Shortest Paths in Directed Planar Graphs with Negative Lengths: a Linear-Space $O(n \log^2 n)$-Time Algorithm

Shay Mozes (Brown University)

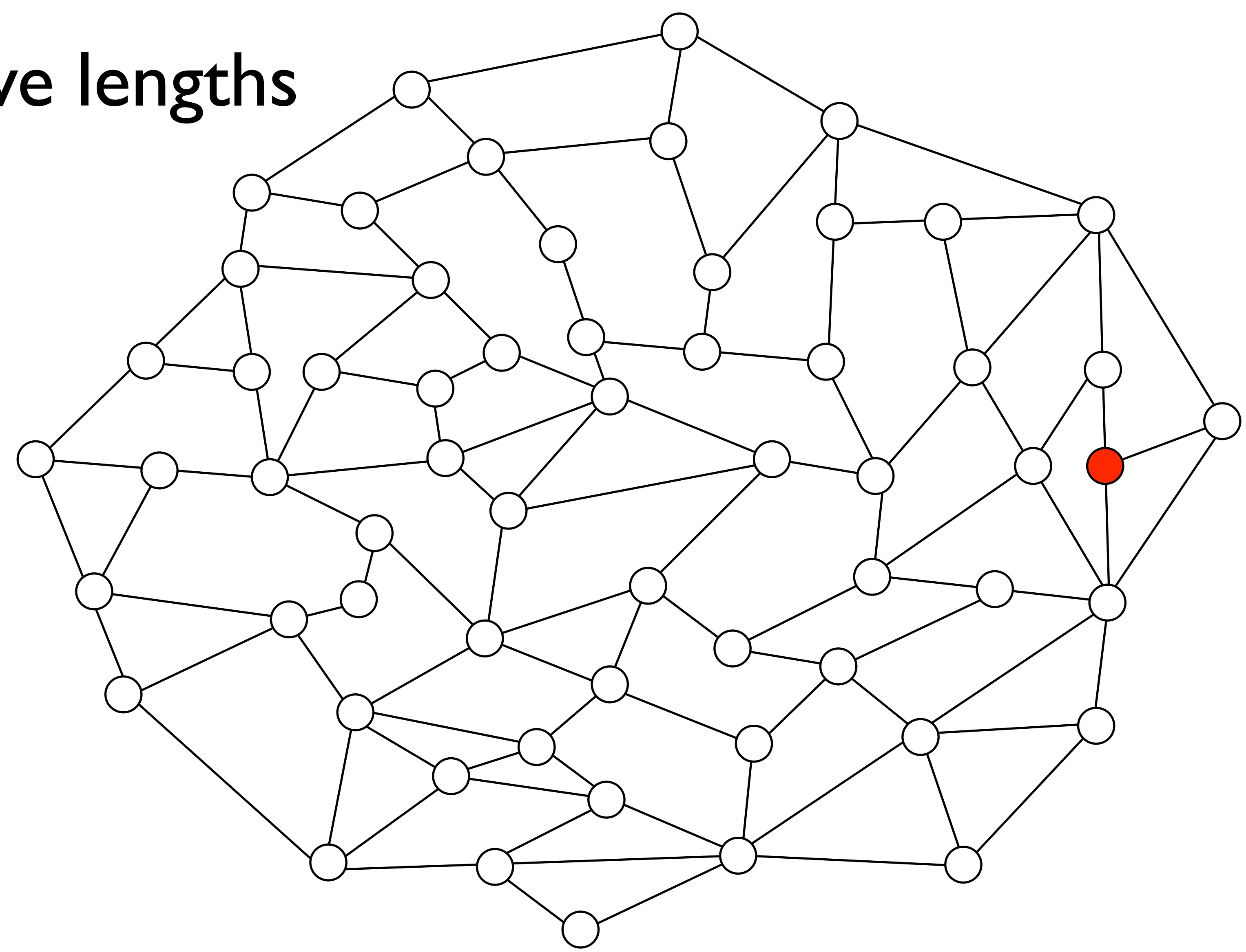
joint work with

Philip Klein (Brown University)

