Efficient Algorithms for Reachability and Path-Selection Problems

http://www.icte.uowm.gr/lgeorg/RPS/

John S. Latsis
Public Benefit Foundation
Research Projects 2010
Research Team

University of Western Macedonia
Department of Informatics and Telecommunications Engineering

University of Ioannina
Department of Computer Science

Alexandra Galani
Loukas Georgiadis (Coordinator)

Stavros Nikolopoulos
Leonidas Palios
Reachability

Reachability Query:

Is vertex $b$ reachable from vertex $a$?

(Is there a path in $G$ from $a$ to $b$?)

Goal: Construct a Data Structure that answers reachability queries **efficiently**
Reachability

Reachability Query:

Is vertex $b$ reachable from vertex $a$?

(Is there a path in $G$ from $a$ to $b$?)

Goal: Construct a Data Structure that answers reachability queries **efficiently**

Efficiency of a Data Structure: $\langle s(n), q(n) \rangle$

$s(n)$ storage space
$q(n)$ query time

Easy: Efficiency $\langle n^2, 1 \rangle$ or $\langle m + n, m + n \rangle$

Hard: Efficiency close to $\langle m + n, 1 \rangle$

So far achieved only for restricted graph classes (e.g., planar graphs)
Join-Reachability

Collection of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_\lambda\}$

Join-Reachability Query:

Report all vertices that reach $b$ in all graphs $G_i \in \mathcal{G}$

(Vertices $a$ such that there is a $a \leadsto b$ path in all $G_i \in \mathcal{G}$)

Efficiency of a Data Structure: $\langle s(n), q(n, k) \rangle$

$s(n)$ storage space
$q(n, k)$ time to report $k$ vertices
Join-Reachability

Collection of graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_\lambda\}$

Join-Reachability Query:

Report all vertices that reach $b$ in all graphs $G_i \in \mathcal{G}$

(Vertices $a$ such that there is a $a \leadsto b$ path in all $G_i \in \mathcal{G}$)

Applications: Graph Algorithms, Data Bases

Example: Rank Aggregation

Given a collection of rankings of some items, we would like to report fast all items ranked higher than a query item in all rankings.
Main Idea: Geometric mapping of simple graphs
Join-Reachability

Given two digraphs $G_1$ and $G_2$ with $n$ vertices we can construct join-reachability data structures with the following efficiency:

(a) $\langle n, k \rangle$ when $G_1$ is an unoriented tree and $G_2$ is an unoriented dipath.

(b) $\langle n, \log n + k \rangle$ when $G_1$ is an out-tree and $G_2$ is an unoriented tree.

(c) $\langle n \log^\varepsilon n, \log \log n + k \rangle$ (for any constant $\varepsilon > 0$), when $G_1$ and $G_2$ are unoriented trees.

(d) $\langle n \log n, k \log n \rangle$ when $G_1$ is planar digraph and $G_2$ is an unoriented tree.

(e) $\langle n \log^2 n, k \log^2 n \rangle$ when both $G_1$ and $G_2$ are planar digraphs.

(f) $\langle n\kappa_1, k \rangle$ when $G_1$ is a general digraph that can be covered with $\kappa_1$ vertex-disjoint dipaths and $G_2$ is an unoriented tree.

(g) $\langle n(\kappa_1 + \log n), k\kappa_1 \log n \rangle$ or $\langle n\kappa_1 \log n, k \log n \rangle$ when $G_1$ is a general digraph that can be covered with $\kappa_1$ vertex-disjoint dipaths and $G_2$ is planar digraph.

(h) $\langle n(\kappa_1 + \kappa_2), \kappa_1 \kappa_2 + k \rangle$ or $\langle n\kappa_1 \kappa_2, k \rangle$ when each $G_i$, $i = 1, 2$, is a digraph that can be covered with $\kappa_i$ vertex-disjoint dipaths.
Join-Reachability

Collection of graphs \( \mathcal{G} = \{G_1, G_2, \ldots, G_\lambda\} \)

Join-Reachability Query:

Report all vertices that reach \( b \) in all graphs \( G_i \in \mathcal{G} \)

(Vertices \( a \) such that there is a \( a \leadsto b \) path in all \( G_i \in \mathcal{G} \))

Computing the smallest \( \mathcal{J}(\mathcal{G}) \) (in terms of the number of arcs plus vertices) is \textbf{NP-hard}
Join-Reachability

Collection of graphs \( \mathcal{G} = \{G_1, G_2, \ldots, G_\lambda\} \)

Construction of a compact join-reachability graph \( \mathcal{J}(\mathcal{G}) \)
Join-Reachability

Given two digraphs $G_1$ and $G_2$ with $n$ vertices, the following bounds on the size of the join-reachability graph $\mathcal{J}([G_1, G_2])$ hold:

(a) $\Theta(n \log n)$ in the worst case when $G_1$ is an unoriented tree and $G_2$ is an unoriented dipath.

