Vertex Connectivity

Danupon Nanongkai

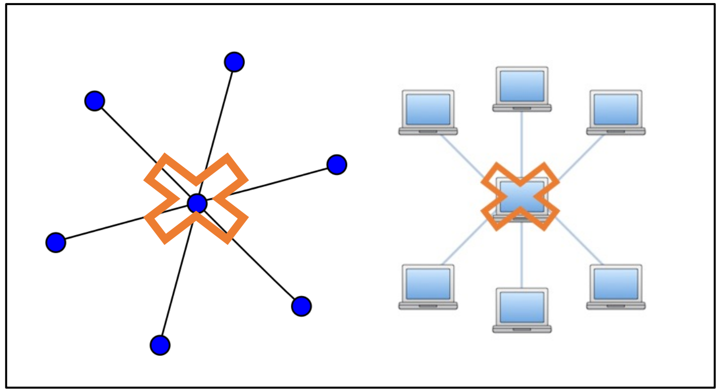
Please ask questions anytime!

1. The Problem

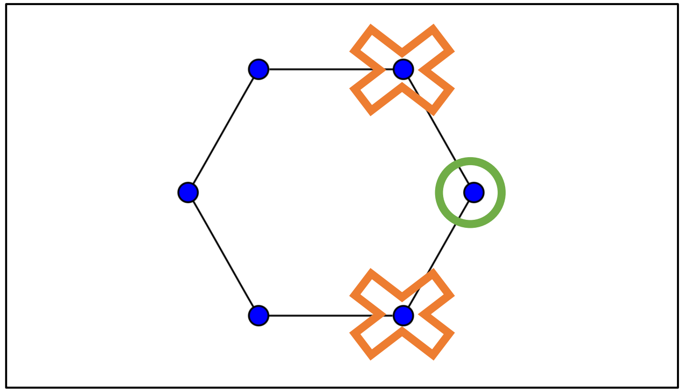
<u>Problem</u>: Compute

$\boldsymbol{k}$ = minimum number of nodes whose removals **disconnect** the input graph

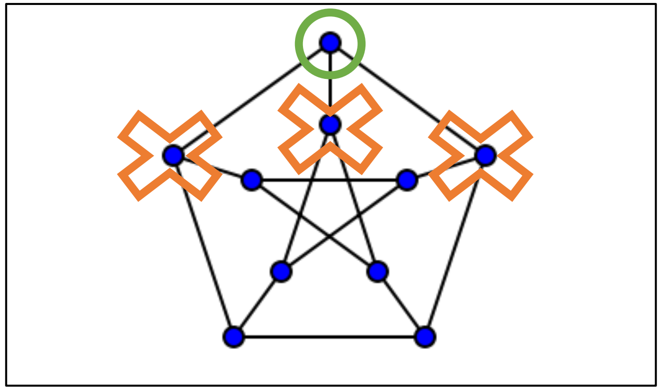
<u>Examples</u>



k=1



k=2



k=3

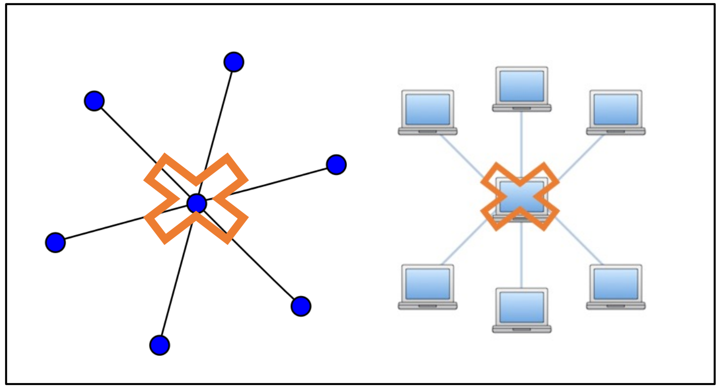
Formally, we define $k(G)$ for input graph G. We omit G since it's clear from the context.

How **fast** can computers compute k?

4

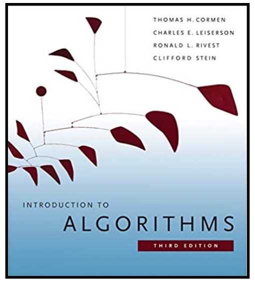
How **fast** can computers compute k?

k=1



Time	Best possible (O(m))
Algorithm	Depth-first search [Tarjan'71]

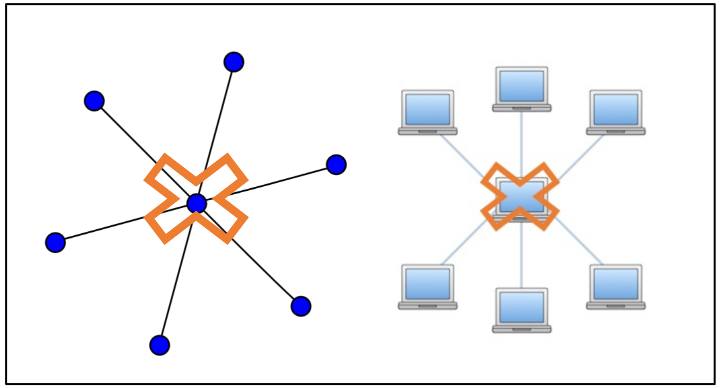
Found in



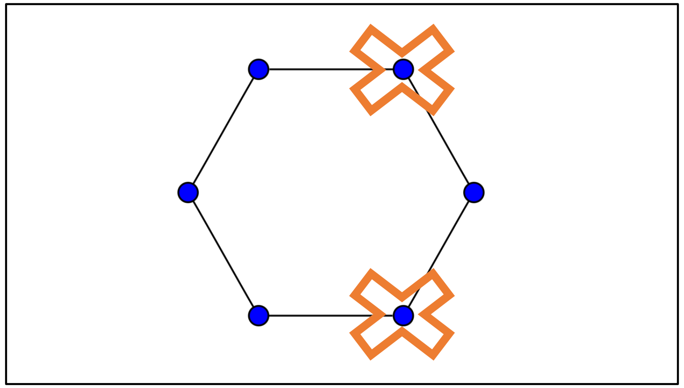
m = number of edges, n = number of nodes

