Improved Algorithms for Min Cuts and Max Flows in Undirected Planar Graphs

Giuseppe F. Italiano
Università di Roma "Tor Vergata"

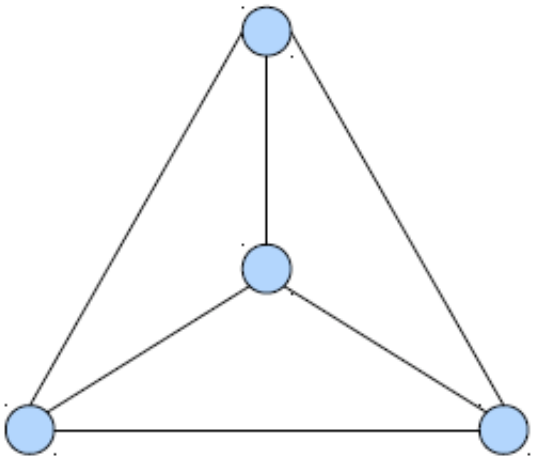
Joint work with Yahav Nussbaum, Piotr Sankowski and Christian Wulff-Nilsen

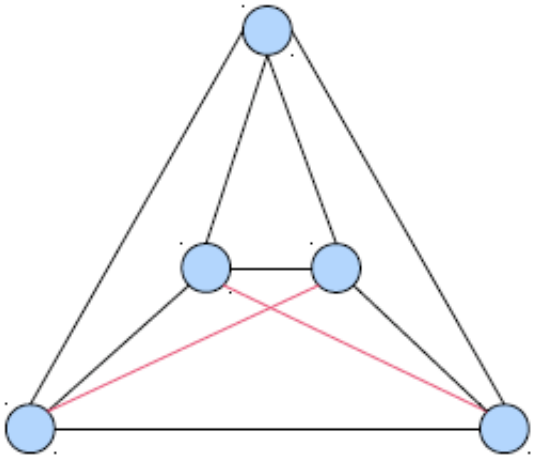
Outline

- Introduction
- History
- Dense Distance Graphs
- Minimum s-t cut in O(n log log n) time
- Applications, including max flow and dynamic min s-t cut

Planar Graphs

- A graph that has an embedding in the plane without crossing edges
- We assume that a combinatorial embedding is given
- Such embedding can be found in O(n) time – Hopcroft & Tarjan [1974]

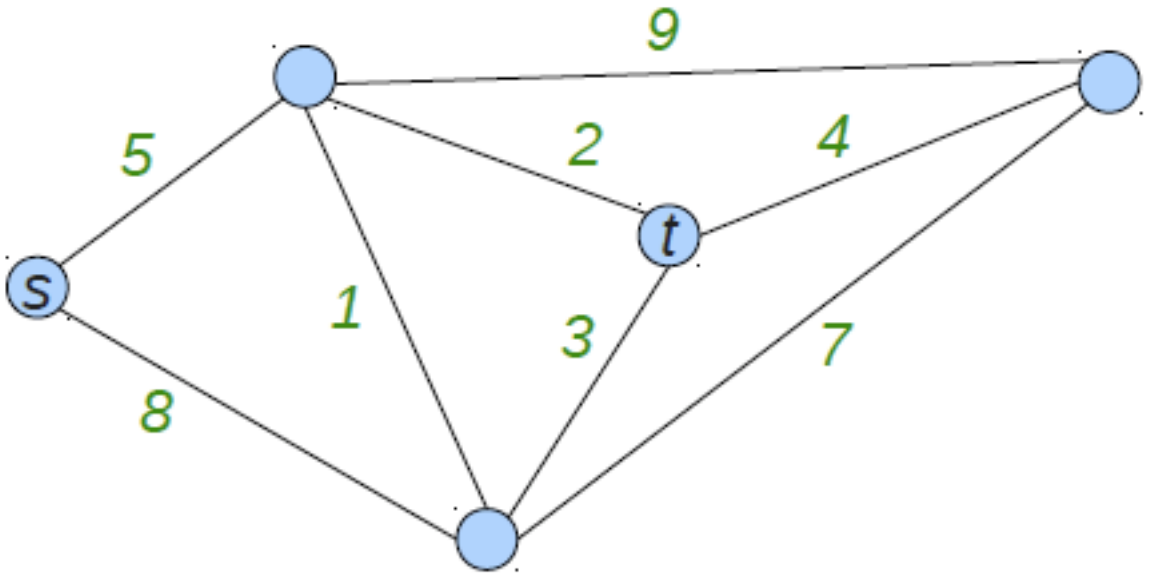




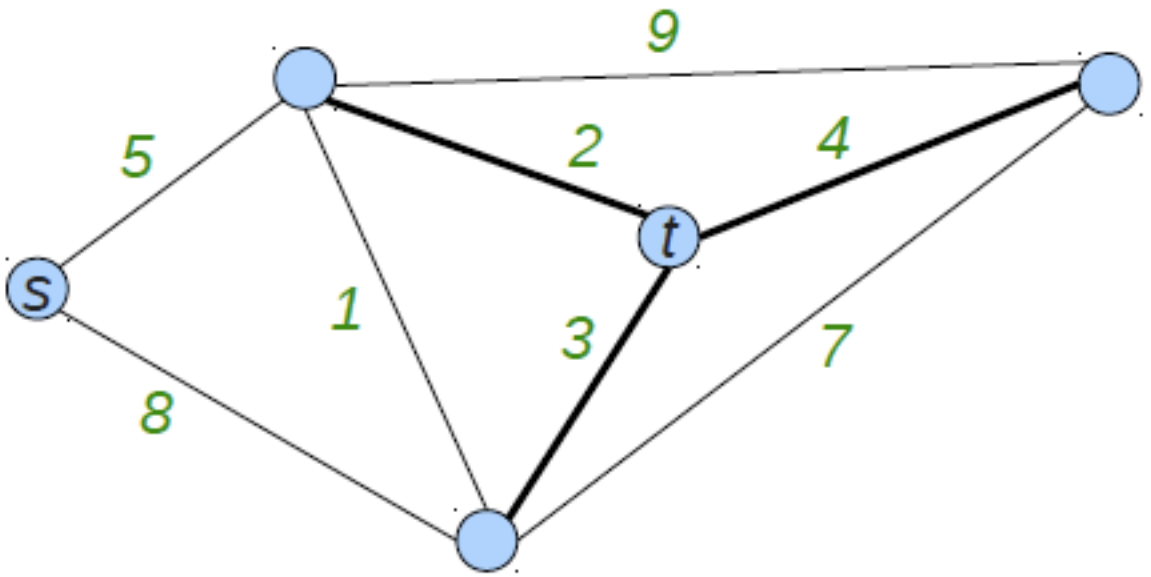
Why Planar Graphs?

- Special properties of planar graphs often allows for simpler and faster algorithms
- Arise in many applications – Transportation, VLSI, etc…

Min s-t Cut



Min s-t Cut



Among the Oldest Problems in CS

aka Warsaw Pact Rail Network

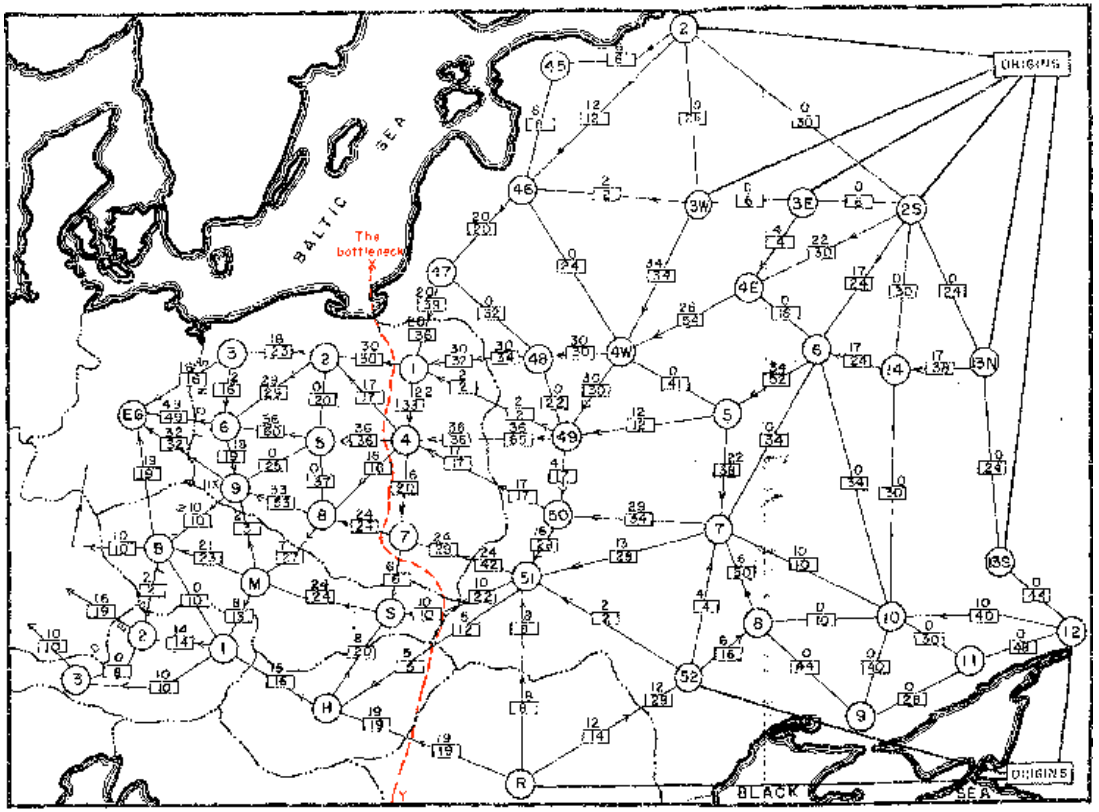
Harris & Ross [1955] studied rail network from Soviet Union to Eastern Europe

Edge weights = shipping rates

Computed by hand:

1. Max amount of stuff that could be moved from Soviet Union to Eastern Europe

2. "The bottleneck", i.e., cheapest way to disrupt network by removing links (blowing up train tracks)

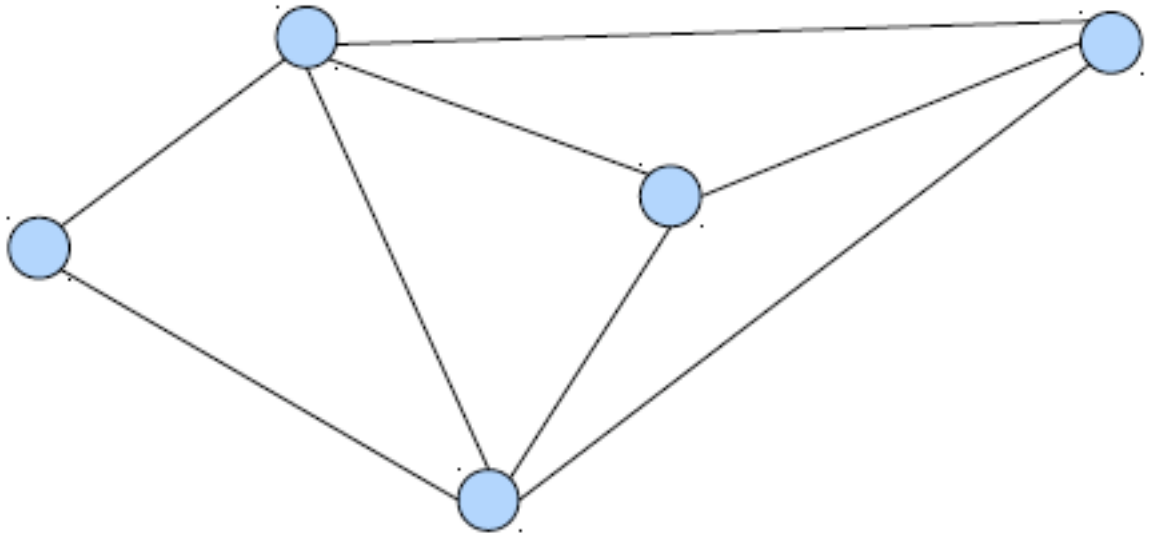


Outline

- ✔ Introduction
- History
- Dense Distance Graphs
- Minimum s-t cut in O(n log log n) time
- Applications

