Vertex Connectivity

Danupon Nanongkai

Please ask questions anytime!
1. The Problem
Problem: Compute $k = \text{minimum number of nodes whose removals disconnect the input graph}$

Examples

Formally, we define $k(G)$ for input graph $G$. We omit $G$ since it's clear from the context.
How fast can computers compute k?
How **fast** can computers compute $k$?

$k=1$

<table>
<thead>
<tr>
<th>Time</th>
<th>Best possible $(O(m))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Depth-first search [Tarjan’71]</td>
</tr>
</tbody>
</table>

Found in

$m = \text{number of edges, } n = \text{number of nodes}$
How fast can computers compute k?

<table>
<thead>
<tr>
<th>k=1</th>
<th>k=2</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Best possible (O(m))</th>
<th>Best possible (O(m))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Depth-first search [Tarjan'71]</td>
<td>SPQR tree [Hopcroft-Tarjan'73]</td>
</tr>
</tbody>
</table>

- m = number of edges, n = number of nodes

Supported in: NetworkX

Found in: Introduction to Algorithms
How fast can computers compute $k$?

<table>
<thead>
<tr>
<th>$k=1$</th>
<th>$k=2$</th>
<th>$k=3$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Best possible ($O(m)$)</th>
<th>Best possible ($O(m)$)</th>
<th>Quadratic ($O(n^2)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Depth-first search [Tarjan’71]</td>
<td>SPQR tree [Hopcroft-Tarjan’73]</td>
<td>Kleitman’69</td>
</tr>
</tbody>
</table>

**Found in**
- NetworkX
- igraph
- Maplesoft

**Supported in**
  - $O(m)$ time for k-connectivity, for any $k$

$m = \text{number of edges}, n = \text{number of nodes}$
Despite many ideas, **quadratic** time for $k \geq 3$ remains

<table>
<thead>
<tr>
<th>Reference</th>
<th>$k=3$</th>
<th>General $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kleitman’69, Podderyugin’73, Even Tarjan’75</td>
<td>$n^2$</td>
<td>$nk \cdot \maxflow_{\geq k}$</td>
</tr>
<tr>
<td>Even’75, Galil’80, Esfahanian Hakimi’84, Matula’87</td>
<td>$n^2$</td>
<td>$(k^2 + n) \cdot \maxflow_{\geq k}$</td>
</tr>
<tr>
<td>Becker et al’82</td>
<td>$n^2$</td>
<td>$n \cdot \maxflow_{\geq k}$</td>
</tr>
<tr>
<td>Linial Lovasz Wigderson’88, Cheriyan Reif’91</td>
<td>$n^{2.37}$</td>
<td>$n^\omega + nk^\omega$</td>
</tr>
<tr>
<td>Nagamochi Ibaraki’92</td>
<td>$n^2$</td>
<td>Can assume $m = O(nk)$</td>
</tr>
<tr>
<td>Henzinger Rao Gabow’00</td>
<td>$n^2$</td>
<td>$mn$ and $mn + mk^3 + mnk$</td>
</tr>
<tr>
<td>Gabow’06</td>
<td>$n^2$</td>
<td>$mn + mk^{2.5} + mn^{3/4}k$</td>
</tr>
</tbody>
</table>

$m = \text{number of edges}, n = \text{number of nodes}, \text{polylog}(n) \text{ hidden}$
Our Result

Joint works with

• Thatchaphol Saranurak, Sorrachai Yingchareonthawornchai, STOC’19
• Sebastian Forster, Thatchaphol Saranurak, Liu Yang, Sorrachai Yingchareonthawornchai, SODA’20
Our algorithm (today)

“Best possible” ($\tilde{O}(m)$) time for $k=3, 4, \ldots, \Theta(1)$

$\tilde{O}$ hides polylog(n)

More generally,

$\tilde{O}(mk^2)$ time for any $k$

Follow-up result (not today)

$m^{1+o(1)}$ time for any $k$

More general: max-flow time

Li, N, Panigrahi, Saranurak, Yingchareonthawornchai, STOC'21

$m =$ number of edges, $n =$ number of nodes, $\tilde{O}$ hides polylog(n)
Experimental confirmation

Results on graphs with 50,000 — 250,000 vertices.

HRG is the previously best result, LOCAL is our algorithm, LOCAL+ & LOCALh+ are algorithm with heuristics.