How **fast** can computers compute k?

k=1

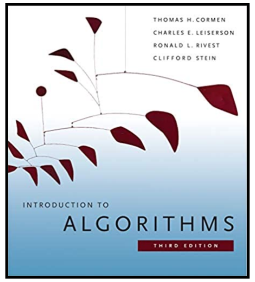


k=2



Time	Best possible (O(m))	Best possible (O(m))
Algorithm	Depth-first search [Tarjan'71]	SPQR tree [Hopcroft-Tarjan'73]

Found in



Supported in

NetworkX
Network Analysis in Python

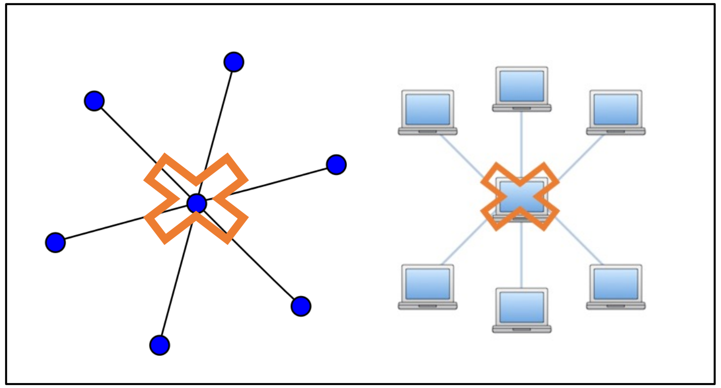
igraph

Maplesoft

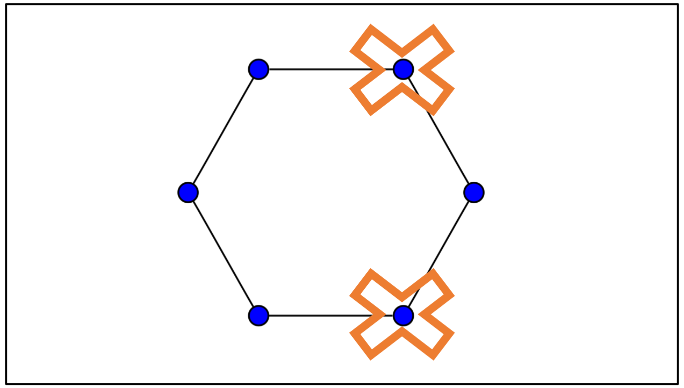
m = number of edges, n = number of nodes

How **fast** can computers compute k?

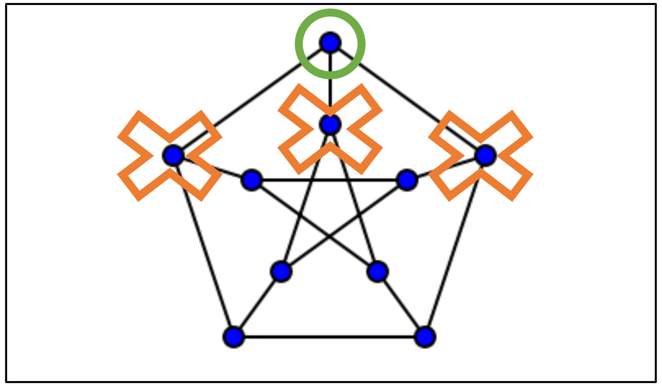
k=1



k=2

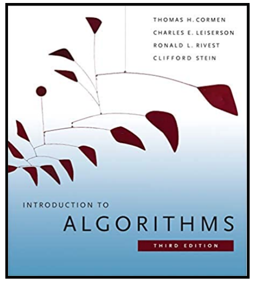


k=3



Time	Best possible (O(m))	Best possible (O(m))	Quadratic ($O(n^2)$)
Algorithm	Depth-first search [Tarjan'71]	SPQR tree [Hopcroft-Tarjan'73]	Kleitman'69

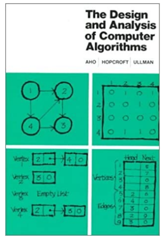
Found in



Supported in NetworkX Network Analysis in Python, igraph, Maplesoft

Aho-Hopcroft-Ullman Conjecture (1974)

O(m) time for k-connectivity, for any k



m = number of edges, n = number of nodes

Despite many ideas, quadratic time for k≥3 remains

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

m = number of edges, n = number of nodes, polylog(n) 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, …, O(1)

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

More generally,

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

Follow-up result (not today)

