree

Root SCC

Active1

Inactive5

Active2

Active4

Active3

Active
Resulting Reachability Out Tree

- **Active1**
  - Active1
  - v1
  - v2
  - v3
  - v4
  - v5
  - v6
  - v7
  - v8
  - v9
  - v10
  - v11
  - v12
  - v13
  - v14
  - v15

- **Inactive5**
  - v11
  - v12

- **Active2**
  - v12
  - v13
  - v14
  - v15
  - v16
  - v17

- **Active3**
  - v16
  - v17

- **Active4**
  - v9
  - v8

- **Root SCC**
  - r
SCC decomposition after Arc Deletions
Reconnecting the disconnected Out Tree
SCC in Decremental Reachability Algorithm

SCC1

Active vertices in SCC1

v1

v2

v3

v4

v5

Inter-component Arcs entering v1

u1

u2

u3

u4
SCC Split in the Decremental Maintenance
Analysis of Decremental Maintenance of Reachability Tree

- $O(m + n \log n)$ in the worst-case
  - $O(m + n \log n)$ in worst-case to maintain the SCC data structures under edge deletions
  - $O(m)$ in worst-case to reconnect the possibly disconnected tree by inspecting every edge only once in each direction
- $O(1)$ worst-case time to check whether the root vertex is a witness to reach vertex $v$ from vertex $u$
Fully Dynamic Reachability Algorithm
In and Out Trees under Insertions

G₀

G₁

G₂

G₃

G₄

R₀

R₁

R₂

R₃

R₄
In and Out Trees under Deletions

\[ G_0', G_1', G_2', G_3', G_4' \]

\[ R_0', R_1', R_2', R_3', R_4' \]

4567
Reachability Trees after recent Insert

Before Delete

After Delete

R₁

R₂

R₁’

R₂’

R₃’
Analysis of Fully Dynamic Reachability Algorithm

- $O(m + n \log n)$ amortized update time
- $O(n)$ worst-case reachability query time
Comparison with the other Deterministic Fully Dynamic Reachability Algorithms

<table>
<thead>
<tr>
<th>Authors</th>
<th>Amortized Update Time</th>
<th>Reachability Query Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[C.Demetrescu, D.Leiserson]</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[L.Roditty]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[L.Roditty, U.Zwick]</td>
<td>$O(m^{\sqrt{n}})$</td>
<td>$O(\sqrt{n})$</td>
</tr>
<tr>
<td>This Paper</td>
<td>$O(m + n \log n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Conclusion

• Concluding Remarks from the paper
  – How to reduce the update time to $O(m)$

• Possible Improvements
  – After finding SCCs under edge insertions/deletions, how about maintaining transitive closure matrices on those SCCs
  – Merge the information from all the reachability trees by constructing a reachability matrix to get better query time (useful when the number of queries after each update operation is relatively large)
  – Rather than maintaining at most $n$ reachability trees, is there any way to be selective in choosing the root vertices so that the total number of reachability trees are reduced without disturbing the correctness of the algorithm
  – Is this applicable in determining bi-connected components??