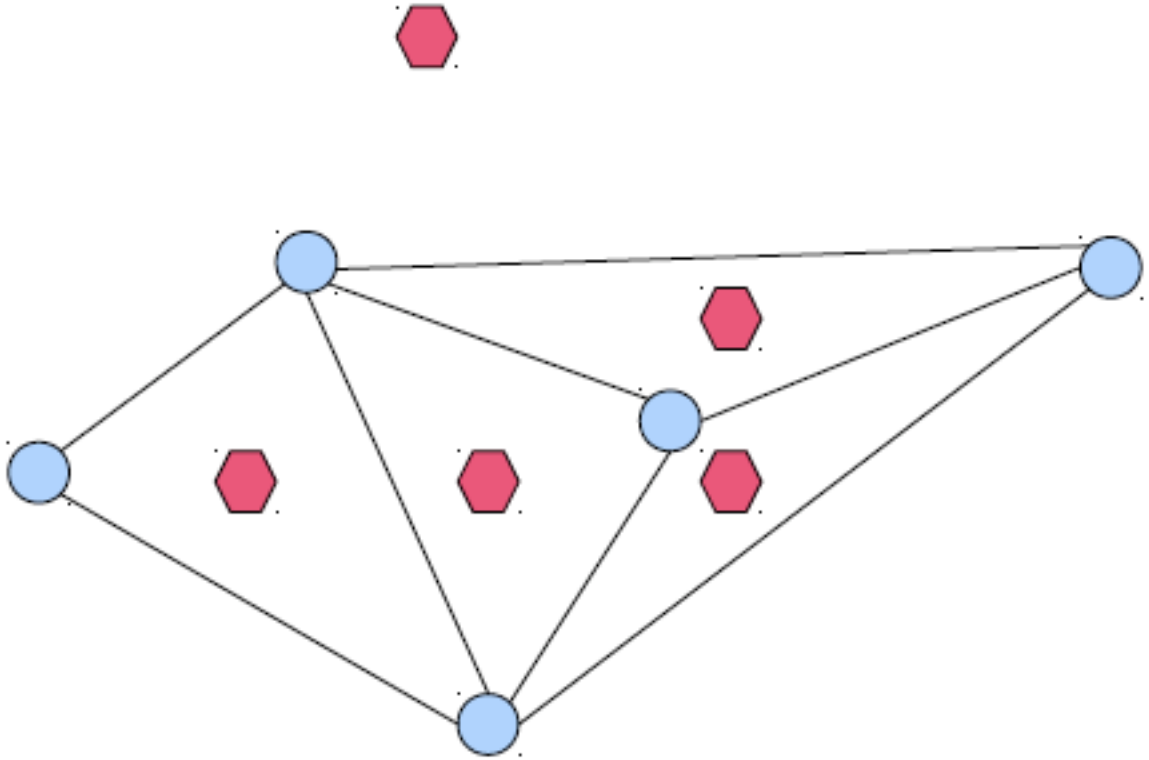
Dual Graph G^*

- Each face becomes a vertex



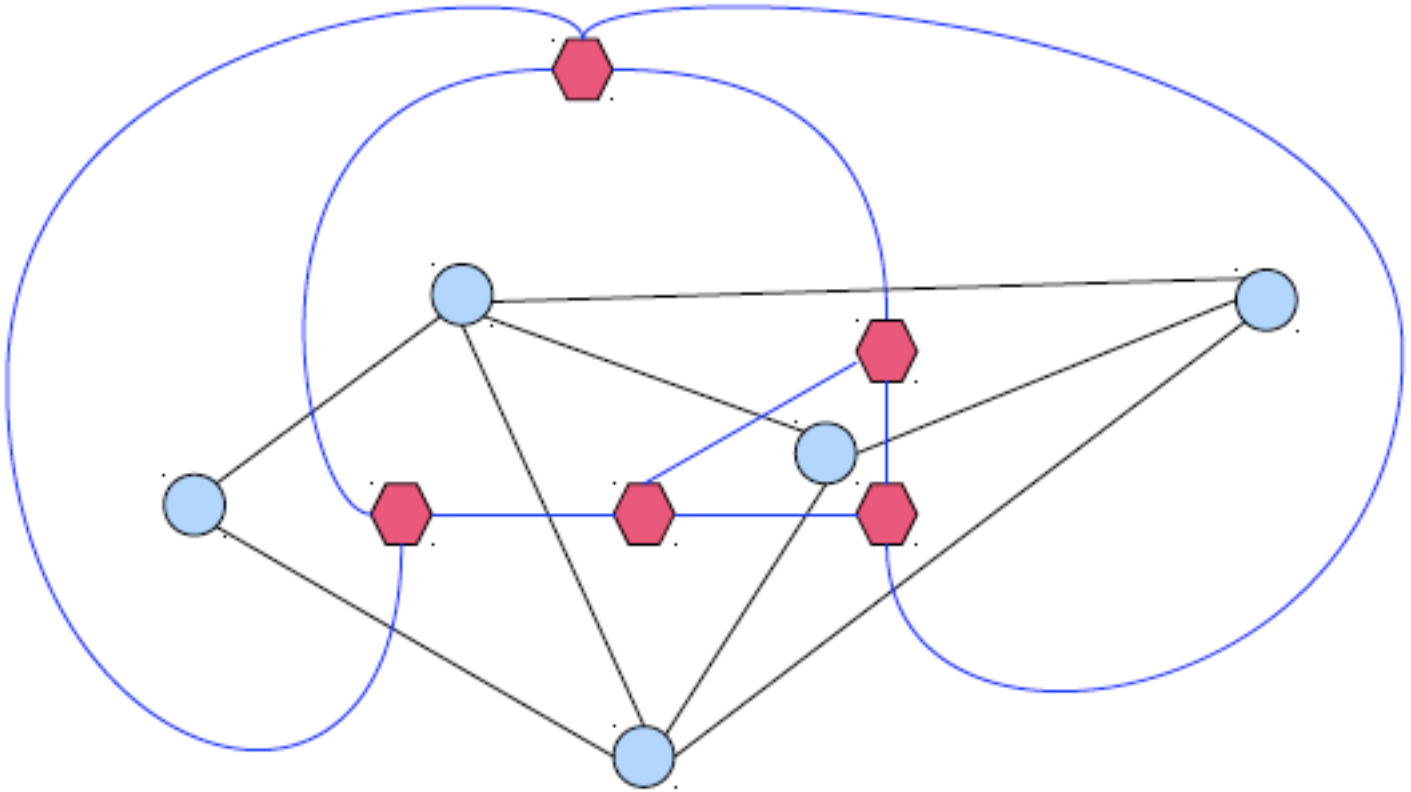
Dual Graph G^*

- Each face becomes a vertex



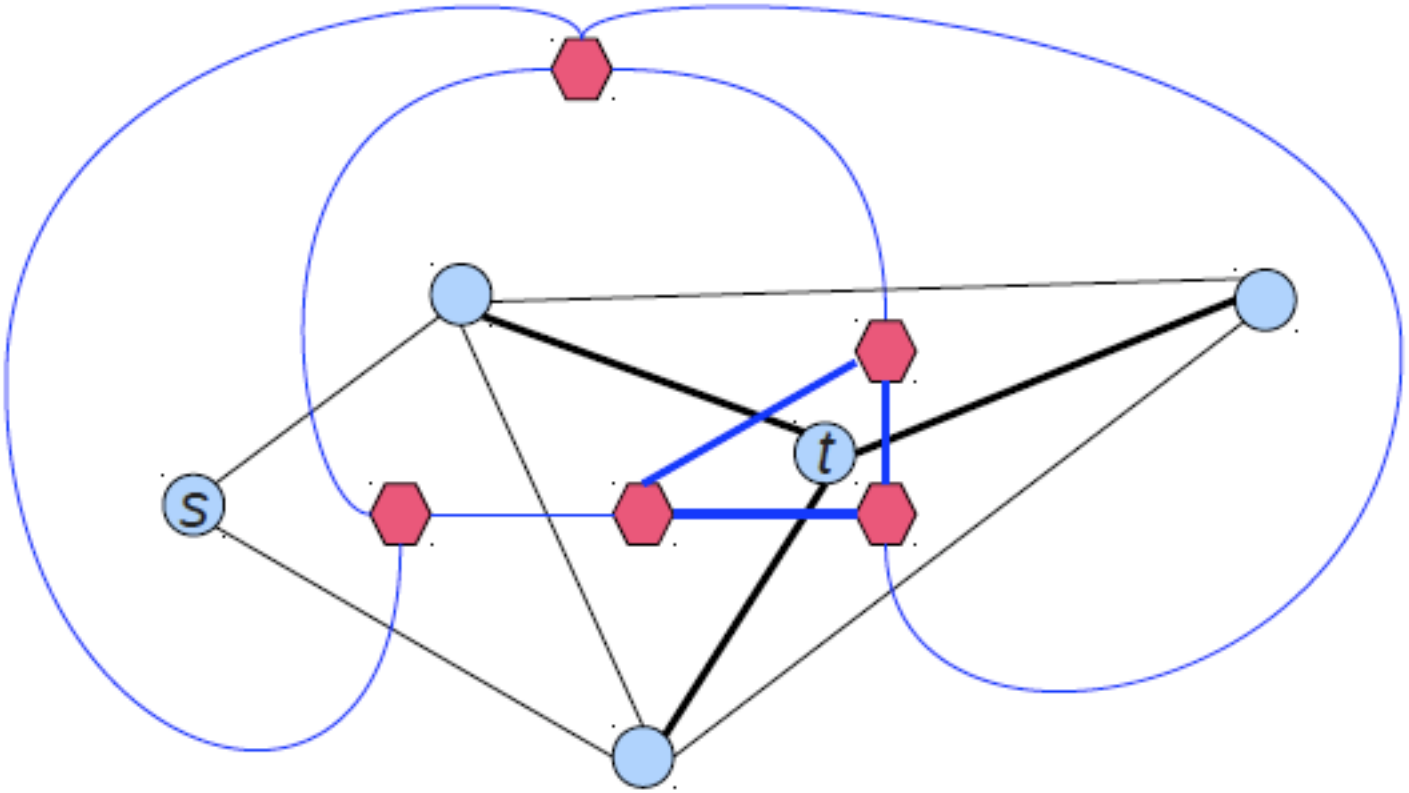
Dual Graph G^*

- Each face becomes a vertex
- Dual of e connects the faces adjacent to e
- The capacity of e is the length of its dual



Separating Cycles

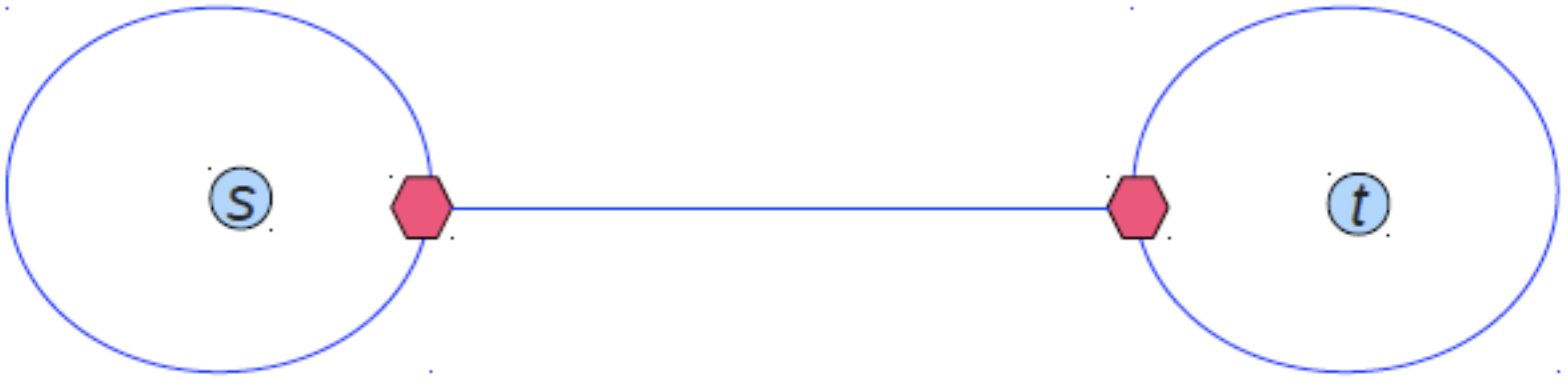
- Dual of an s-t cut is is a cycle that separates s and t
- A min s-t separating cycle in G defines a min s-t cut in G^* – Itai & Shiloach [1979]



Itai & Shiloach [1979]

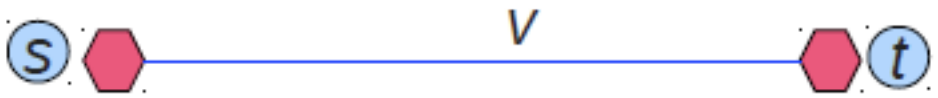
Itai & Shiloach [1979]

- Find the shortest path P from a face s* to a face t*



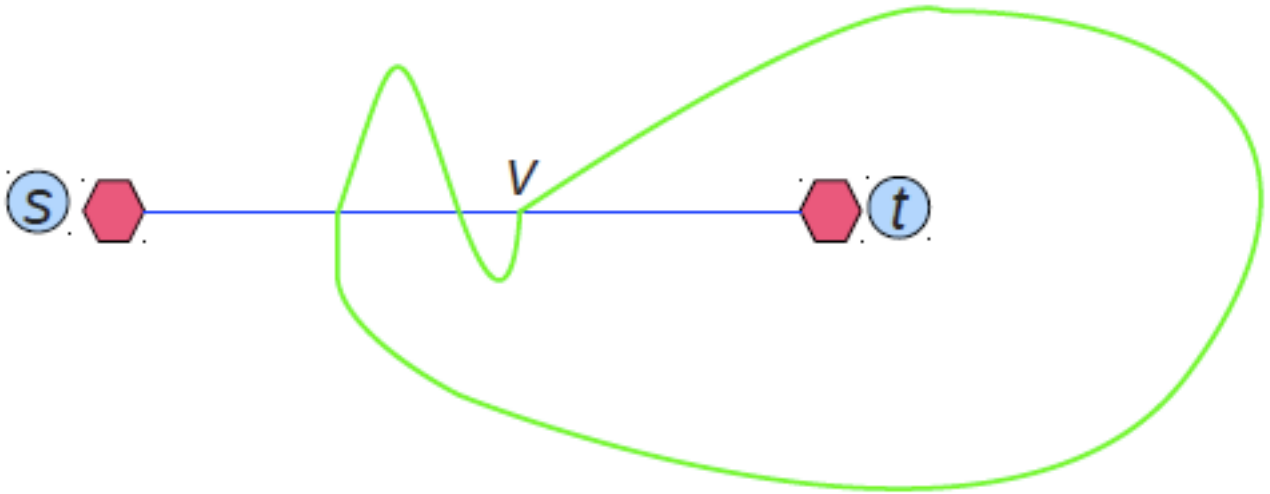
Itai & Shiloach [1979]

- Find the shortest path P from a face s* to a face t*
- The min s-t separating cycle that contains a specific vertex of P must cross P exactly once



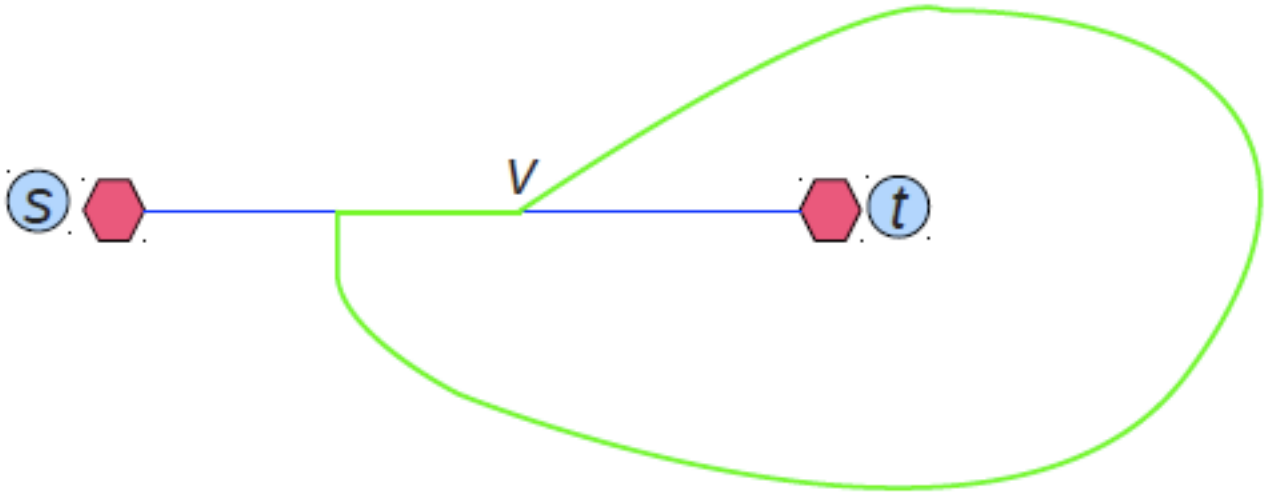
Itai & Shiloach [1979]

- Find the shortest path P from a face s* to a face t*
- The min s-t separating cycle that contains a specific vertex of P must cross P exactly once



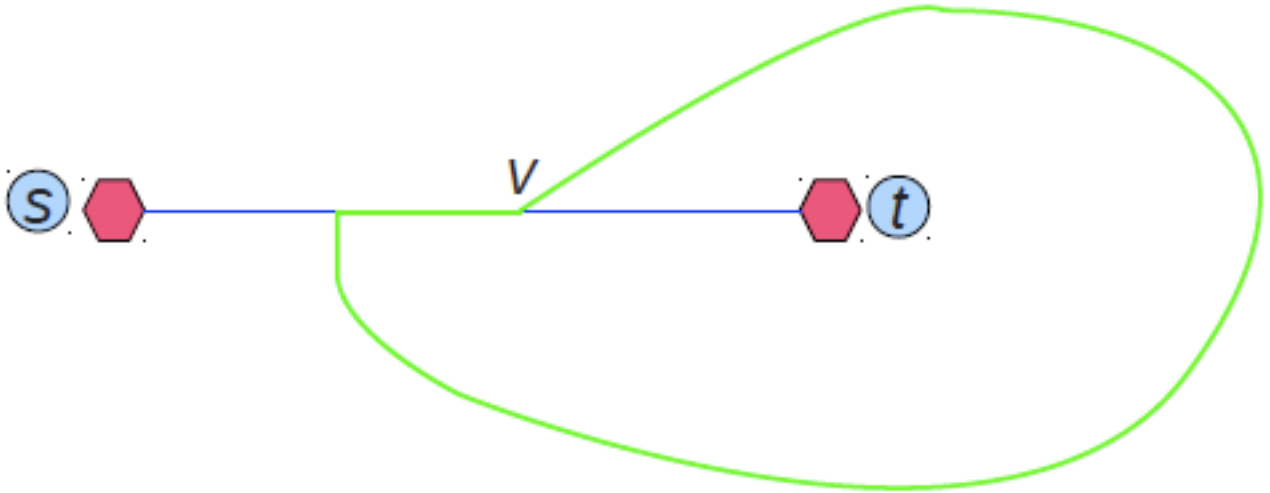
Itai & Shiloach [1979]

- Find the shortest path P from a face s* to a face t*
- The min s-t separating cycle that contains a specific vertex of P must cross P exactly once