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

More general: max-flow time

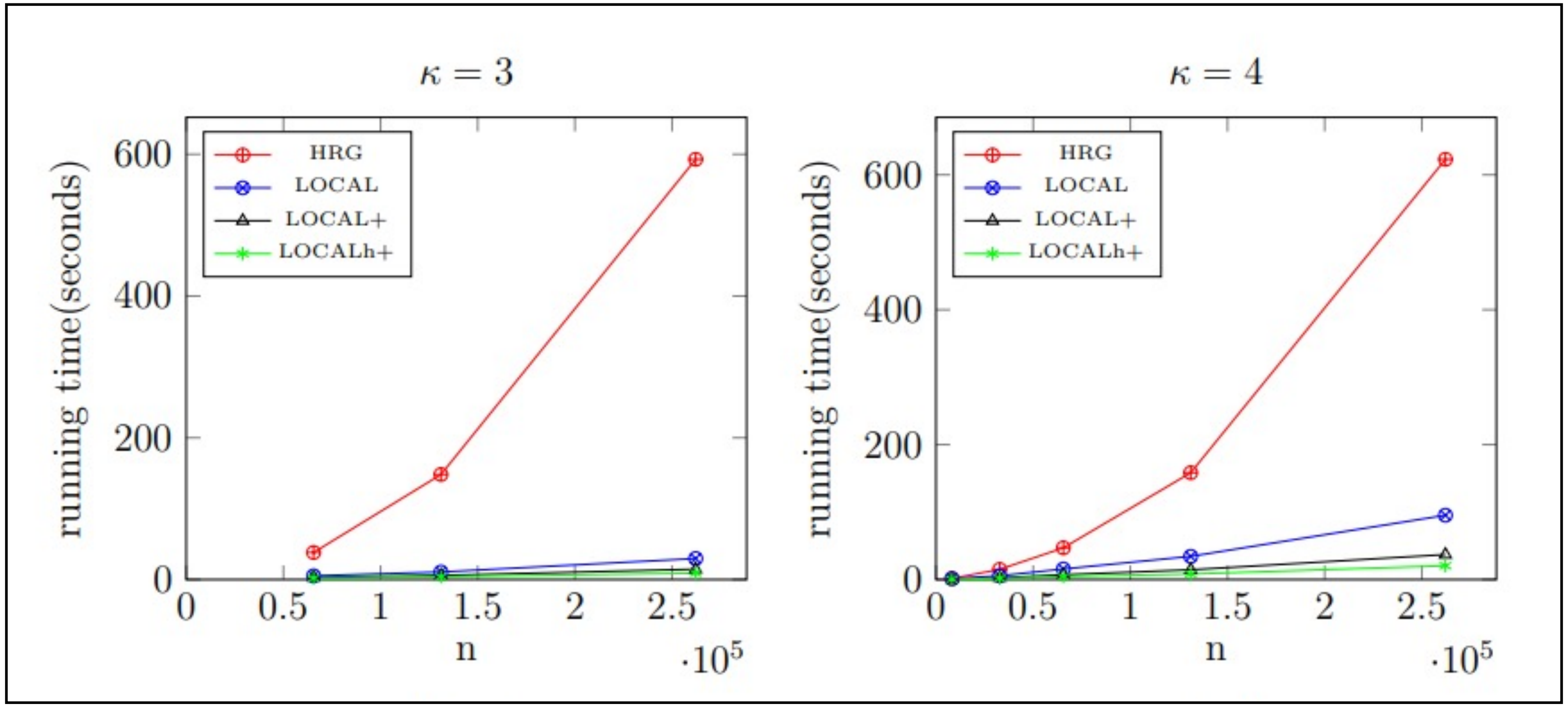
Li, **N,** Panigrahi, Saranurak, Yingchareonthawornchai, STOC'21

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**

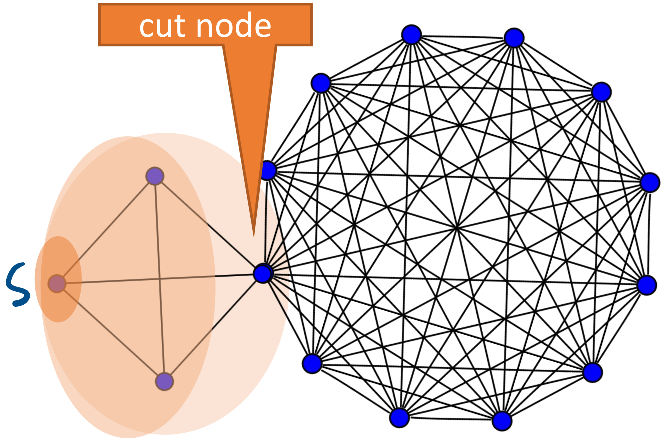
Don’t read the whole input!

Key idea: Explore **locally**

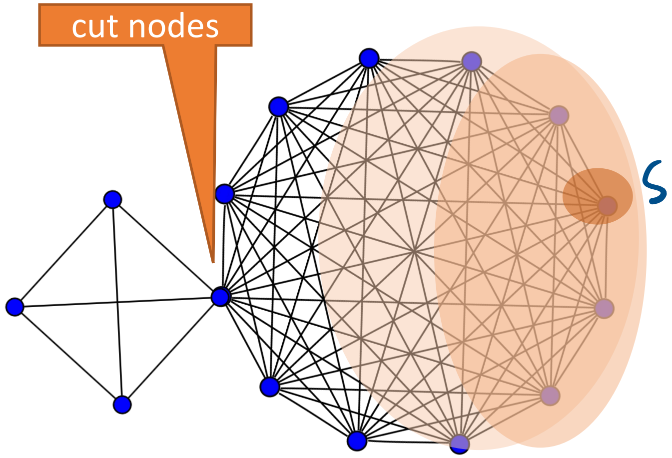
Don't read the whole input!

<u>Example for vertex connectivity:</u>

Try to start at **s** <u>near</u> the cut nodes → find solution by exploring locally



Fast to find cut node

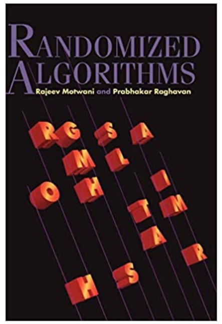


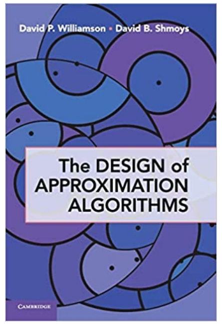
Take **longer** to find cut node

(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)

Skip to conclusion

The Framework (for k=O(1))