(b) $O(n \log^2 n)$ when both $G_1$ and $G_2$ are unoriented trees.

(c) $O(n \log^2 n)$ when $G_1$ is a planar digraph and $G_2$ is an unoriented dipath.

(d) $O(n \log^3 n)$ when both $G_1$ and $G_2$ are planar digraphs.

(e) $O(\kappa_1 n \log n)$ when $G_1$ is a digraph that can be covered with $\kappa_1$ vertex-disjoint dipaths and $G_2$ is an unoriented dipath.

(f) $O(\kappa_1 n \log^2 n)$ when $G_1$ is a digraph that can be covered with $\kappa_1$ vertex-disjoint dipaths and $G_2$ is a planar graph.

(g) $O(\kappa_1 \kappa_2 n \log n)$ when each $G_i$, $i = 1, 2$, is a digraph that can be covered with $\kappa_i$ vertex-disjoint dipaths.
Path-Selection

Compute paths in a graph $G$ so that certain requirements are satisfied

E.g.

Avoid a forbidden part of $G$

Disjoint paths

Applications: Communications, Scheduling, VLSI design
Vertex Connectivity

Strongly connected digraph $G = (V, E)$ contains an $s \leadsto t$ path for any pair $s, t \in V$

$k$-vertex connected digraph $G = (V, E)$
the removal of any subset $X \subseteq V, |X| \leq k - 1$
leaves the graph strongly connected

Basic problems:
• Compute vertex connectivity (largest $k$ such that $G$ is $k$-vertex connected)
• Test if the given digraph is $k$-vertex connected
Vertex Connectivity

Basic problems:

• Compute vertex connectivity $\kappa = \text{largest } k \text{ such that } G \text{ is } k\text{-vertex connected}$
  
  $O((n + \min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$ \text{ [Gabow 2006]}

• Test if the given digraph is $k\text{-vertex connected}$

  
  $O(\min\{k^3 + n, kn\}m)$ \text{ [Henzinger, Rao and Gabow 2000]}
  
  $O(mn)$ \text{ with error probability } 1/2

  $O((M(n) + nM(k))\log n)$ \text{ with error probability } $1/n$ \text{ [Cheriyan and Reif 1994]}
  
  $O((M(n) + nM(k))k)$ \text{ expected}

$n = |V|, m = |A|, M(n) = \text{matrix multiplication time} (\approx O(n^{2.376}))$
Vertex Connectivity

Undirected graphs: $O(m + n)$ algorithms for testing

$k = 2$ [Tarjan 1972]

$k = 3$ [Hopcroft and Tarjan 1973]

Directed graphs: $O(m + n)$ algorithm for testing $k = 2$ ?

$n = |V|, m = |A|$
Results

\[ O(m + n) \]-time algorithm for testing 2-vertex connectivity

\[ O(n) \]-space data structure:

compute two vertex-disjoint \( s-t \) paths in \( O(\log^2 n) \) time

report the two paths, \( P \) and \( Q \), in \( O(|P| + |Q|) \) time

\( n = |V|, m = |A| \)
Vertex Connectivity

A $k$-vertex connected digraph $G = (V, E)$ is vertex connected if the removal of any subset $X \subseteq V$, $|X| \leq k - 1$ leaves the graph strongly connected.