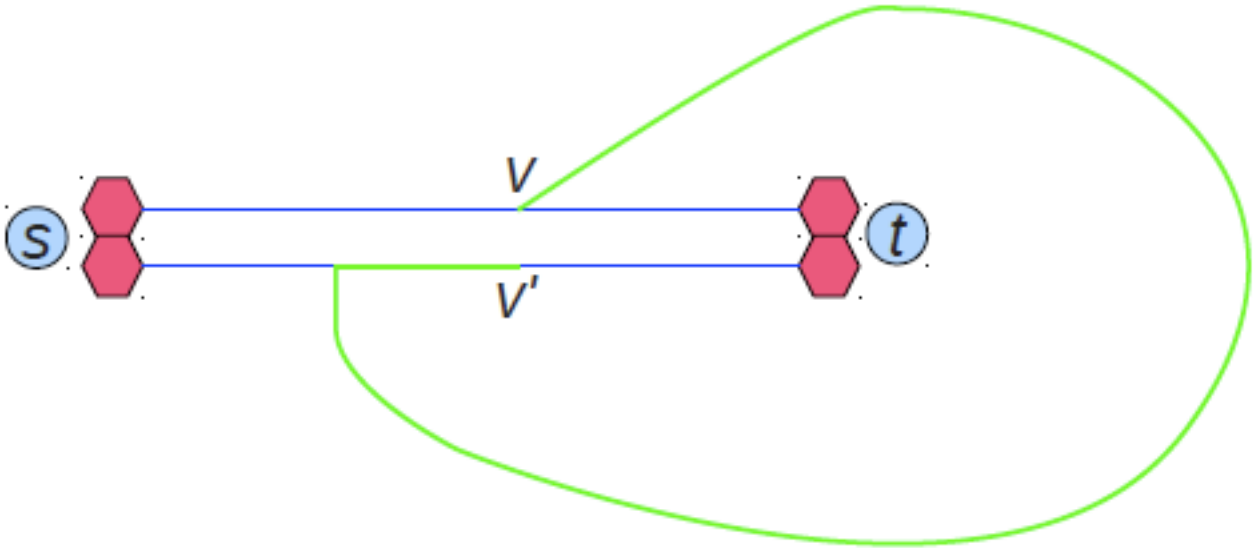
Itai & Shiloach [1979]

- Cut the graph along P



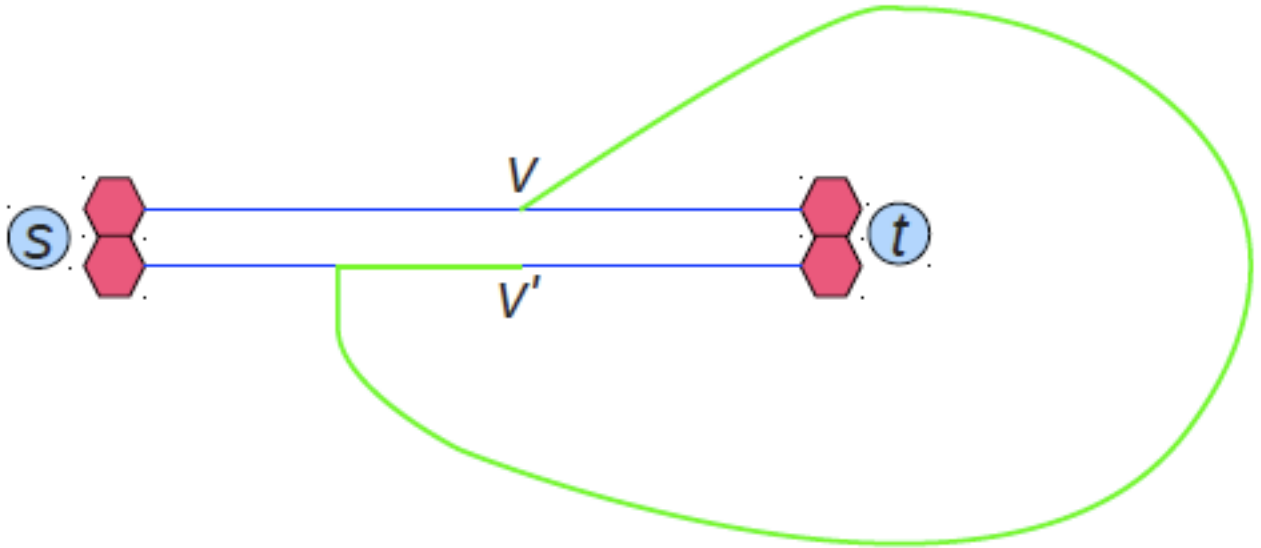
Itai & Shiloach [1979]

- Cut the graph along P



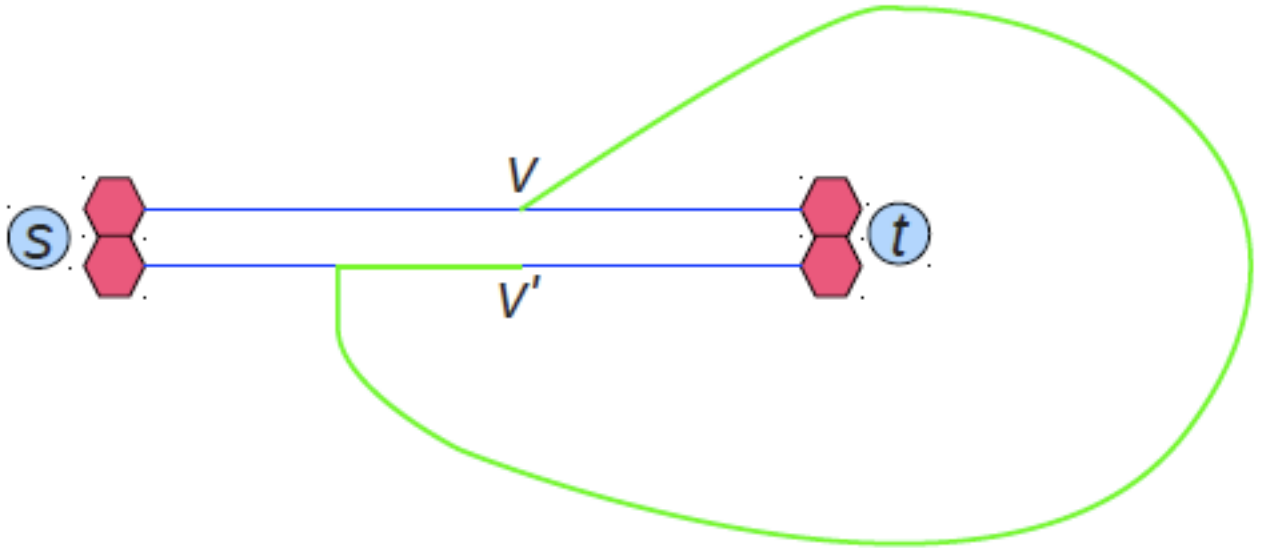
Itai & Shiloach [1979]

- Cut the graph along P
- Find a shortest path from each vertex of P to its copy
  - This gives us a min s-t separating cycle that contains each vertex of P



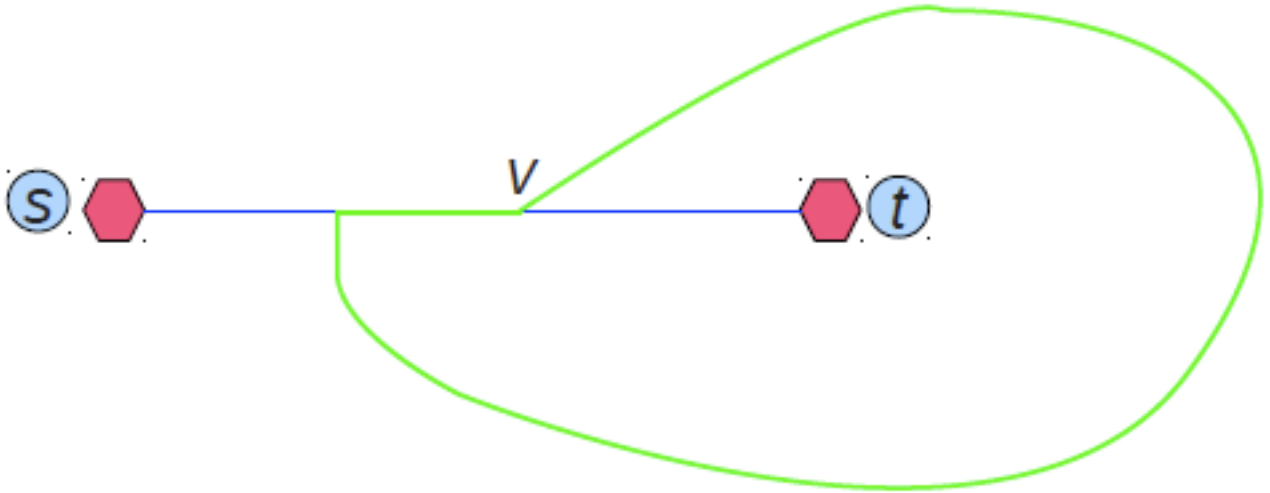
Itai & Shiloach [1979]

- Cut the graph along P
- Find a shortest path from each vertex of P to its copy
  - This gives us a min s-t separating cycle that contains each vertex of P
- O($n^2 \log n$) time
  - The length of P × time for running Dijkstra's algorithm



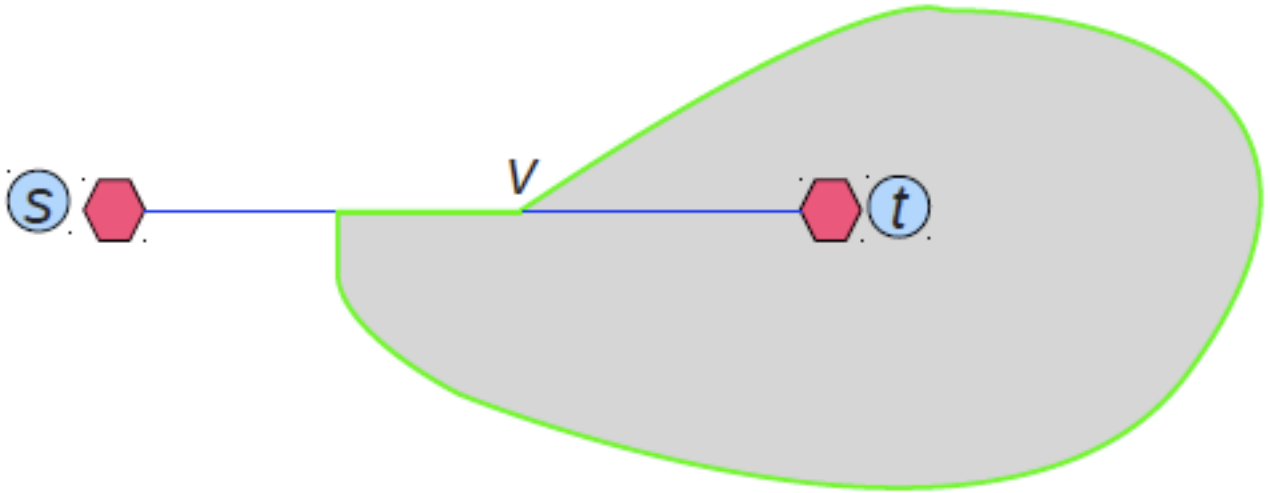
Reif [1983]

- Start with the middle vertex v of P



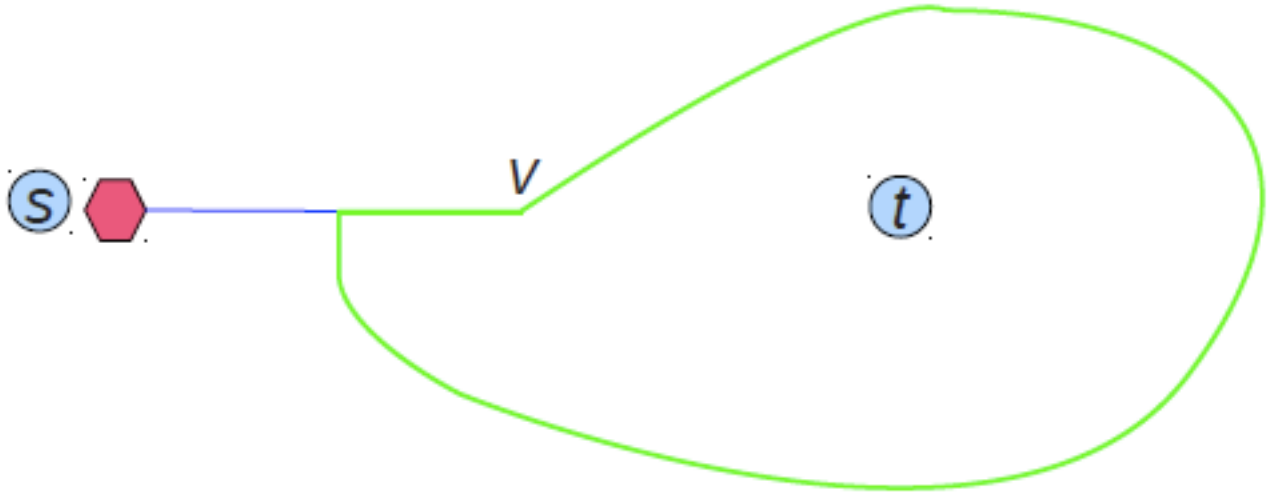
Reif [1983]

- Start with the middle vertex v of P
- Every min s-t separating cycle that contains a vertex of P is either inside, or outside the min s-t separating cycle that contains v



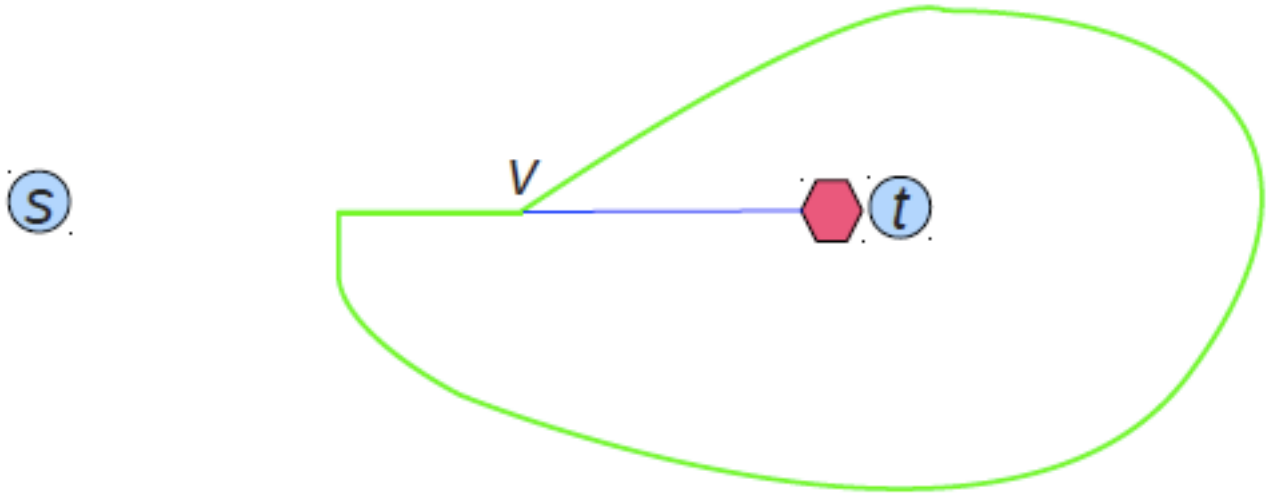
Reif [1983]

- Start with the middle vertex v of P
- Every min s-t separating cycle that contains a vertex of P is either inside, or outside the min s-t separating cycle that contains v