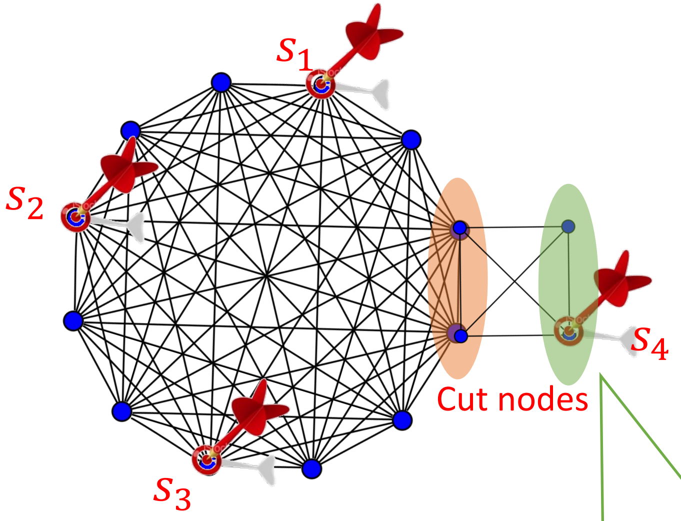
For **r=1, 2, 4, 8, …, m/4** do these:

1. **Sample** nodes $s_1, \dots, s_r$

   Probability proportional to **degrees**

   Goal: “hit” smaller side of the cut

Example



Claim The smaller side is hit if it is incident to ≈ **m/r edges**

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

The Framework (for k=O(1))

For **r=1, 2, 4, 8, …, m/4** do these:

1. **Sample** nodes $s_1, \dots, s_r$

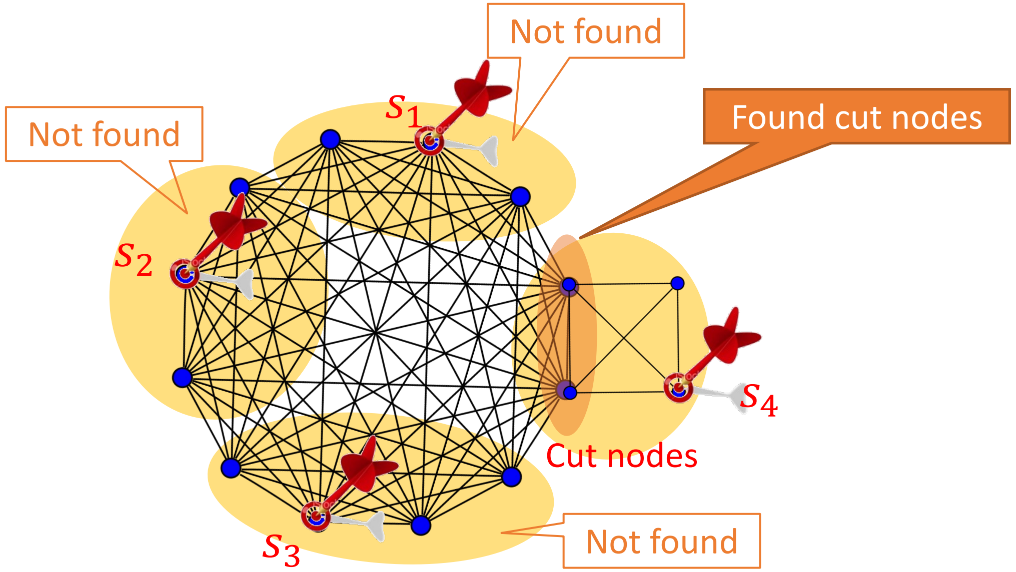
   Probability proportional to **degrees**

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

Subroutine (sketched): LocalVC(s, m/r)

- Spend **O(m/r)** time reading **O(m/r)** edges **near s**
- … to find cut nodes

Example

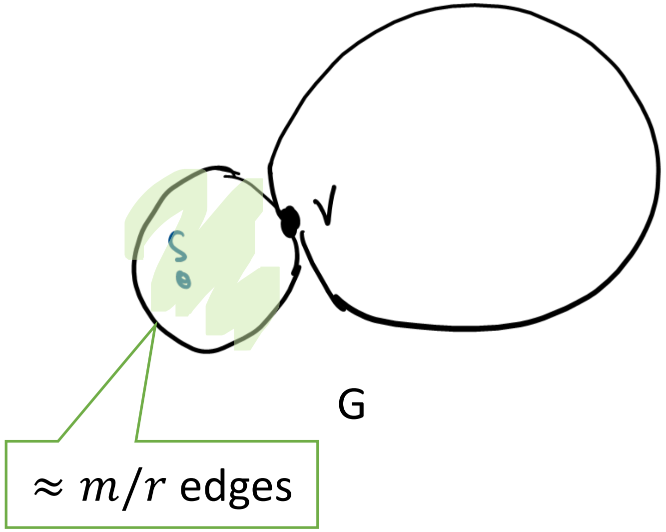


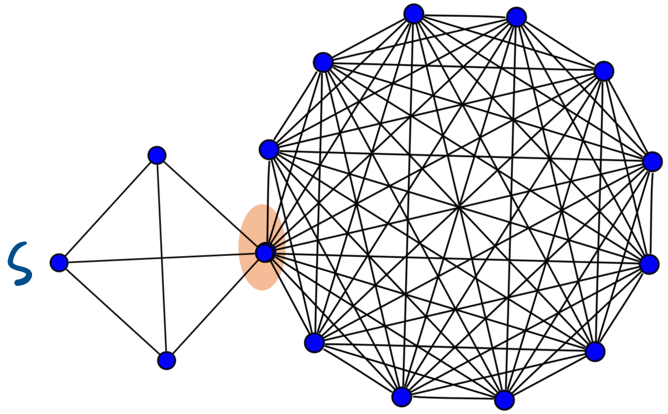
Time: $\tilde{O}\left(r \times \left(\frac{m}{r}\right)\right) = \tilde{O}(m)$