Credit: Max Franck (Aalto University, Finland)
2. Local Algorithm Paradigm
Key idea: Explore locally

Don’t read the whole input!
Key idea: Explore **locally**

Example for vertex connectivity:
Try to start at **s near** the cut nodes → find solution by exploring locally

**Fast** to find cut node

**Take longer** to find cut node

(More on this later)
Roles of the Local Paradigm

• Natural paradigm for distributed computing, property testing, etc.

• Relatively new paradigm for sequential algorithms (compared to, e.g., randomized and approximation algorithms)

• Recently led to many exciting results, e.g.
  • Fast linear system solver (Spielman-Teng’04)
  • Edge connectivity (Kawarabayashi-Thorup’18)
  • Dynamic minimum spanning tree (N., Saranurak, Wulff-Nilsen’17)
  • Vertex connectivity (today)
  • Fast max-flow (CKLPPS’22)
3. Algorithm (Sketch)
The Framework (for $k=O(1)$)

For $r=1, 2, 4, 8, \ldots, \frac{m}{4}$ do these:

1. **Sample** nodes $S_1, \ldots, S_r$
   - Probability proportional to degrees
   - Goal: “hit” smaller side of the cut

Example

Claim: The smaller side is hit if it is incident to $\approx \frac{m}{r}$ edges

$m = \text{number of edges}, n = \text{number of nodes}, \tilde{O} \text{ hides polylog}(n)$
The Framework \((for \, k=O(1))\)

For \(r=1, 2, 4, 8, \ldots, m/4\) do these:

1. **Sample** nodes \(S_1, \ldots, S_r\)
   Probability proportional to **degrees**

2. **Explore locally**: For each \(s_i\), call \(\text{LocalVC}(s_i, m/r)\) (next)

**Subroutine (sketched):** \(\text{LocalVC}(s, m/r)\)
- Spend \(O(m/r)\) time reading \(O(m/r)\) edges near \(s\)
- ... to find cut nodes

Time: \(\tilde{O}\left(r \times \left(\frac{m}{r}\right)\right) = \tilde{O}(m)\)

\(m = \text{number of edges, } n = \text{number of nodes, } \tilde{O} \text{ hides polylog}(n)\)
LocalVC for $k=1$

$\approx \frac{m}{r}$ edges
LocalVC for $k=1$

1. Convert $G$ to digraph $G'$
   For every edge $(u,v)$

Analogy:
LocalVC for $k=1$

$G \approx \frac{m}{r}$ edges

Only way from $s$ to the bigger side
LocalVC for $k=1$

2. Depth-first search (DFS) over $\frac{4m}{r}$ edges

Observe At least $\frac{m}{r}$ of these edges are on the bigger side
LocalVC for $k=1$

3. Sample stopping node $t$
4. Flip directions of all edges from $s$ to $t$

$\approx \frac{m}{r}$ edges

$\geq \frac{m}{r}$ edges
LocalVC for $k=1$

5. Depth-first search (DFS) once more
   → Got stuck
   → Found cut edge/vertex

$\approx \frac{m}{r}$ edges

$\geq \frac{m}{r}$ edges
LocalVC for k=3

1

2

3

Stuck
The Pseudocode (for k=O(1))

Main algorithm
For r=1, 2, 4, 8, ..., m/4 do these:
1. **Sample** nodes $s_1, ..., s_r$
   Probability proportional to degrees
2. **Explore locally**: For each $s_i$, call LocalVC($s_i, m/r$)
   Also repeat this algorithm log(n) times

Subroutine (sketched): LocalVC($s, \mu$)
Repeat $k + 1$ times on corresponding digraph $G'$
1. Grow **depth-first search** tree $T$ from $s$ for $4\mu$ edges
   • If get stuck, found the cut.
   Output & terminate
2. **Sample** an edges $(t', t)$ in $T$
3. Reverse the path $P_{st}$ in $T$
   Terminate with no cut.

Corner case: DFS gets stuck but visits all nodes, then sample all edges in the graph

Time: $O(k\mu) = O\left(\frac{m}{r}\right)$
For $k=O(1)$

$m = \text{number of edges}, n = \text{number of nodes}, \text{polylog}(n) \text{ omitted}$
Conclusion
Potential project examples

- **Theory projects**: Improve the state of the art in other settings or in special cases, e.g.
  - Distributed: improve Jiang-Mukhopadhyay STOC’23, tight lower bound, big k
  - Parallel: beat reachability computation
  - Cut query, communication, quantum
  - Directed edge connectivity, weighted vertex cut

- **Implementation projects**: Implement existing algorithms and improve with heuristics
  - Vertex connectivity
  - Negative-weight shortest paths
  - Mincut
Thank you. Question?