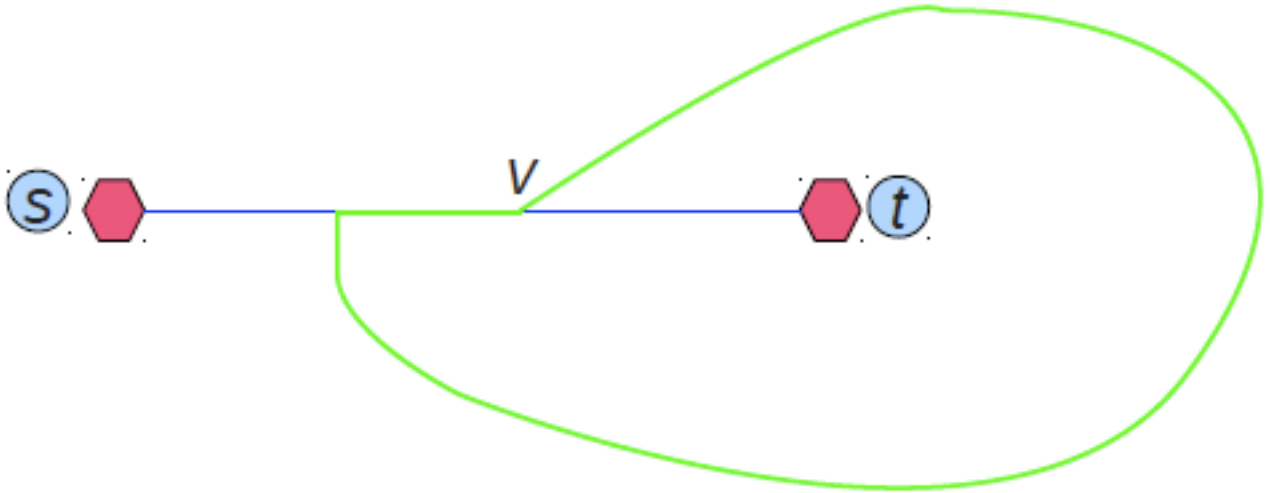
Reif [1983]

- Start with the middle vertex v of P
- Every min s-t separating cycle that contains a vertex of P is either inside, or outside the min s-t separating cycle that contains v

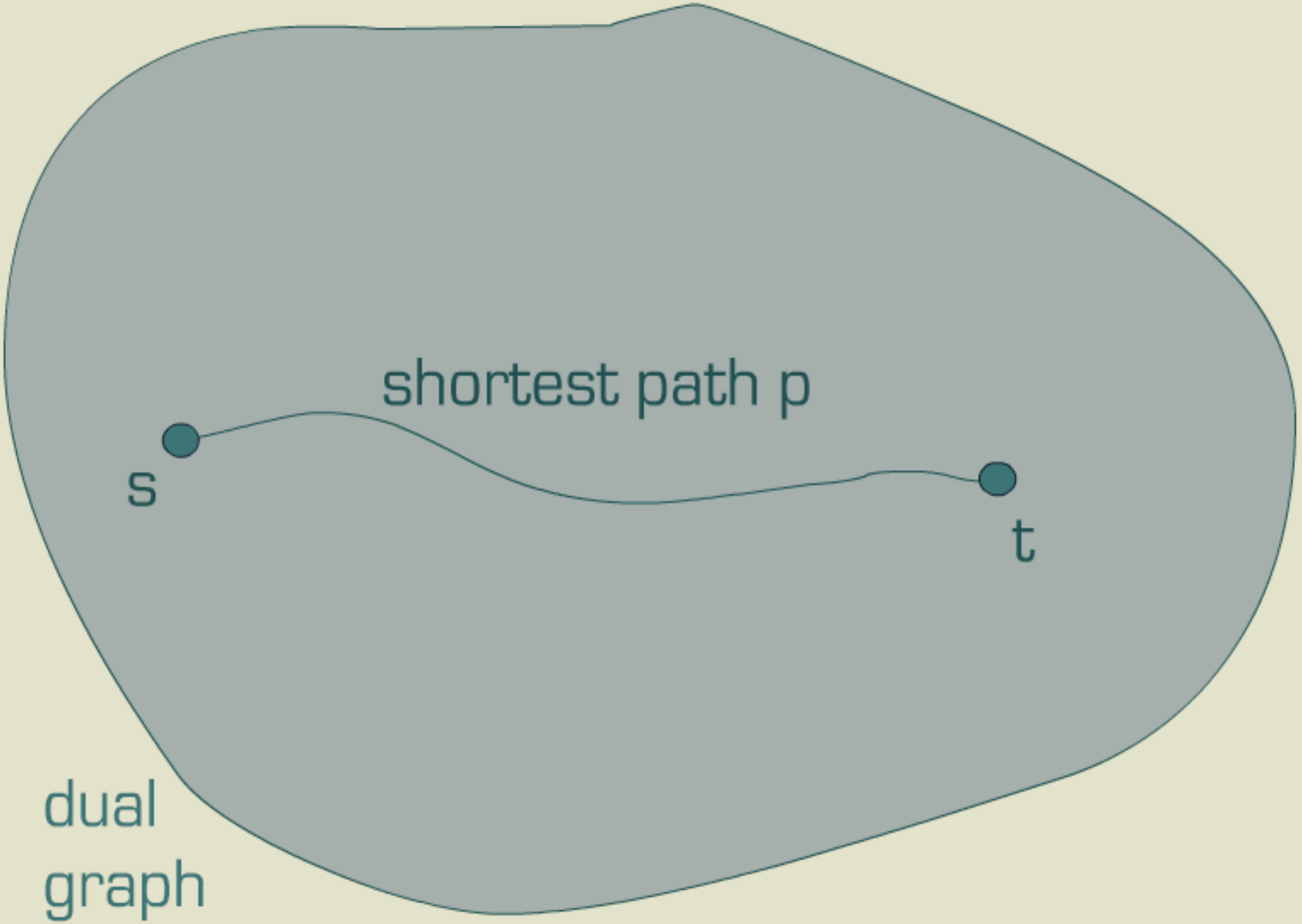


Reif [1983]

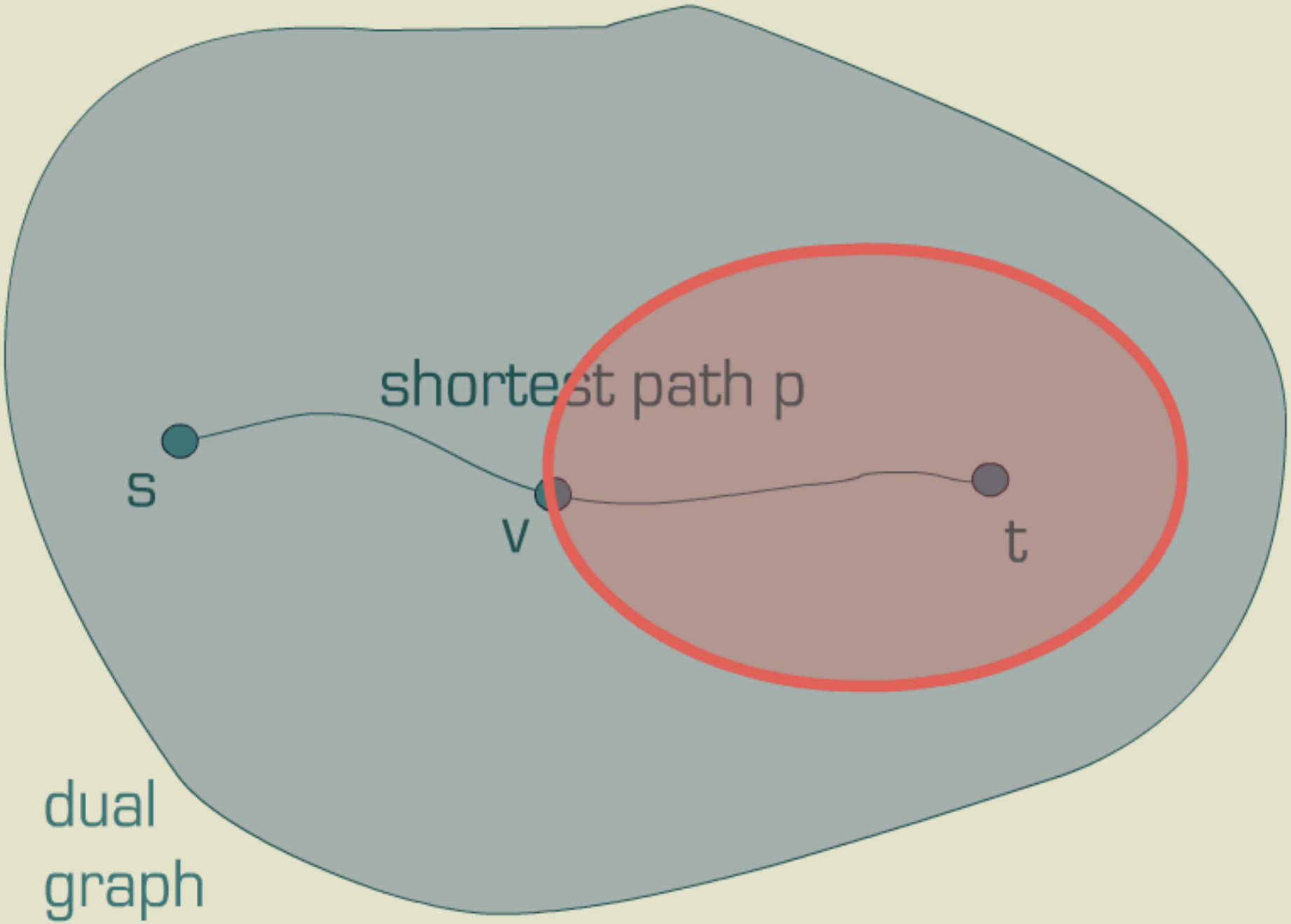
- Start with the middle vertex v of P
- Every min s-t cycle of a vertex of P is either inside, or outside the min s-t cycle that contains v
- Total time is $O(n \log^2 n)$
  - Recursion depth × time for running Dijkstra's algorithm

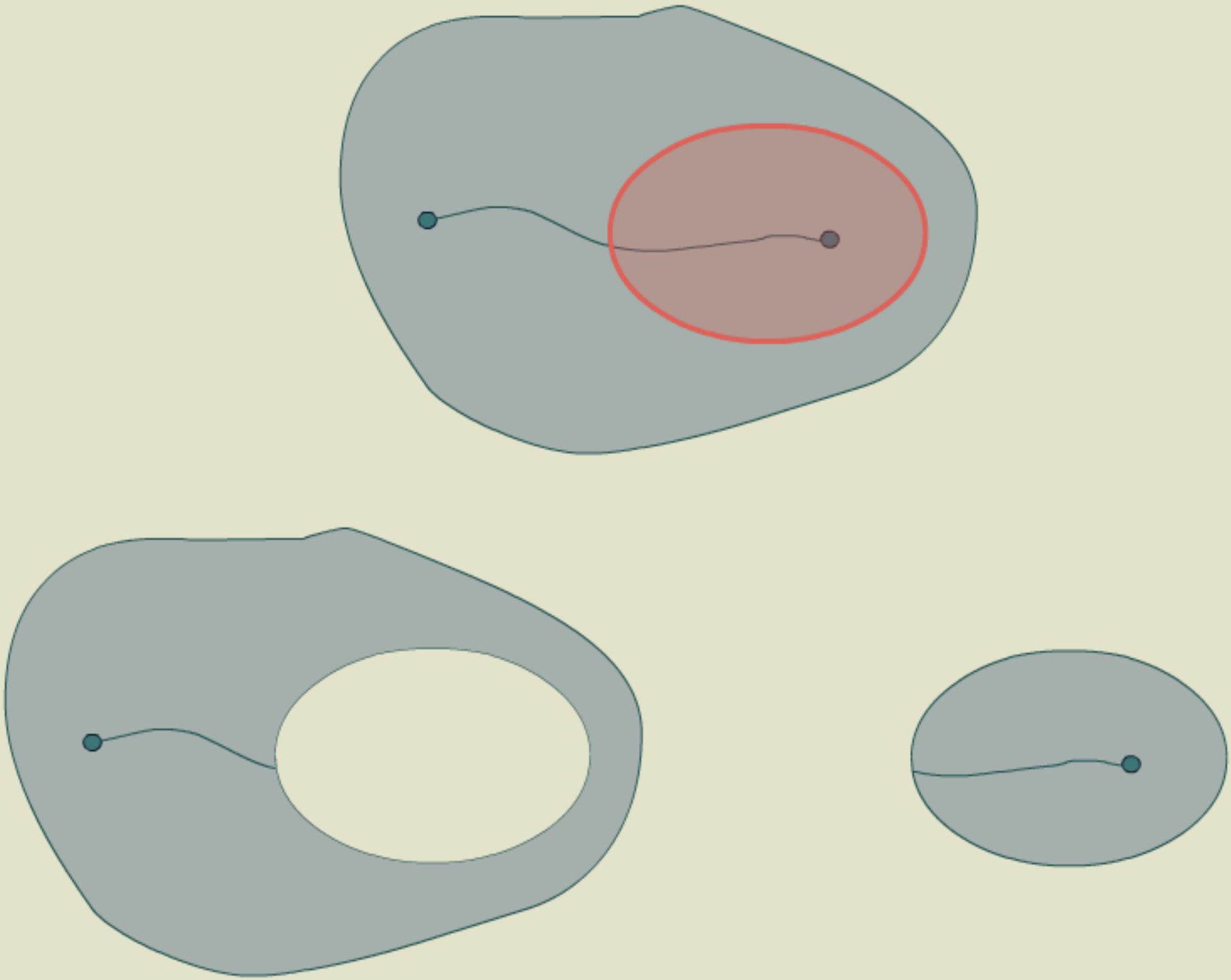


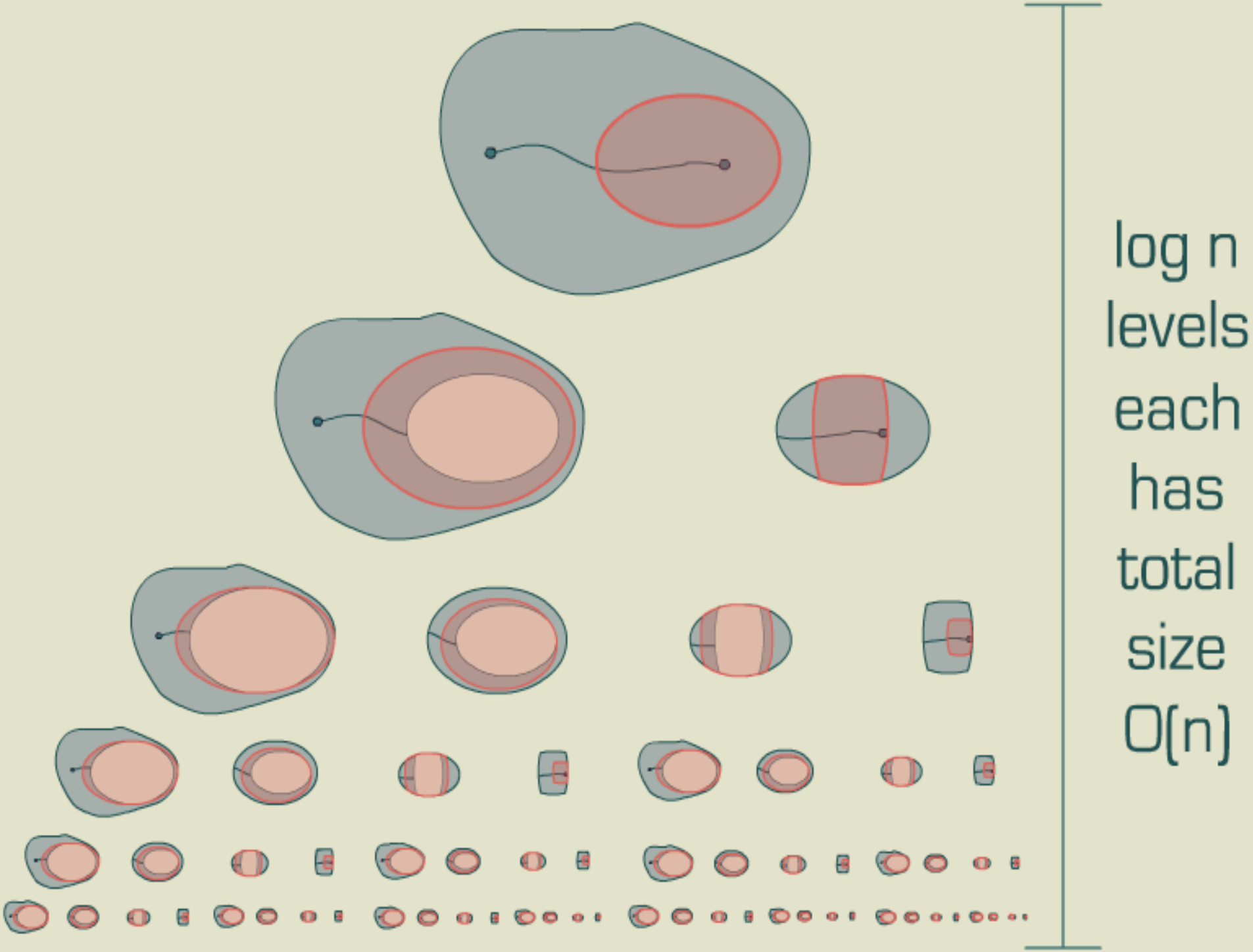
Reif's Algorithm



Reif's Algorithm






log n
levels
each
has
total
size
O(n)