From Menger’s theorem:

$G$ is $k$-vertex connected if and only if $G$ contains $k$ vertex-disjoint $s$-$t$ paths for any $s, t \in V$. 
2-Vertex Connectivity

2-vertex connected digraph \( G = (V, E) \)
the removal of at most one vertex
leaves the graph strongly connected

If \( G \) is strongly connected but not 2-vertex connected:

There are \( s, t \in V \) such that all \( s-t \) paths contain a common vertex \( x \neq s, t \)
2-Vertex Connectivity

2-vertex connected digraph \( G = (V, E) \)

the removal of at most one vertex
leaves the graph strongly connected

If \( G \) is strongly connected but not 2-vertex connected:

There are \( s, t \in V \) such that all \( s-t \) paths contain a common vertex \( x \neq s, t \)
2-Vertex Connectivity

2-vertex connected digraph $G = (V, E)$

the removal of at most one vertex
leaves the graph strongly connected

If $G$ is strongly connected but not 2-vertex connected:

There are $s, t \in V$ such that all $s-t$ paths contain a common vertex $x \neq s, t$
Flowgraphs and Dominators

Flowgraph $G(s) = (V, E, s)$: all vertices are reachable from start vertex $s$

$v$ dominates $w$ if every path from $s$ to $w$ includes $v$

dom($w$): set of vertices that dominate $w$

Trivial dominators: $s, w \in \text{dom}(w)$

Application areas: Program optimization, VLSI testing, theoretical biology, distributed systems, constraint programming
Flowgraphs and Dominators

Flowgraph $G(s) = (V, E, s)$: all vertices are reachable from start vertex $s$

$v$ dominates $w$ if every path from $s$ to $w$ includes $v$

$G(s)$

O($m\alpha(m, n)$) algorithm: [Lengauer and Tarjan ’79]
O($m + n$) algorithms:
[Alstrup, Harel, Lauridsen, and Thorup ‘97]
[Buchsbaum, Kaplan, Rogers, and Westbrook ‘04]
[G., and Tarjan ‘04]
2-Vertex Connectivity

Main Idea: Compute dominators in $G(s)$ and $G^r(s)$ for arbitrary $s \in V$

$G^r$: has reversed arcs
2-Vertex Connectivity

Main Idea: Compute dominators in $G(s)$ and $G^r(s)$ for arbitrary $s \in V$

$G^r$: has reversed arcs
2-Vertex Connectivity

Main Idea: Compute dominators in $G(s)$ and $G^r(s)$ for arbitrary $s \in V$

$G^r$: has reversed arcs
2-Vertex Connectivity

Main Idea: Compute dominators in $G(s)$ and $G^r(s)$ for arbitrary $s \in V$

$G^r$: has reversed arcs
2-Vertex Connectivity

Main Idea: Compute dominators in $G(s)$ and $G^r(s)$ for arbitrary $s \in V$

$G^r$: has reversed arcs
Vertex-Disjoint s-t Paths

Given a digraph $G = (V, E)$ how fast can we compute a pair of vertex-disjoint $s$-$t$ paths?
Vertex-Disjoint s-t Paths

Given a digraph $G = (V, E)$ how fast can we compute a pair of vertex-disjoint $s$-$t$ paths?

$O(m + n)$ time: “vertex-splitting” + “flow augmentation”
Vertex-Disjoint s-t Paths

Given a digraph $G = (V, E)$ how fast can we compute a pair of vertex-disjoint $s$-$t$ paths?

$O(m + n)$ time : “vertex-splitting” + “flow augmentation”

We can get a more efficient solution when $G$ is 2-vertex connected

• Use a 2-vertex connected spanning subgraph of $G$ with $O(n)$ arcs

[Cheriyan and Thurimella 2000] : $1 + 1/k$ approximation of the minimum $k$-vertex connected spanning subgraph in $O(km^2)$ time
Vertex-Disjoint s-t Paths

Given a digraph $G = (V, E)$ how fast can we compute a pair of vertex-disjoint $s-t$ paths?

$O(m + n)$ time: “vertex-splitting” + “flow augmentation”

We can get a more efficient solution when $G$ is 2-vertex connected

• Use a 2-vertex connected spanning subgraph of $G$ with $O(n)$ arcs
  
  [Cheriyan and Thurimella 2000]: $1 + 1/k$ approximation of the minimum $k$-vertex connected spanning subgraph in $O(km^2)$ time

• Use pairs of independent trees
Vertex-Disjoint s-t Paths

Any flowgraph $G(s) = (V, A, s)$ has two spanning trees, $B$ and $R$, such that for any $v \in V$

$$B[s, v] \cap R[s, v] = dom(v)$$

the two trees can be computed in linear time
**Vertex-Disjoint s-t Paths**

Corollary: If $G(s)$ has trivial dominators only then for any $v \in V$

$$B(s, v) \cap R(s, v) = \emptyset$$

the two trees can be computed in linear time
Vertex-Disjoint s-t Paths

Corollary: A digraph $G = (V, A)$ is 2-vertex connected if and only if for two arbitrary vertices $a, b \in V$ ($a \neq b$) the flowgraphs $G(a), G^r(a), G(b)$ and $G^r(b)$ have trivial dominators only.

