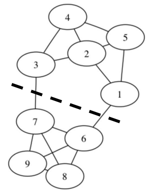
Edge connectivity (Global minimum cut)

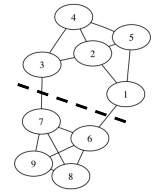
turn the graph into directed graph, set all edge capacities to 1
pick any node v
for all $u \in V\backslash\{v\}$
    run max-flow algorithm with source v and sink u
output the minimum flow obtained



Complexity: $O(n \cdot n^3) = O(n^4)$

Edge connectivity (Global minimum cut)

turn the graph into directed graph, set all edge capacities to 1
pick any node v
for all $u \in V\backslash\{v\}$
    run max-flow algorithm with source v and sink u
output the minimum flow obtained



Complexity: $O(n \cdot n^3) = O(n^4)$

Improvements:

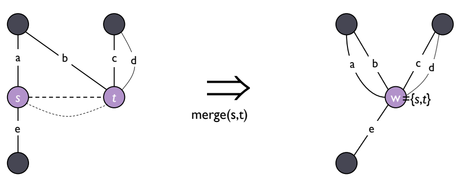
- $O(m \cdot polylog(n))$ [Karger 1991] *probabilistic algorithm*
- $O\left(m + K^2 n \log\frac{n}{K}\right)$ where K is edge connectivity [Gabow 1995]
- $O(nm + n^2 \log n)$ [Stoer, Wagner 1997] (*simple*! weighted case)
- $O(m \cdot polylog(n))$ [Kawarabayashi, Thorup 2018]

Considerations

- Enumerating all $u \in V\backslash\{v\}$ in the previous algorithm seems inefficient and *may* be improved
- Computing edge connectivity *may* be simpler than computing maximal flow, as we don't have fixed s and t. (We only need to find *some* s and t in opposite sides of the cut)

[Stoer, Wagner 97]: first idea

- Consider some nodes s and t and assume we know $mincut(s,t)$ (minimum cut which separates s and t)
- Case 1: $mincut(s,t)$ is the global minimum cut
- Case 2: otherwise, s and t are on the same side of the global min cut ⇒ global min cut is not changed if s and t are ***merged*** (parallel edges allowed)



[Stoer, Wagner 97]: first idea

- Consider some nodes s and t and assume we know $mincut(s,t)$ (minimum cut which separates s and t)
- Case 1: $mincut(s,t)$ is the global minimum cut
- Case 2: otherwise, s and t are on the same side of the global min cut ⇒ global min cut is not changed if s and t are ***merged*** (parallel edges allowed)

function $GlobalMinCut(G)$
  if $V = \{u,v\}$ **then**
    return nb of edges between u and v
  else
    $(C_1, s, t) = stMinCut(G)$
    $C_2 = GlobalMinCut(G/\{s,t\})$
    return $\min\{C_1, C_2\}$

returns $s,t \in V$ with $C_1 = mincut(s,t)$

G with merged s,t

[Stoer, Wagner 97]: second idea

- $stMinCut(G)$ returns *some* nodes $s,t \in V$ with $C_1 = mincut(s,t)$
- can be done more efficiently than computing max flow!

[Stoer, Wagner 97]: second idea

- $stMinCut(G)$ returns *some* nodes $s,t \in V$ with $C_1 = mincut(s,t)$
- can be done more efficiently than computing max flow!

function $stMinCut(G)$
  $A = \{v\}$ arbitrary node
  while $A \neq V$
    pick $u \in V \setminus A$ s.t. nb of edges bewteen A and u is maximized
    $A = A \cup \{u\}$
  let s,t be the last two nodes added to A and C the number of edges between t and $V \setminus \{t\}$,
  return (C, s, t)

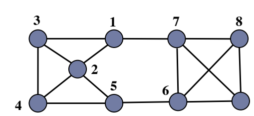
[Stoer, Wagner 97]: second idea

- $stMinCut(G)$ returns *some* nodes $s,t \in V$ with $C_1 = mincut(s,t)$
- can be done more efficiently than computing max flow!

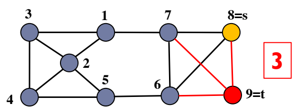
function $stMinCut(G)$
  $A = \{v\}$ arbitrary node
  while $A \neq V$
    pick $u \in V \setminus A$ s.t. nb of edges bewteen A and u is maximized
    $A = A \cup \{u\}$
  let s,t be the last two nodes added to A and C the number of edges between t and $V \setminus \{t\}$,
  return (C, s, t)

Theorem: $stMinCut$ is correct
Proof: cf [Stoer, Wagner 97] or
http://www.cs.tau.ac.il/~zwick/grad-algo-08/gmc.pdf

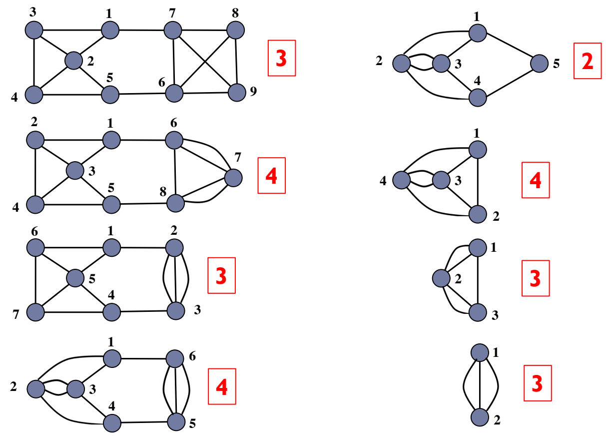
Example



Example



Example



[Stoer, Wagner 97]: resulting complexity

- how $stMinCut(G)$ is implemented?

[Stoer, Wagner 97]: resulting complexity

- how $stMinCut(G)$ is implemented?
- max-priority queue!
  - maintain all nodes outside A in a max-priority queue
  - when adding a node u to A, increment keys of all nodes $x \in V \setminus A$ by the number of edges $\{u, x\}$
- implementation with binary heaps:
  - construction: $O(n \log n)$ (all keys set to 0)
  - $n - 1$ extract-max: $O(n \log n)$
  - m updates (increments): $O(m \log n)$
  - altogether, $stMinCut(G)$ takes time $O((m + n) \log n)$
- resulting complexity of $GlobalMinCut(G)$: $O(n(n + m) \log n)$
- with Fibonacci heaps: $O(nm + n^2 \log n)$