State of the Art

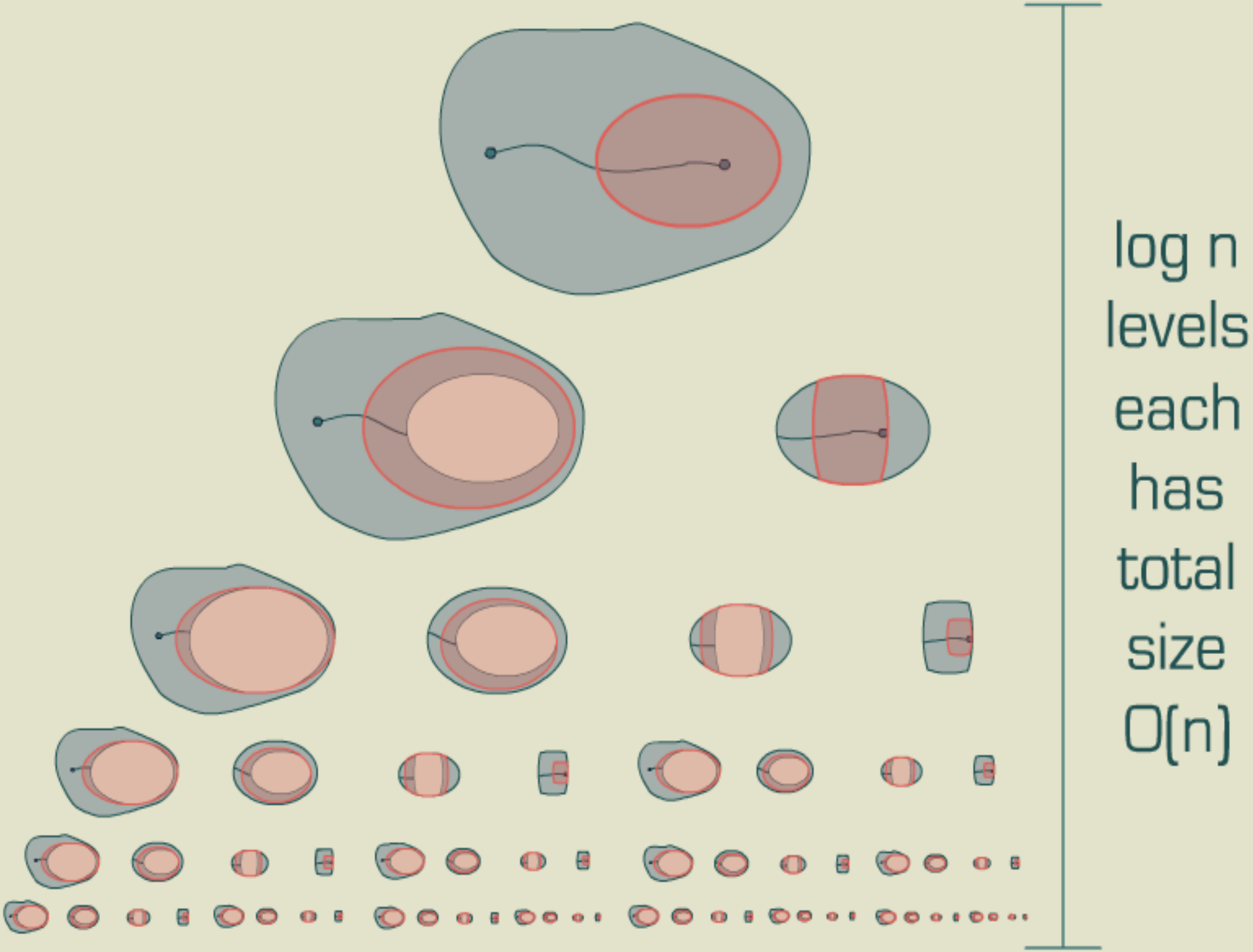
- Itai & Shiloach [1979] – $O(n^2 \log n)$ time for min cut
- Reif [1983] – $O(n \log^2 n)$ time for min cut
- Hassin & Johnson [1985] – Find also max flow
- Frederickson [1987] – $O(n \log n)$ time, with $O(n (\log n)^{1/2})$ shortest path algorithm
- Johnson [1987] – Also for directed graphs (parallel algorithm)
- Henzinger et al. [1997] – $O(n \log n)$ time, with faster $O(n)$ shortest path algorithm

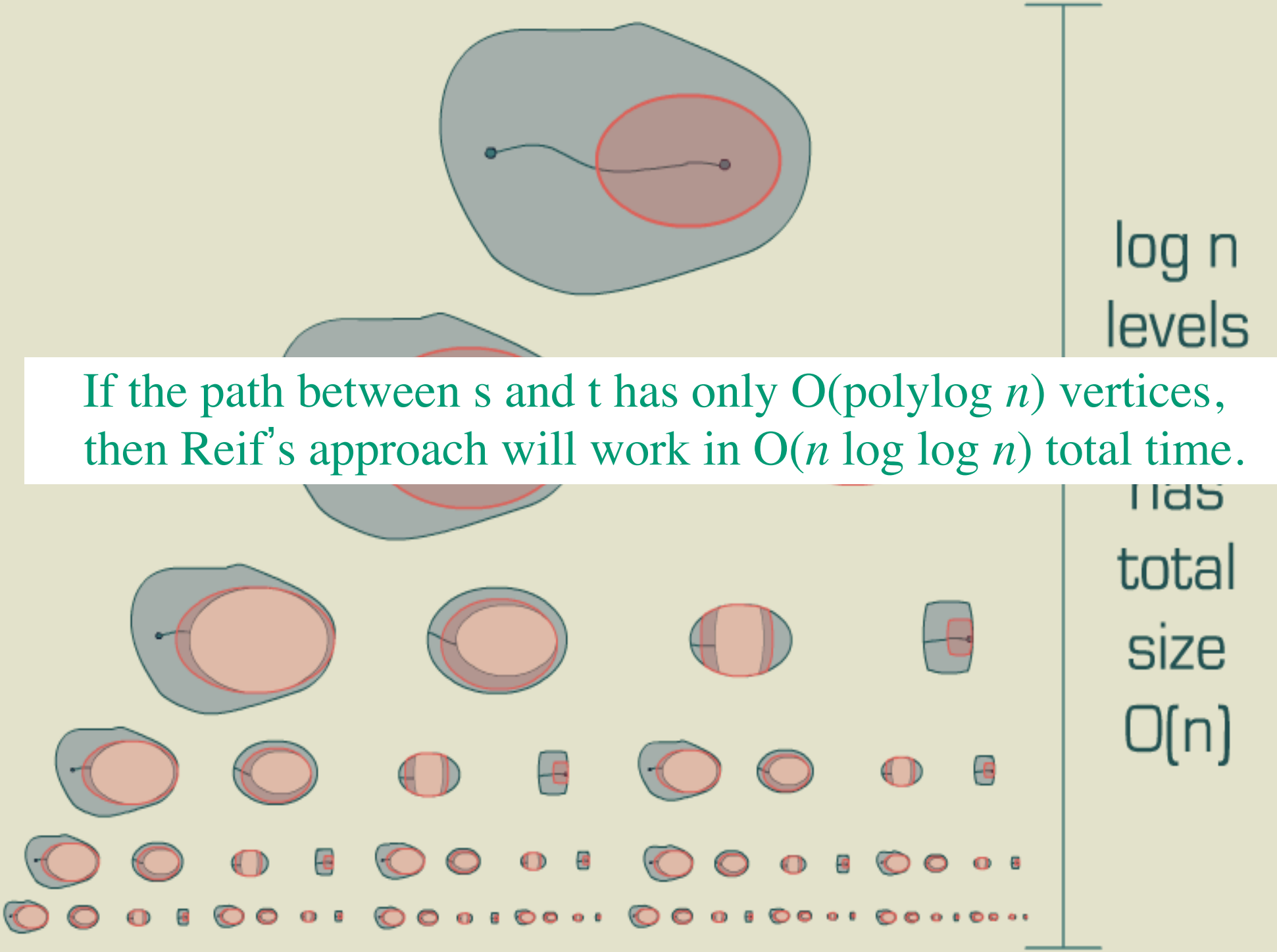
Our Contribution

- Itai & Shiloach [1979] – $O(n^2 \log n)$ time for min cut
- Reif [1983] – $O(n \log^2 n)$ time for min cut
- Hassin & Johnson [1985] – Find also max flow
- Frederickson [1987] – $O(n \log n)$ time, with faster $O(n (\log n)^{1/2})$ shortest path algorithm
- Johnson [1987] – Also for directed graphs (parallel algorithm)
- Henzinger et al. [1997] – $O(n \log n)$ time, with faster $O(n)$ shortest path algorithm
- This talk – $O(n \log \log n)$ time

Our Contribution

- Min s-t cut in undirected planar graphs can be solved in $O(n \log \log n)$ time
- Max s-t flow in undirected planar graphs can be also solved in $O(n \log \log n)$ time
- Min s-t cut and max s-t flow can be maintained dynamically in $O(n^{2/3} \text{polylog}(n))$ time per update operation


log n
levels
each
has
total
size
O(n)



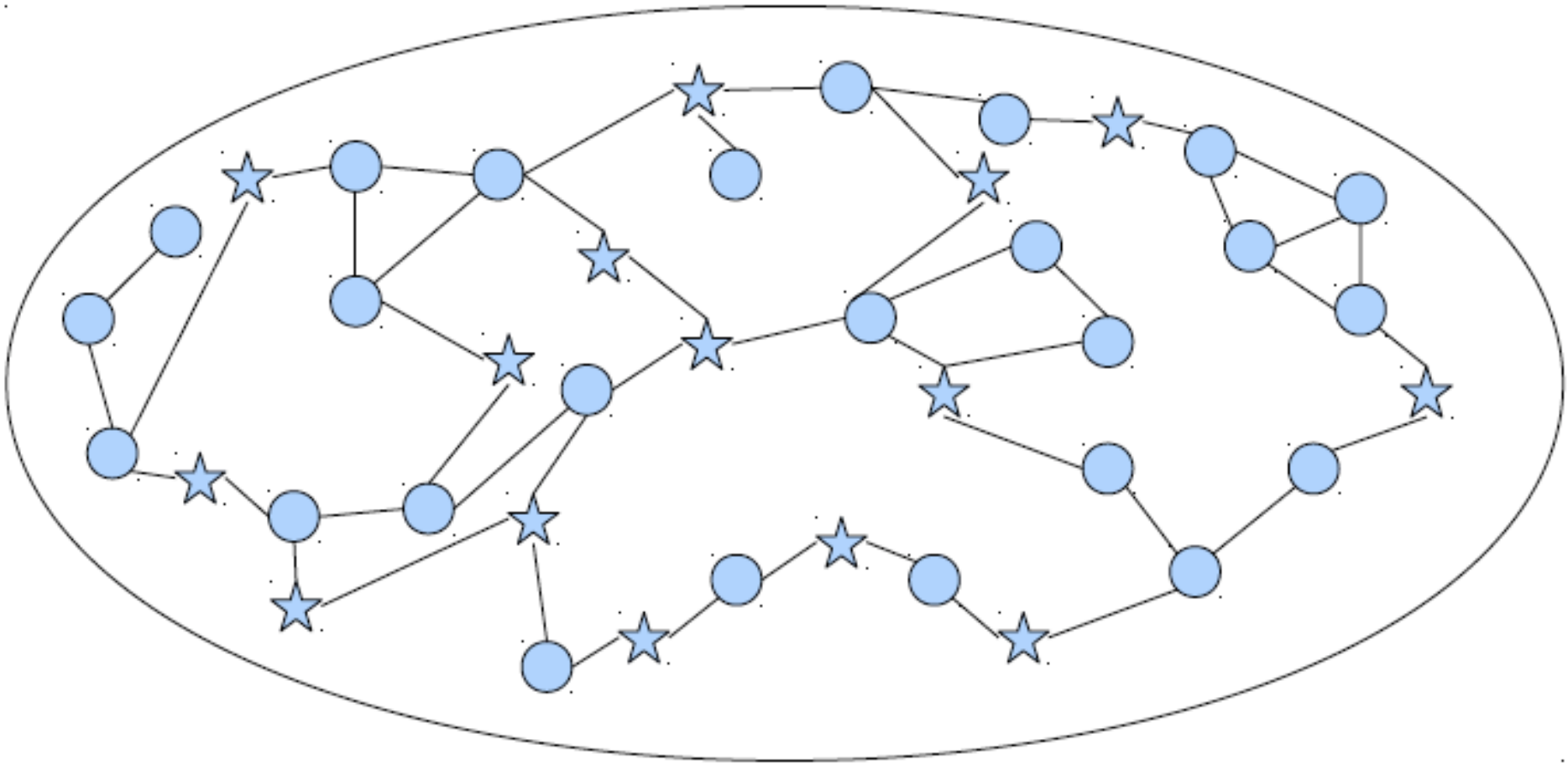
If the path between s and t has only O(polylog n) vertices, then Reif's approach will work in O(n log log n) total time.