We use a pair of independent spanning trees for each of the flowgraphs

$$G(a), G^r(a), G(b), G^r(b)$$
2-Vertex Connectivity

$P_1, P_2$ : vertex-disjoint $a$-$t$ paths

$P_3, P_4$ : vertex-disjoint $s$-$a$ paths

Suppose

$P_3[s, a) \cap (P_1(a, t] \cup P_2(a, t)) \neq \emptyset$

$P_4(s, a) \cap (P_1(a, t) \cup P_2(a, t)) = \emptyset$
2-Vertex Connectivity

\( P_1, P_2 : \) vertex-disjoint \( a-t \) paths

\( P_3, P_4 : \) vertex-disjoint \( s-a \) paths

Suppose

\[ P_3[s, a) \cap (P_1(a,t] \cup P_2(a,t)] \neq \emptyset \]

\[ P_4(s, a) \cap (P_1(a,t) \cup P_2(a,t)) = \emptyset \]

Let \( x \) be the first vertex on \( P_3[s, a] \) such that \( x \in (P_1(a,t] \cup P_2(a,t]) \).
2-Vertex Connectivity

$P_1, P_2 :$ vertex-disjoint $a$-$t$ paths

$P_3, P_4 :$ vertex-disjoint $s$-$a$ paths

Suppose

$$P_3[s, a] \cap (P_1(a, t) \cup P_2(a, t)) \neq \emptyset$$

$$P_4(s, a) \cap (P_1(a, t) \cup P_2(a, t)) = \emptyset$$

Let $x$ be the first vertex on $P_3[s, a]$ such that $x \in (P_1(a, t) \cup P_2(a, t))$.

Consider $x \in P_1(a, t) \implies$

$$P_3[s, x] \cdot P_1[x, t] \text{ and } P_4[s, a] \cdot P_2[a, t] \text{ are vertex-disjoint } s$-$t \text{ paths}$$
2-Vertex Connectivity

Data Structure: Given rooted trees $S_1$ and $S_2$ on the same nodes support the operations:

(i) Test if $S_1[x_1, y_1]$ contains $x_2$.

(ii) Return the topmost vertex in $S_1(x_1, y_1)$.

(iii) Test if $S_1[x_1, y_1]$ and $S_2[x_2, y_2]$ contain a common vertex.

(iv) Find the lowest ancestor of $y_2$ in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$.

(v) Find the highest ancestor of $y_2$ in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$. 
2-Vertex Connectivity

Data Structure : Given rooted trees $S_1$ and $S_2$ on the same nodes support the operations:

(i) Test if $S_1[x_1, y_1]$ contains $x_2$.

(ii) Return the topmost vertex in $S_1(x_1, y_1)$.

(iii) Test if $S_1[x_1, y_1]$ and $S_2[x_2, y_2]$ contain a common vertex.

(iv) Find the lowest ancestor of $y_2$ in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$.

(v) Find the highest ancestor of $y_2$ in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$.

• A query uses a constant number of these operations.

• We give an $O(n)$-space data structure with $O(\log^2 n)$ time per operation.
Example: Pairs of Disjoint Paths in the New York Area
Example: Pairs of Disjoint Paths in the New York Area
Example: Pairs of Disjoint Paths in the New York Area
Example: Pairs of Disjoint Paths in the New York Area
Computational Morphological Analysis

**Morphological Analysis**: the study of the internal structure of words

**Fundamental Aim**: identification of the constituents of words and the properties they express.

```
  e.g.  play    kind    read
       play-ed  kind-ness  read-ing
       play-ing  read-er
       play-er  read-er-s
       play-er-s  read-able
```

**Issues**:  
- What morphological units languages consist of?  
- What features are represented in each morpheme?  
- How do morphemes and features interact with one another?  
- Are there any constraints in the selection of morphemes in specific environments?
Computational Morphological Analysis

Computational Approach: Morphological patterns as graph reachability and path selection problems