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

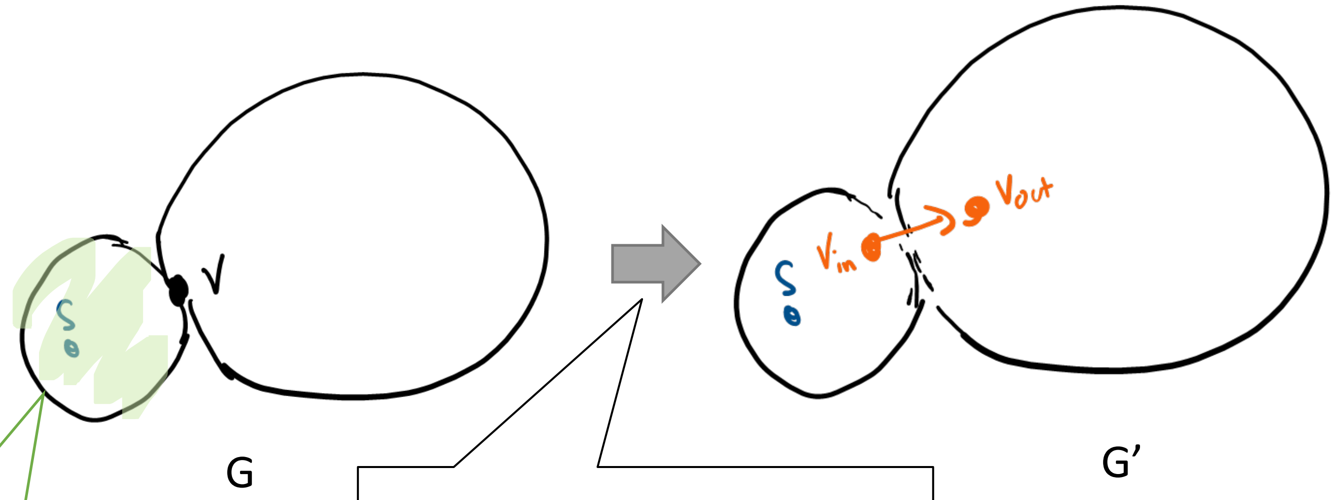
Skip to conclusion

LocalVC for k=1



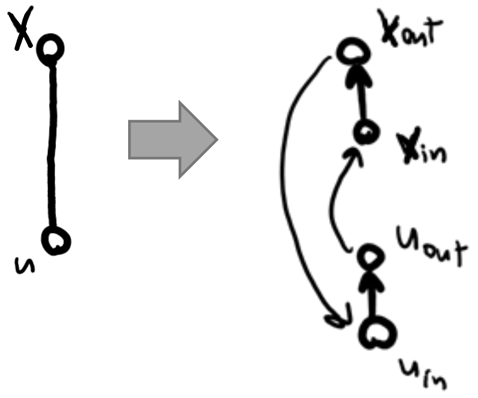


LocalVC for k=1

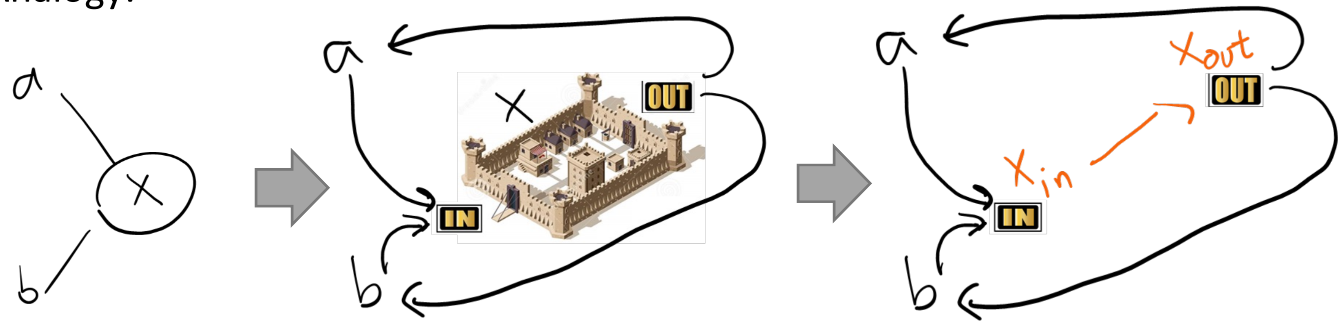


$\approx m/r$ edges

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

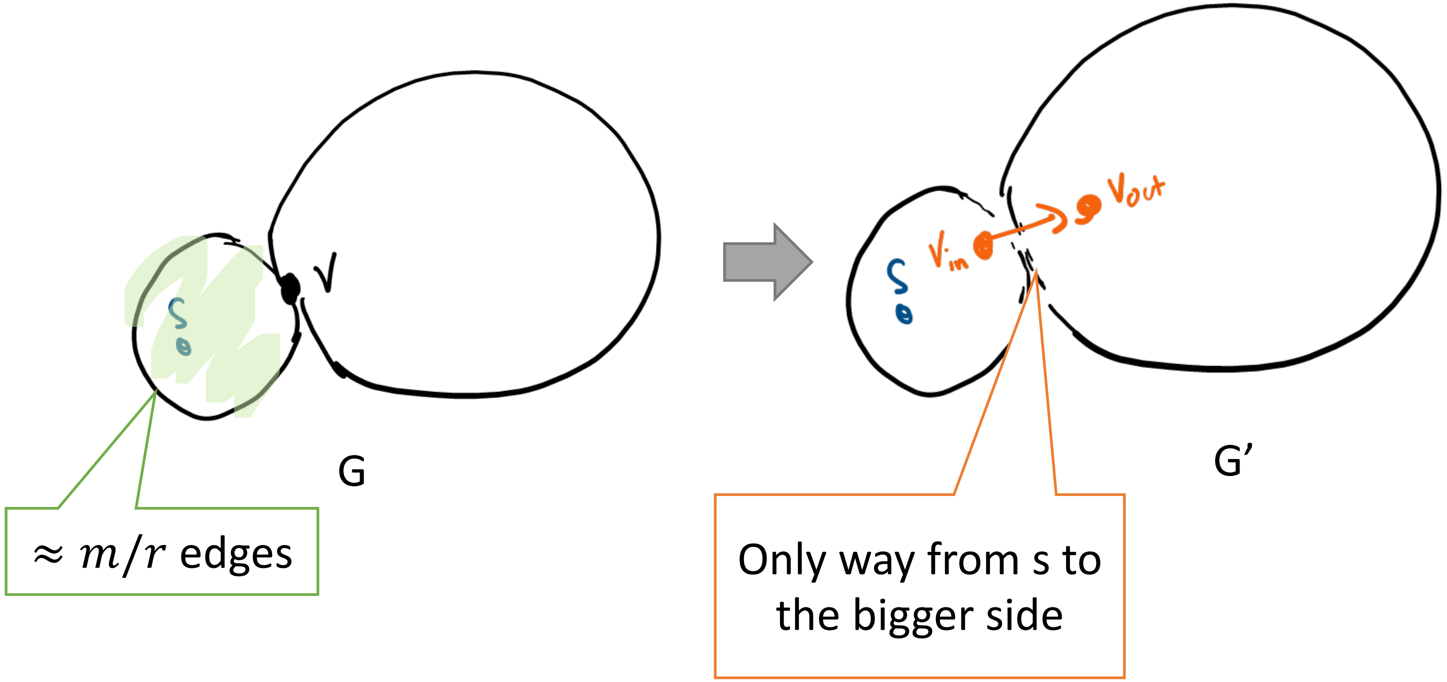


Analogy:

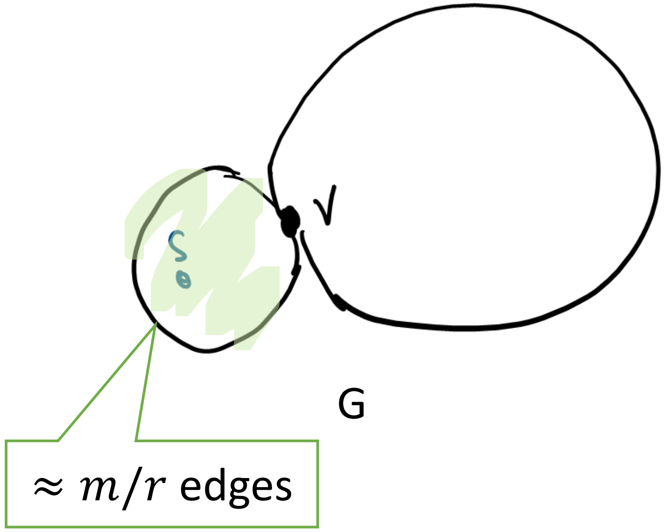


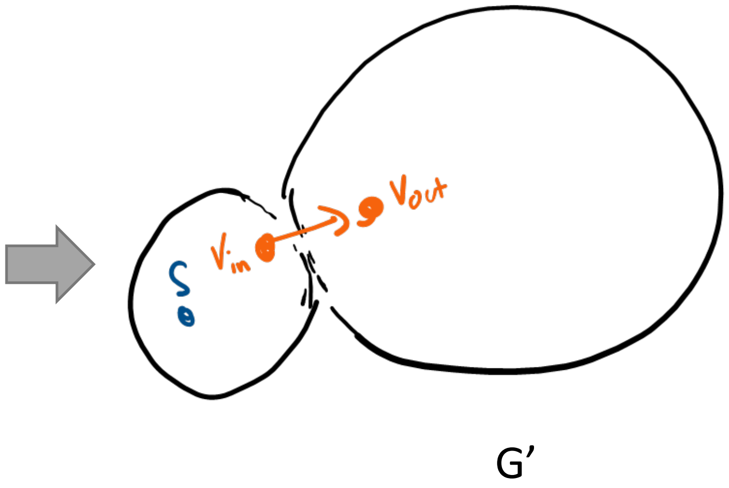
Skip to conclusion

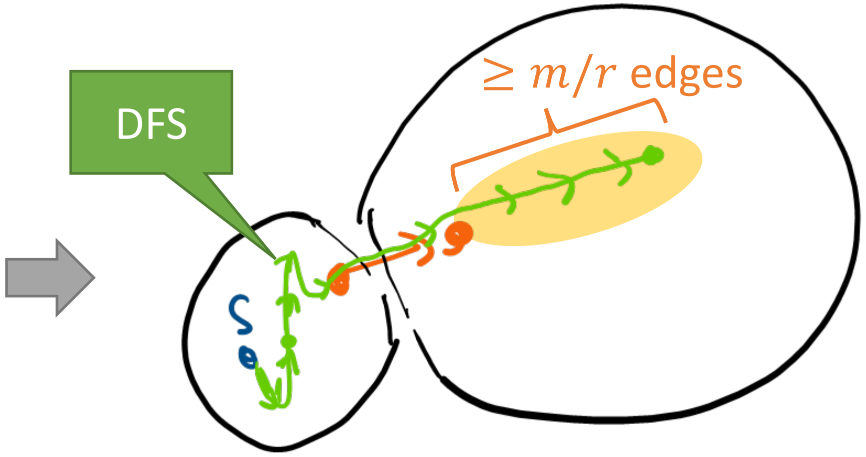
LocalVC for k=1



LocalVC for k=1



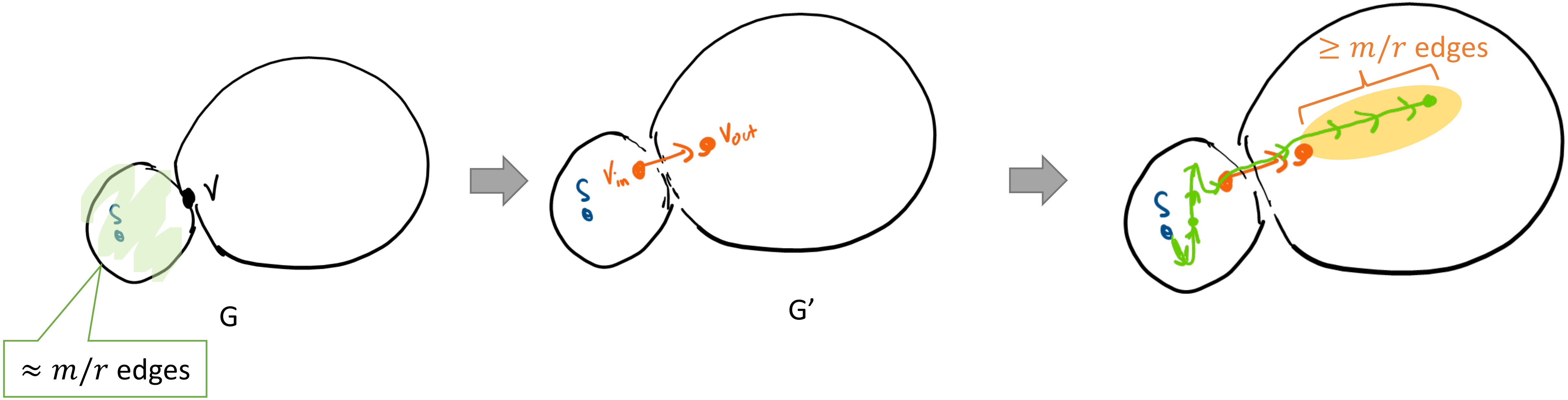


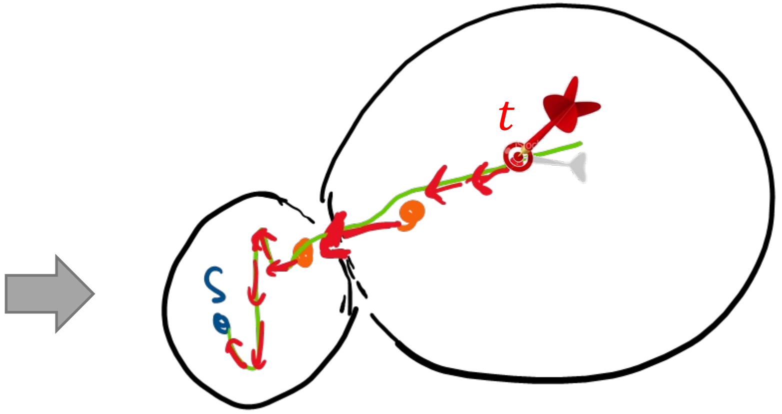


2. **Depth-first search (DFS)** over $\frac{\mathbf{4m}}{\mathbf{r}}$ edges

<u>Observe</u> At least m/r of these edges are on the bigger side

LocalVC for k=1

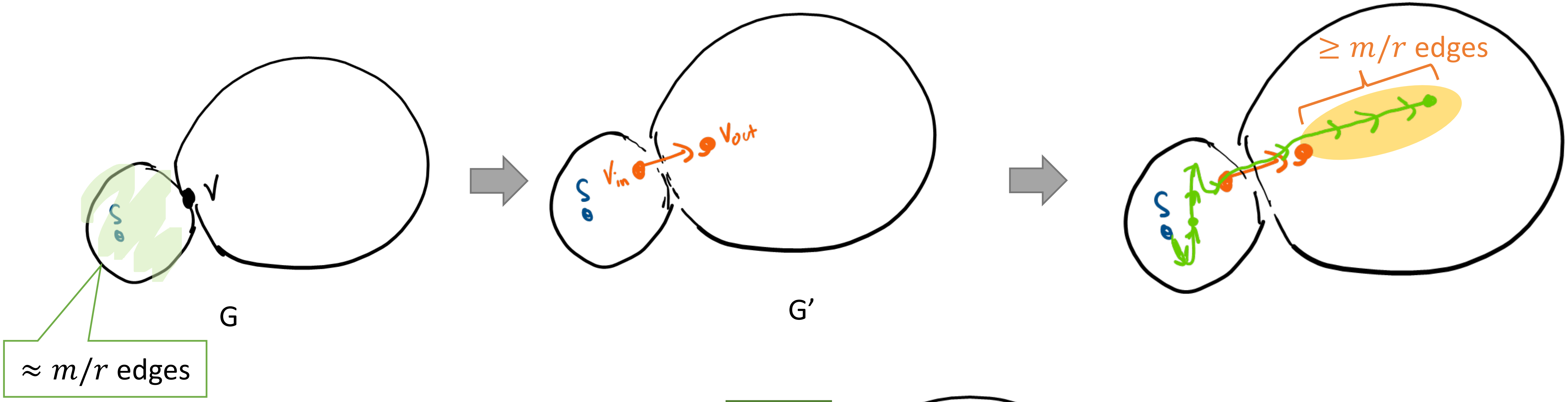


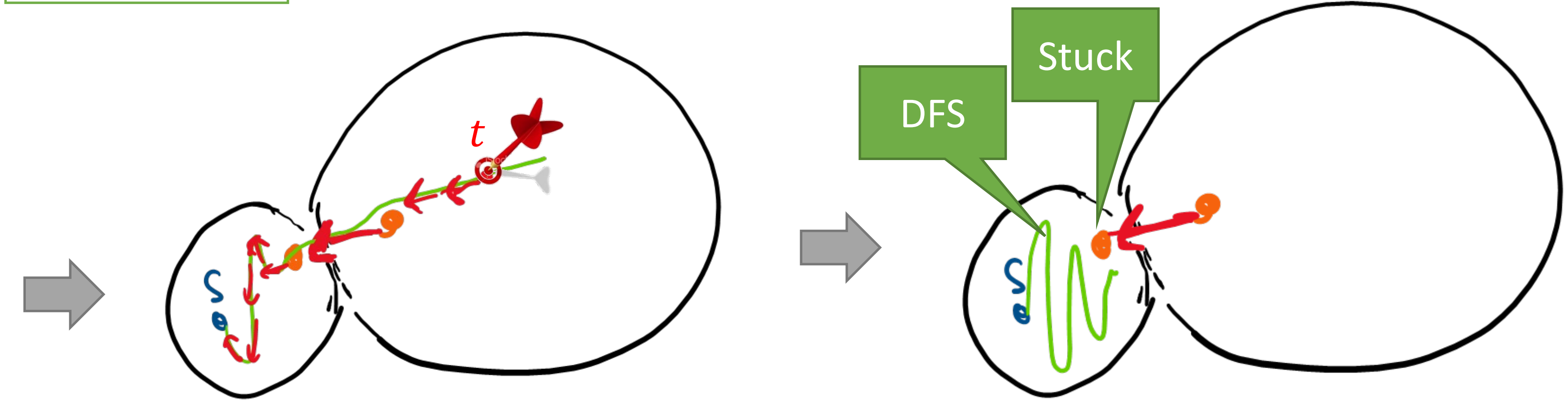