Outline

- ✔ Introduction
- ✔ History
- Dense Distance Graphs
- Minimum s-t cut in O(n log log n) time
- Applications

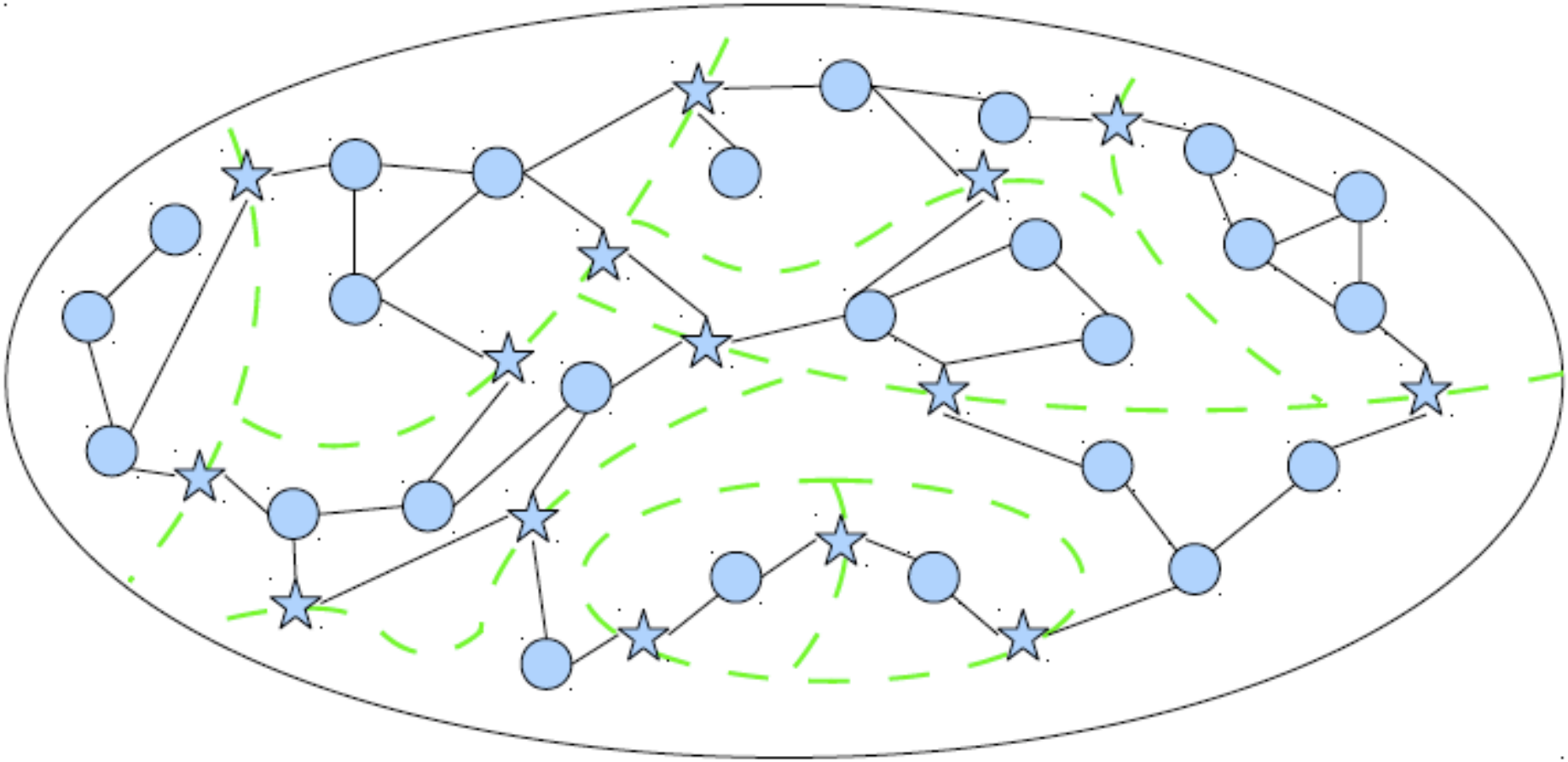
$(\log^6 n)$-division

- We divide the graph into pieces
- $O(n / \log^6 n)$ pieces, each with $O(\log^6 n)$ vertices and edges, $O(\log^3 n)$ boundary vertices and **O(1)** holes



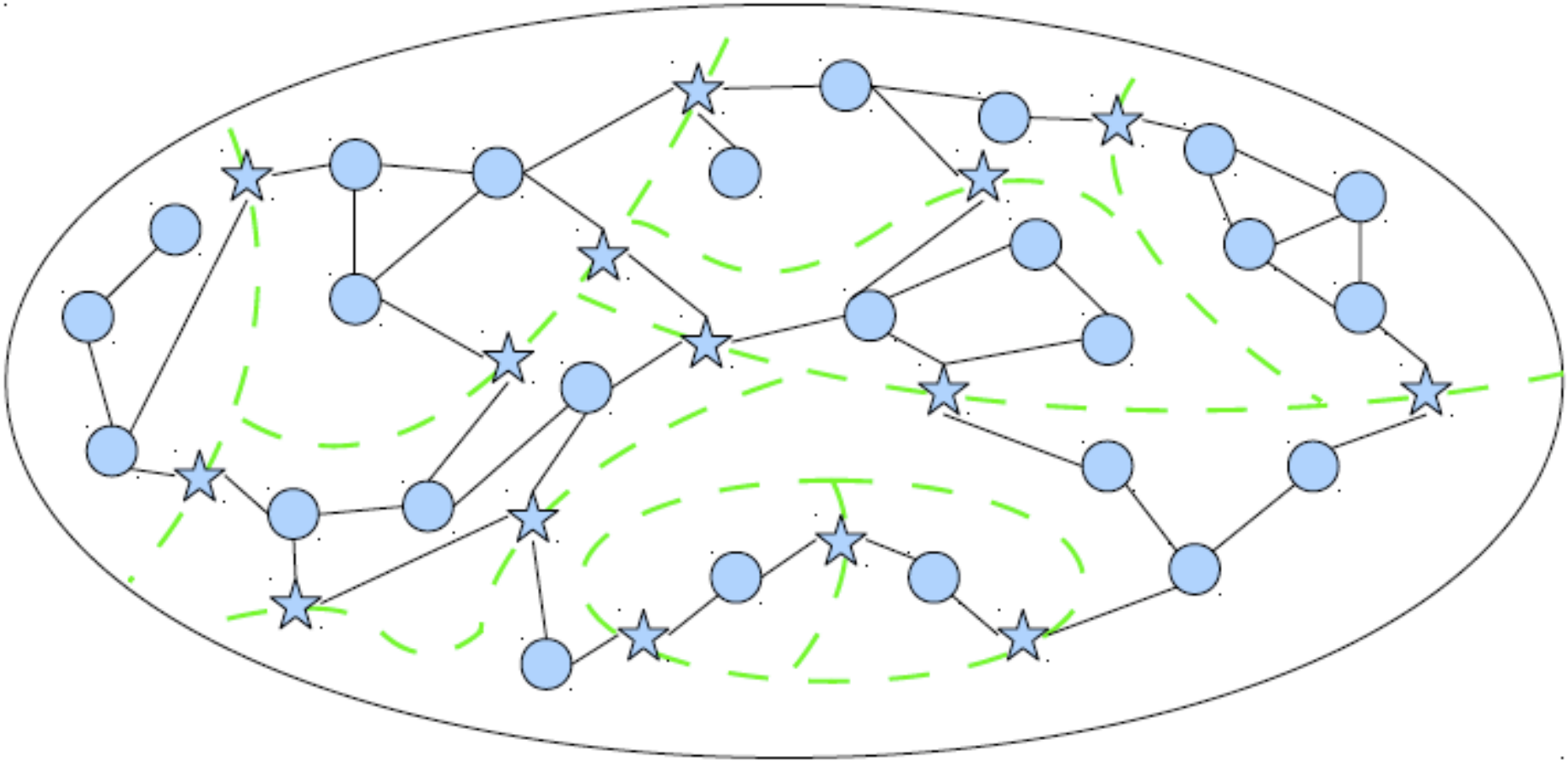
$(\log^6 n)$-division

- We divide the graph into pieces
- $O(n / \log^6 n)$ pieces, each with $O(\log^6 n)$ vertices and edges, $O(\log^3 n)$ boundary vertices and $O(1)$ holes



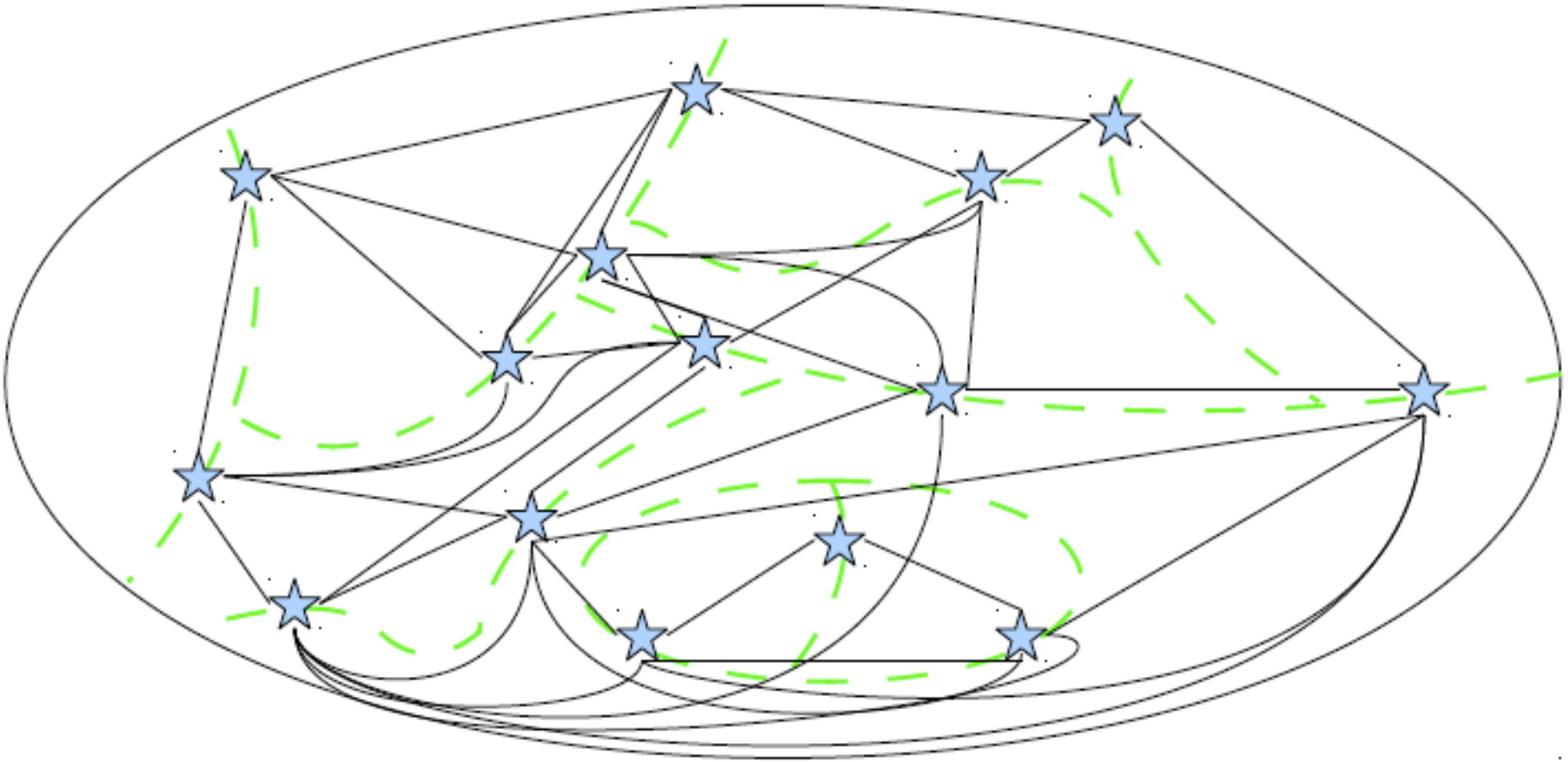
Dense Distance Graph

- Fakcharoenphol and Rao [2006]
- We are interested in distances among boundary vertices



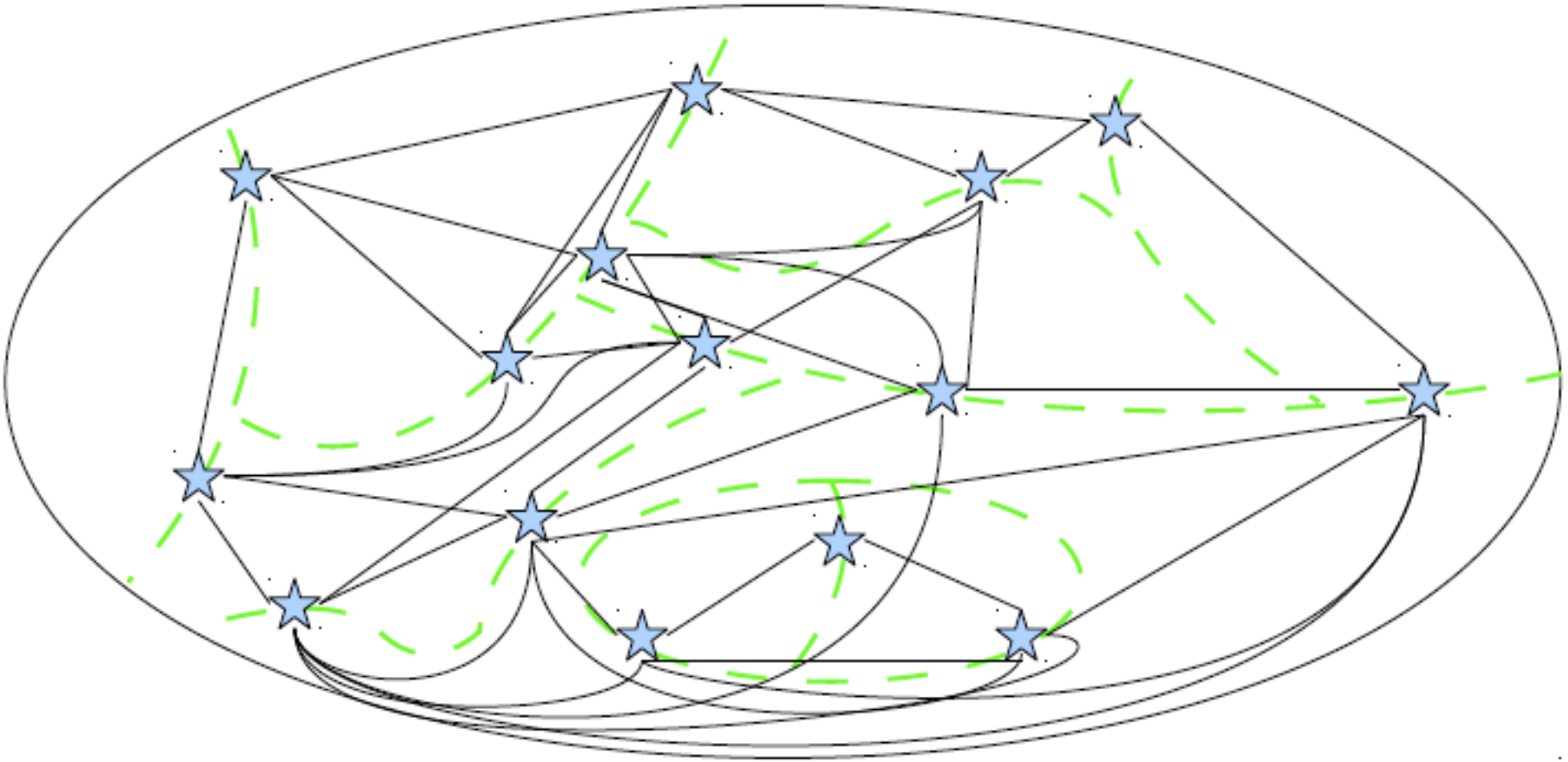
Dense Distance Graph

- Replace internal vertices with cliques on the boundary vertices
- The length of an edge is the distance between its endpoints



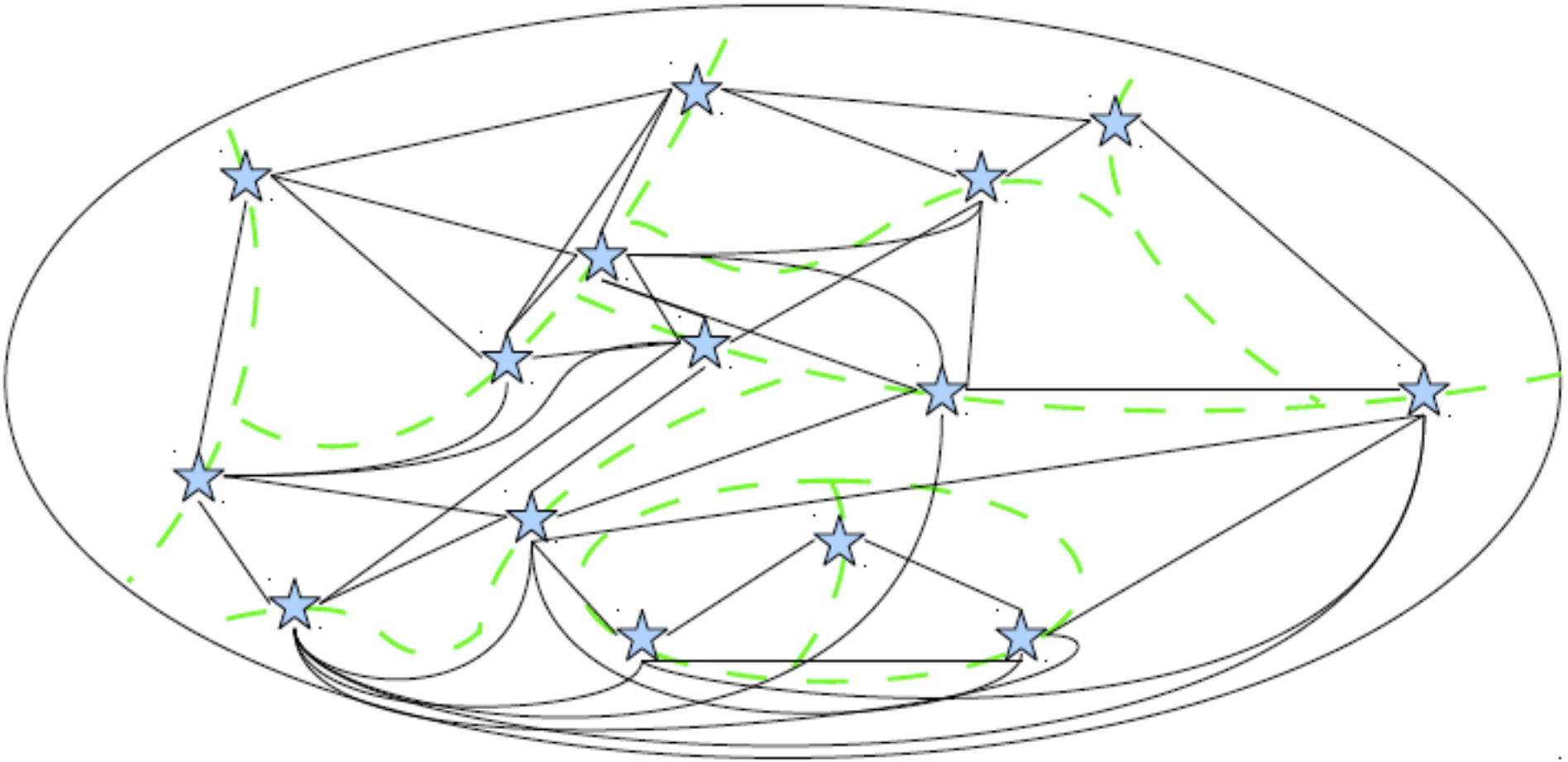
Dense Distance Graph

- Encode distances between boundary vertices
- Graph is no longer planar
- Much more edges but less vertices overall – $O(n / \log^3 n)$



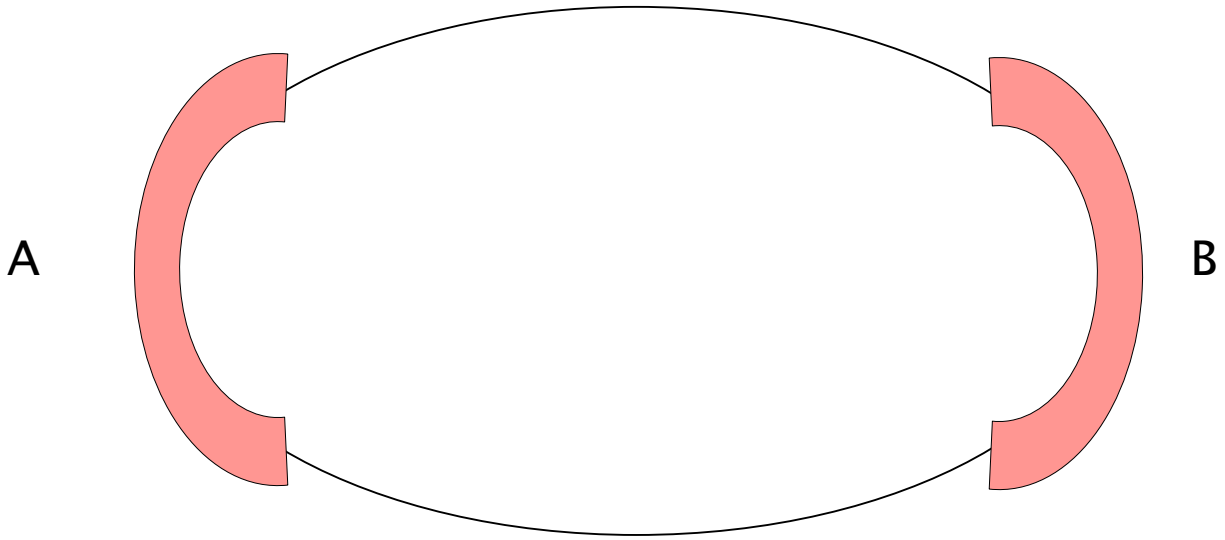
Dense Distance Graph

- Can compute the dense distance graph for a single piece in time $O(\log^6 n \log\log n)$ with the algorithm of Klein [2006] for multiple source shortest paths on a single face
- Total time is $O(n / \log^6 n) \times O(\log^6 n \log\log n) = O(n \log\log n)$



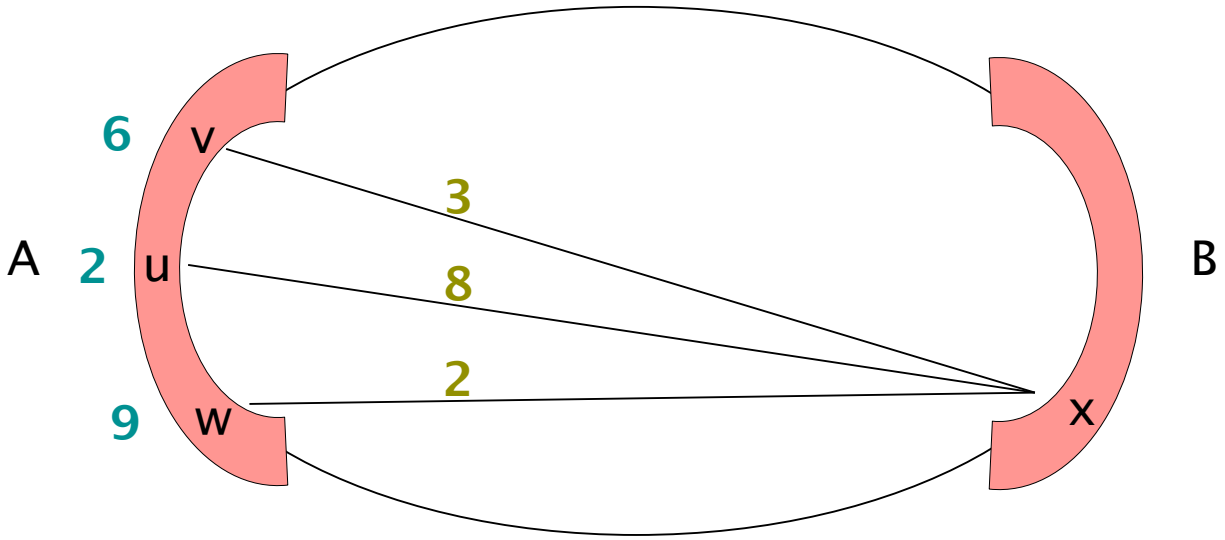
Distance between boundary vertices

- Consider the distances between two disjoint sets of consecutive boundary vertices A and B
- We have a complete bipartite graph between A and B



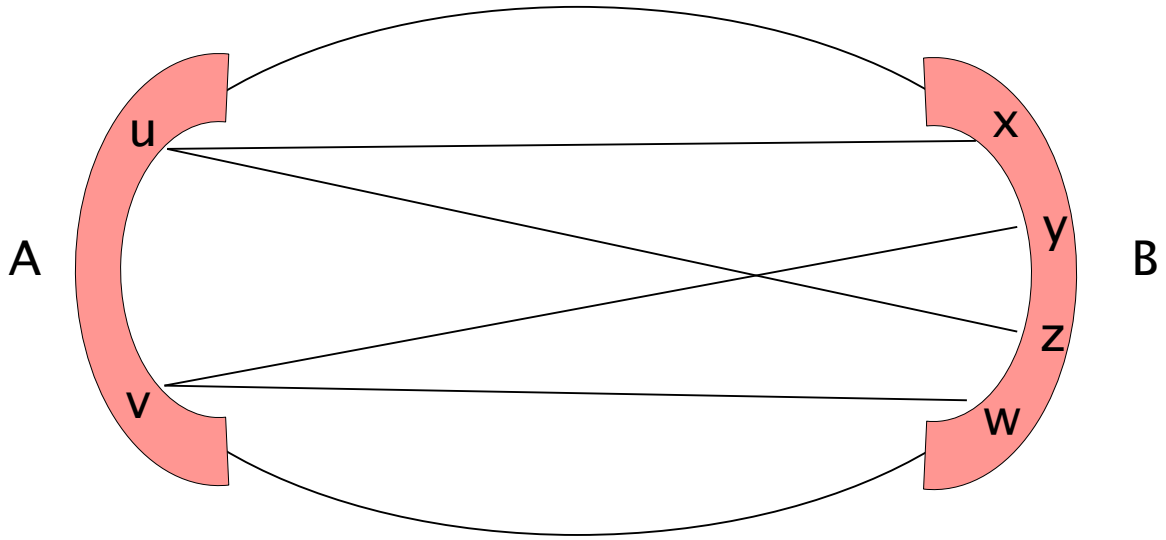
Distance between boundary vertices

- Fast implementation of Dijkstra after Fakcharoenphol & Rao [2006]
- Assume we have some distance labeling
- $v \in A$ is the parent of $x \in B$ if v minimizes $d(v) + d(v,x)$



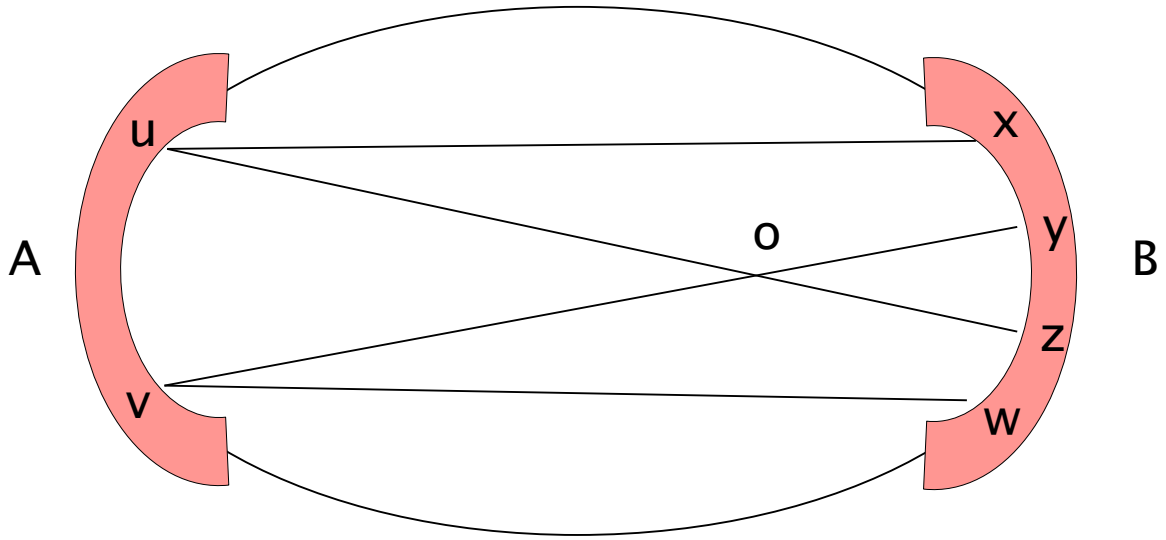
The non crossing property

- The children of a member of A are consecutive in B



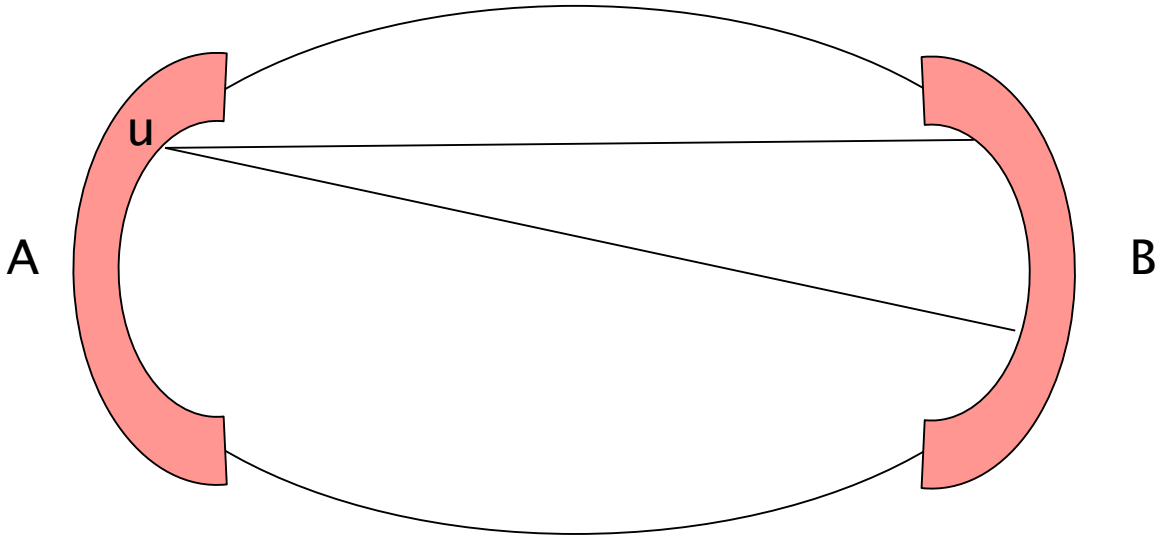
The non crossing property

- The children of a member of A are consecutive in B
- $d(u, z) < d(v, z) \Rightarrow d(u, o) < d(v, o) \Rightarrow$
  $d(u, y) < d(v, y)$



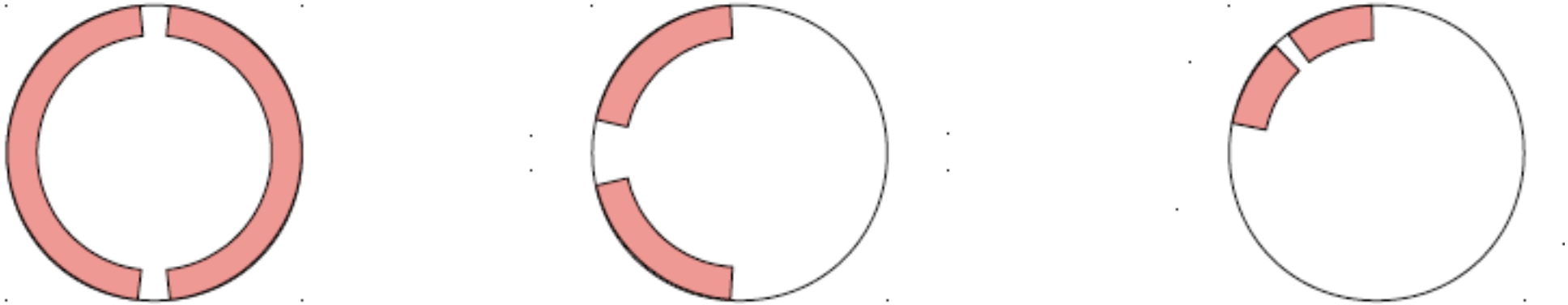
Fast Dijkstra with Global Heap

- Due to the non crossing property, we can run Dijkstra's shortest path algorithm more efficiently on the dense distance graph
- When a vertex in A gets a new label, we update the labels of all of its children in B at once
- This takes O(log n) time per vertex