3. Sample stopping node t
4. Flip directions of all edges from s to t

Skip to conclusion

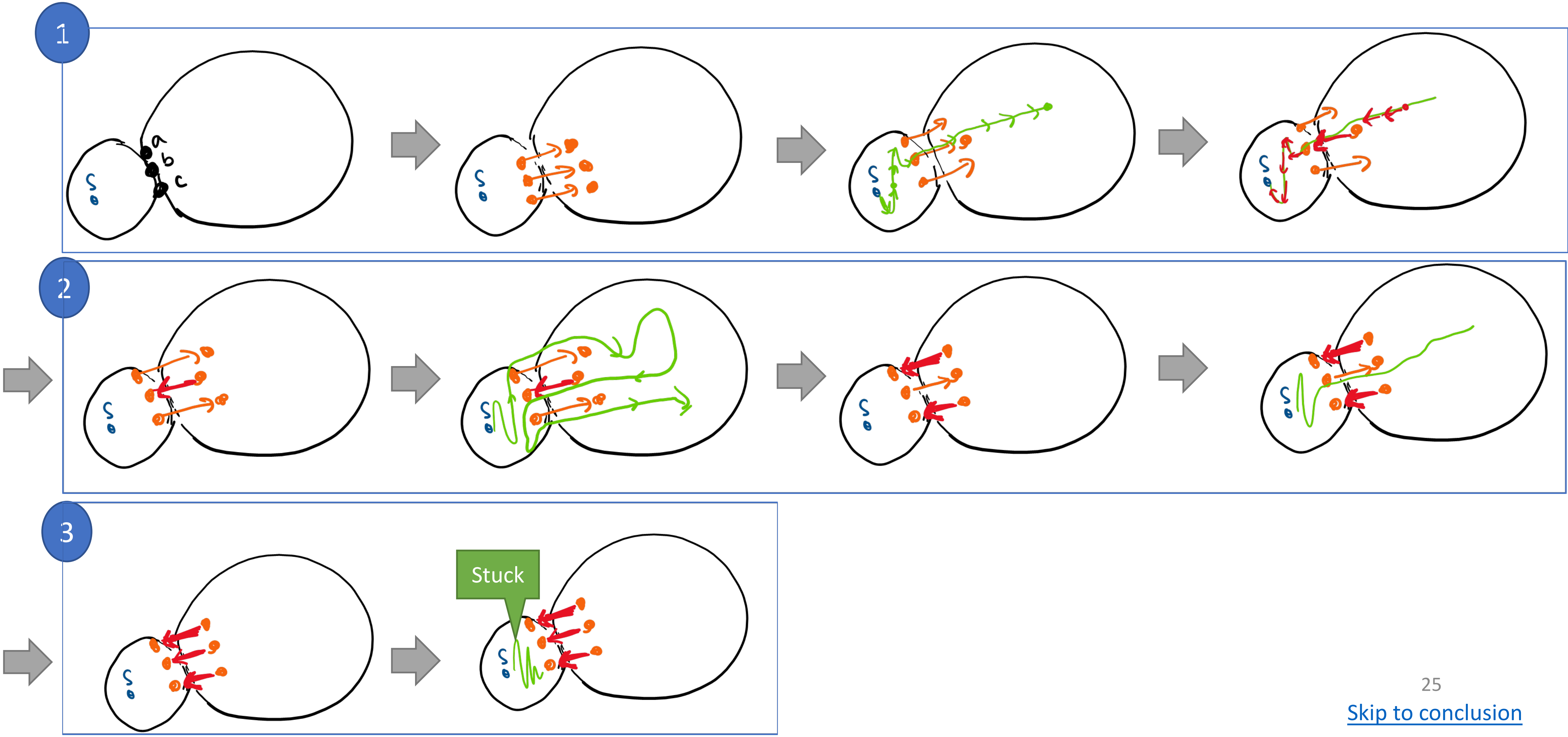
LocalVC for k=1





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

LocalVC for k=3



The Pseudocode (for k=O(1))

m/r

Main algorithm

For **r=1, 2, 4, 8, …, m/4** do these:

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

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

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

Subroutine (sketched): LocalVC(s, μ)

Repeat $k+1$ times on corresponding digraph **G'**

1. Grow **depth-first search tree** T from s for 4μ 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

m = number of edges, n = number of nodes, polylog(n) omitted

Skip to conclusion

Conclusion

Potential project examples

- <u>Theory projects</u>: 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
- <u>Implementation projects:</u> Implement existing algorithms and improve with heuristics
  - Vertex connectivity
  - Negative-weight shortest paths
  - Mincut

Thank you. Question?