Fast Dijkstra

- With O(log n) bipartite graphs we can cover all pairs of boundary vertices



- This gives O(b log^2 n) implementation of Dijkstra's algorithm for a graph composed of dense distance graphs with a total of b boundary vertices

Shortest paths in a ($\log^6 n$)-division

- We can divide the graph in $O(n \log \log n)$ time
  - Based on planar separators
  - Requires a new, more efficient algorithm (O(1) holes)
- We build the dense distance graphs on this division in $O(n \log \log n)$ time
- Using the fast Dijkstra implementation, we can find the shortest path between any two boundary vertices in $O(n / \log n)$ time:
  - $O(n / \log^6 n)$ pieces × $O(\log^3 n)$ boundary vertices per piece × extra $O(\log^2 n)$ per boundary vertex

Outline

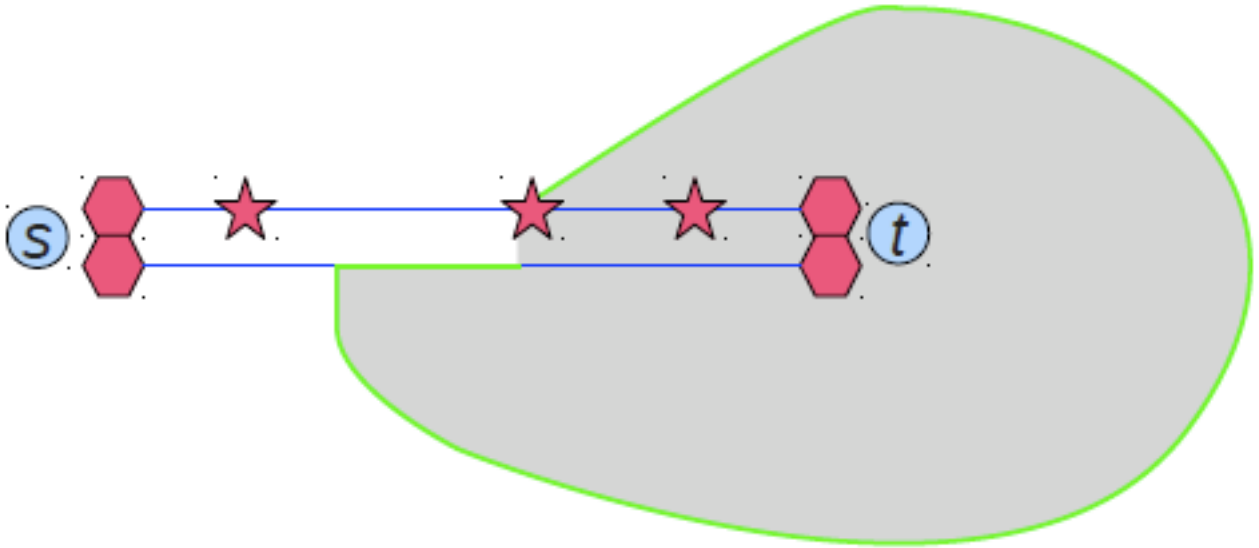
- ✔ Introduction
- ✔ History
- ✔ Dense Distance Graphs
- Minimum s-t cut in O(n log log n) time
- Applications

Our min s-t cut algorithm

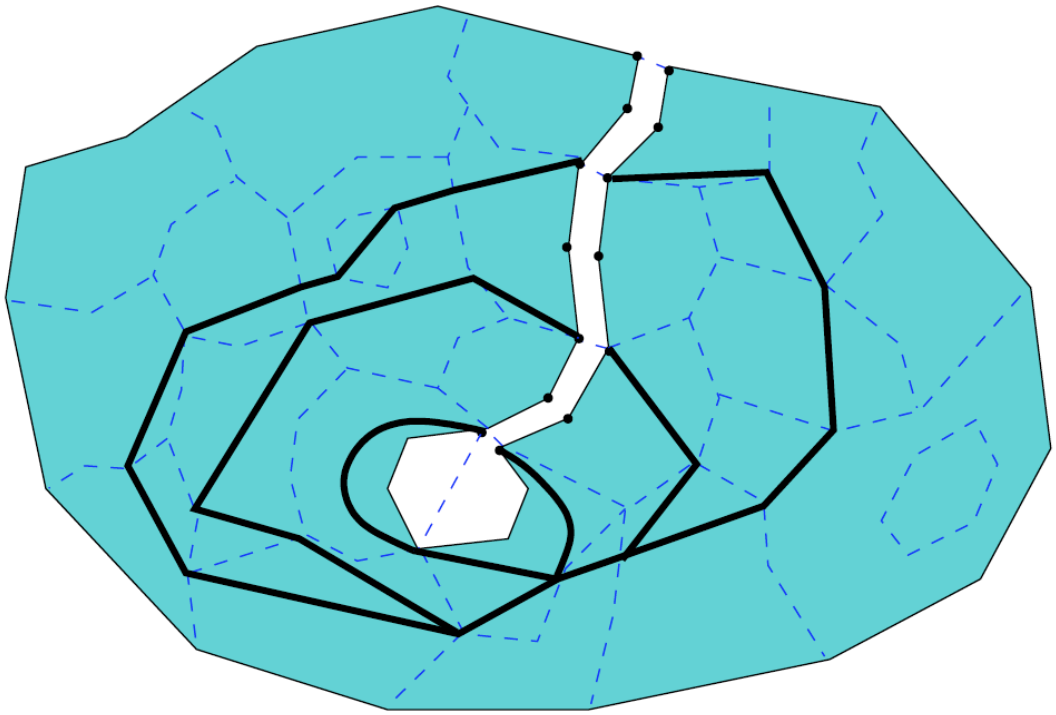
- Compute ($\log^6 n$)-division and dense distance graphs in $O(n \log \log n)$ time
- Based on Reif's min separating cycle algorithm
- Two levels of Reif's algorithm:
  - "Coarse Reif" on the dense distance graph
  - "Refined Reif" for the rest of the s* to t* shortest path P

“Coarse Reif”

- Find path P as in Reif's algorithm and cut open graph along it
- Run Reif's algorithm, but only on the boundary vertices in P
  - This takes $O(n / \log n) \times O(\log n)$ (fast Dijkstra × recursion depth) = $O(n)$ time
- Now have min s-t separating cycle containing each boundary vertex in P



“Coarse Reif”



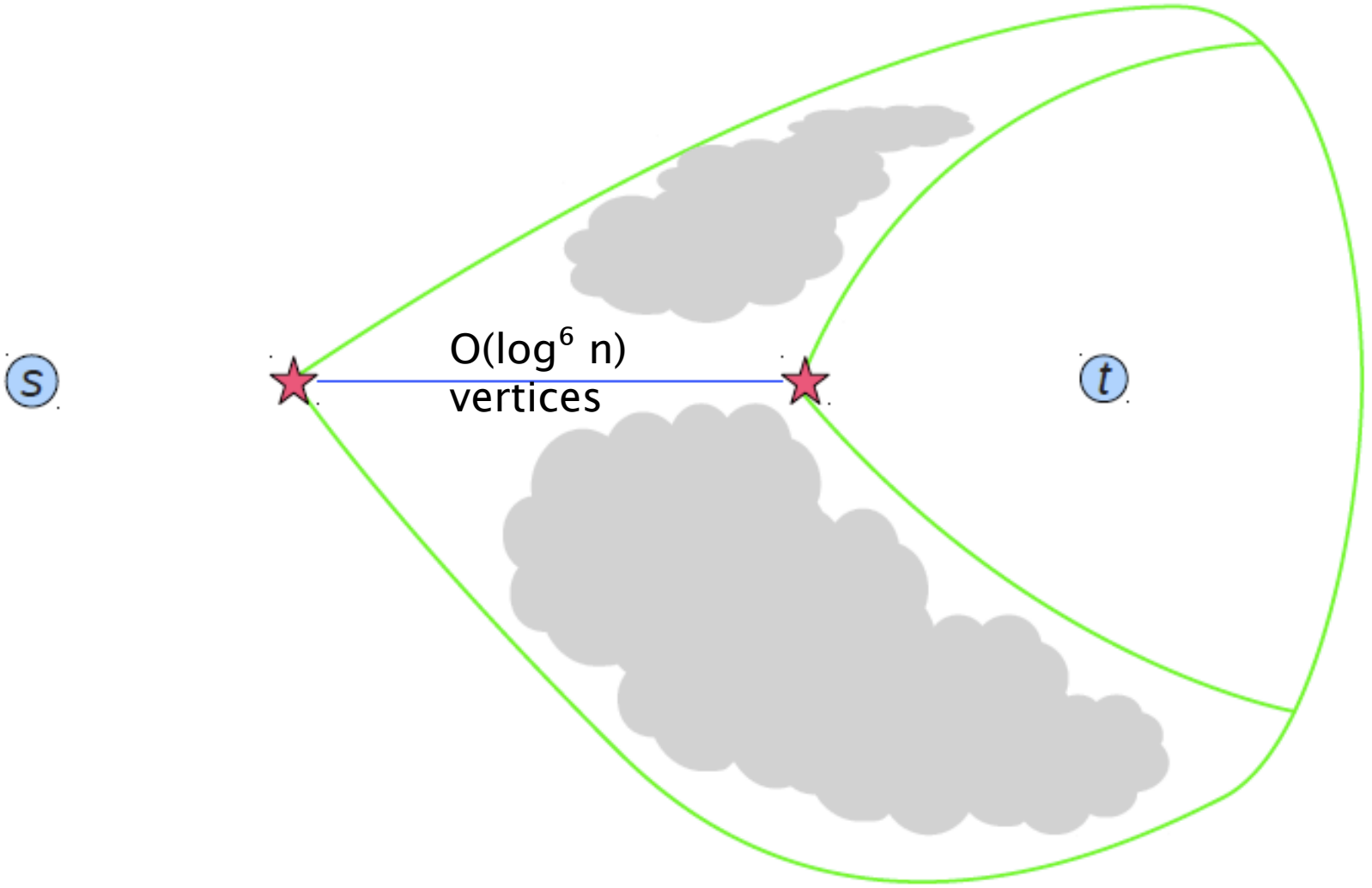
- Obtain min s-t separating cycles for boundary vertex in P
- The s-t separating cycles obtained partition G into subgraphs: each subgraph has subpath of P fully contained inside a piece
- Each such subpath has length $O(\log^6 n)$ [# edges in a piece]

“Refined Reif”

- Between two boundary vertices u, v continue with normal Reif algorithm

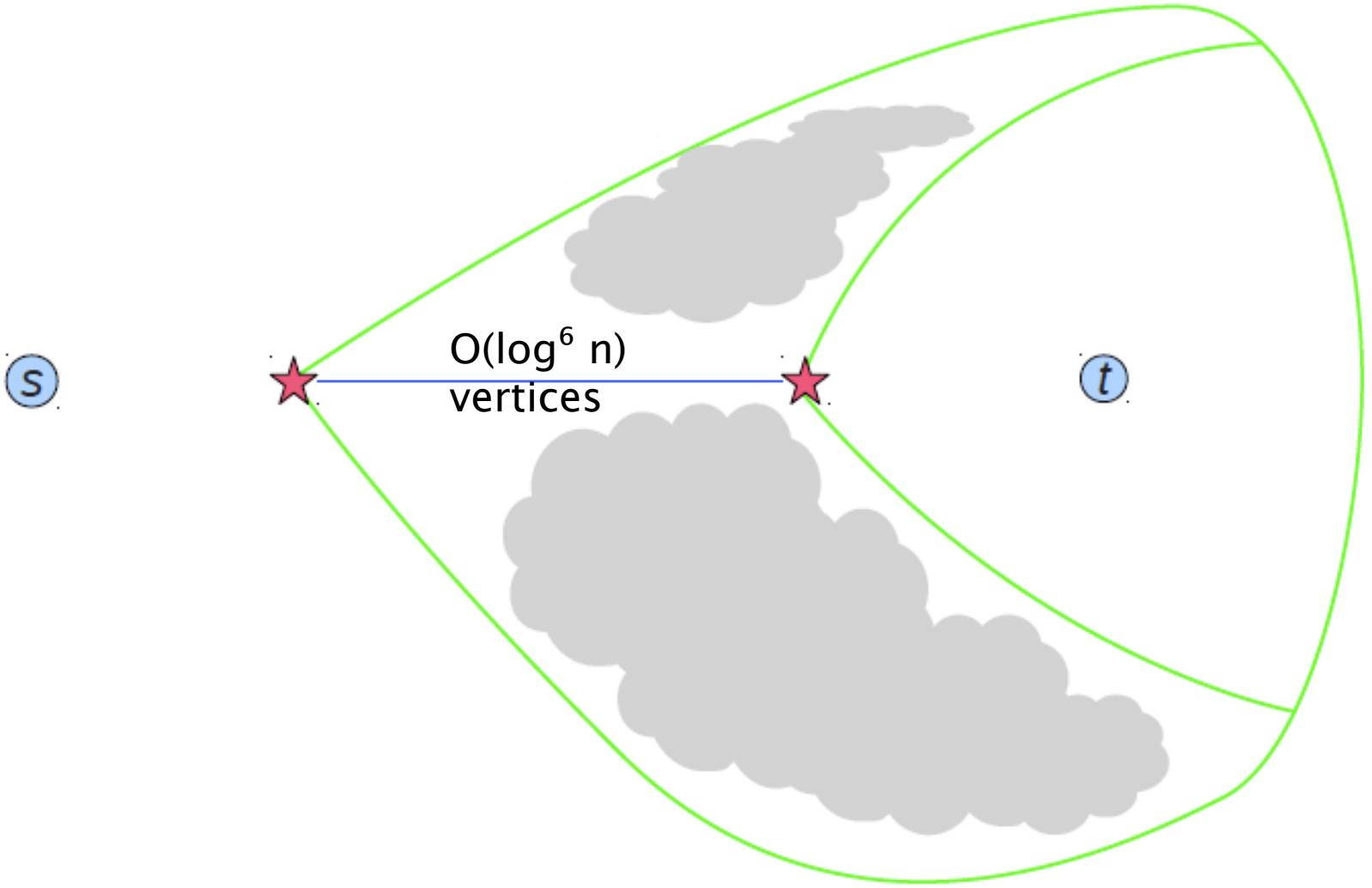
"Refined Reif"

- There are $O(\log^6 n)$ vertices of P between u and v



"Refined Reif"

- Depth of recursion is O(log log n)
- Make sure total size of all parts still O(n) (non-trivial)



Time Analysis

- O(n log log n) for constructing the (log^6 n)-division and the dense distance graph
- O(n) for "Coarse Reif" step
- O(n log log n) for "Refined Reif" step
- Obtain min s-t separating cycle for each vertex of P, exactly as Itai & Shiloach (but faster!)

Outline

- ✔ Introduction
- ✔ History
- ✔ Dense Distance Graphs
- ✔ Minimum s-t cut in O(n log log n) time
- Applications

Applications

- Computing min separating cycle in planar graphs is the bottleneck for the following problems:
  - Maximum flow in undirected planar graphs – Hassin & Johnson [1985], improved to $O(n \log \log n)$
  - Global minimum cut / girth of planar graphs – Charlermoosk et al. [2004], improved to $O(n \log n \log \log n)$
  - Shortest non-trivial cycle in undirected graphs on a surface with a bounded genus – Kutz [2006], improved to $O(g^{O(g)} n \log \log n)$
  - Minimum s-t cut in undirected graphs on a surface with a bounded genus – Chambers et al. [2009], improved to $O(g^{O(g)} n \log \log n)$

Dynamic data structure for min cut queries

- Extend efficient Dijkstra implementation of Fakcharoenphol and Rao [2006] to a dynamic shortest path queries data structure
  - Need to recompute pieces affected by edge insertion / deletion / capacity change
- This allows also a dynamic data structure for min cut algorithm, using our ideas
- $O(n^{2/3} \log^{8/3} n)$ time per operation