Oren Weimann (MIT)

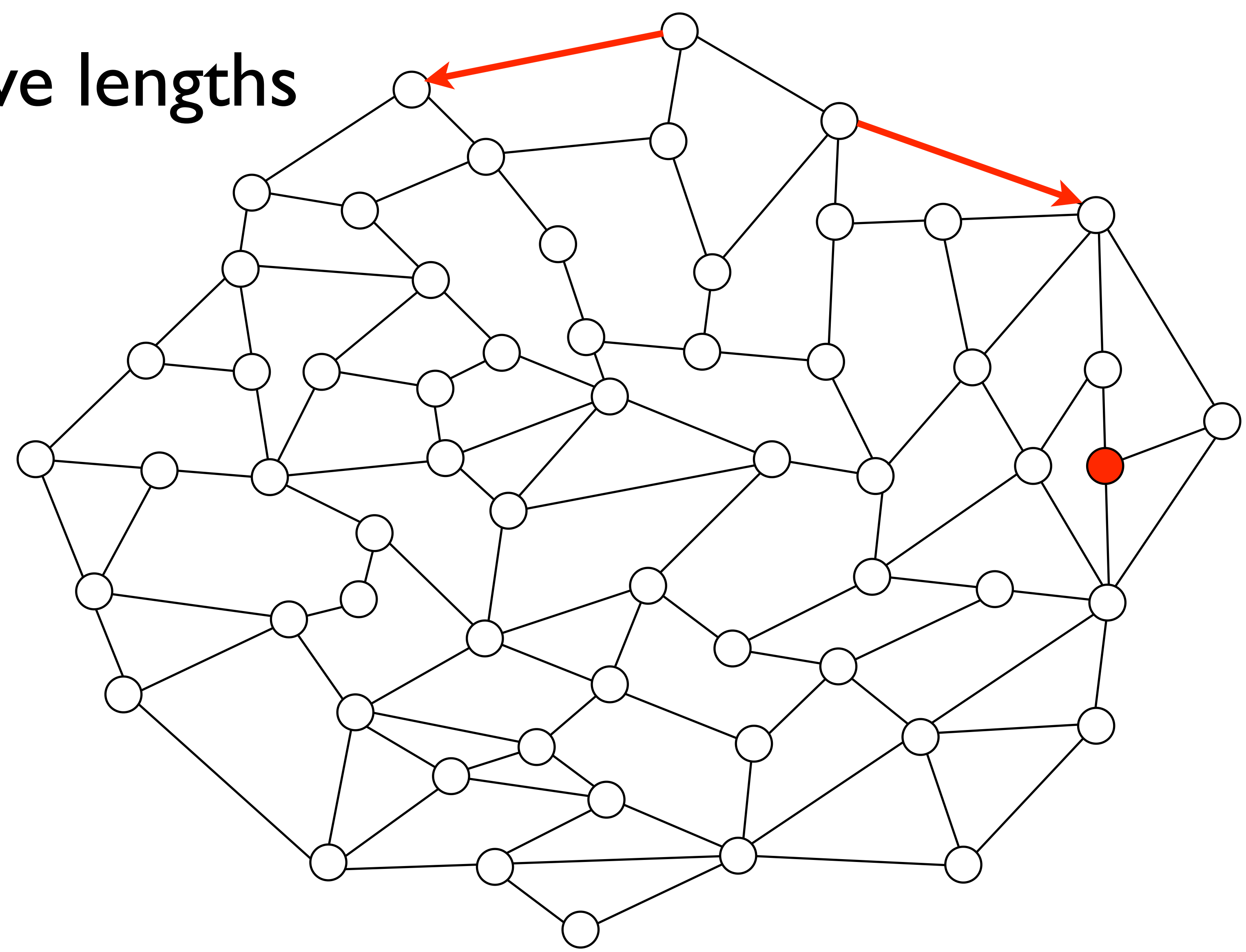
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles



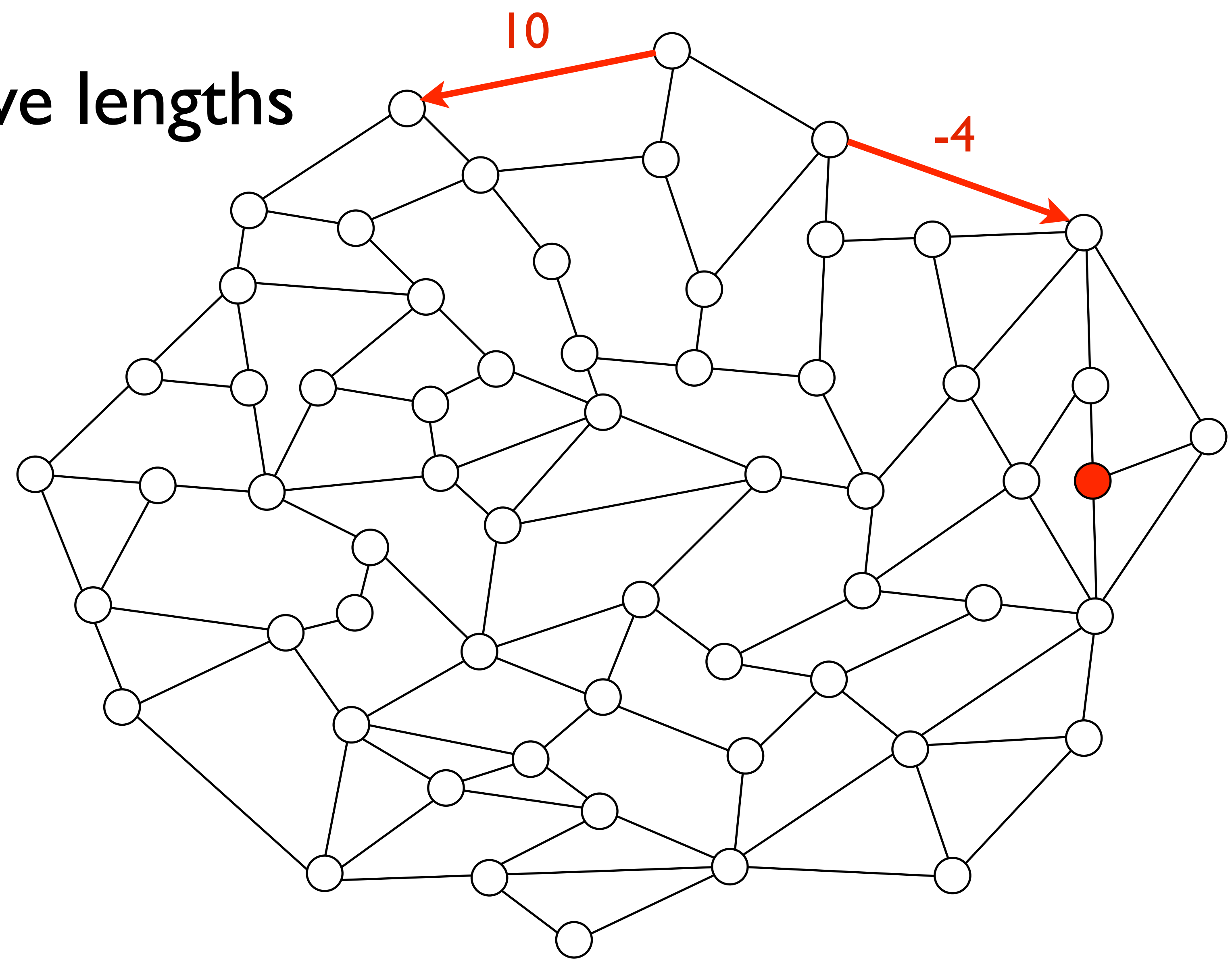
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles



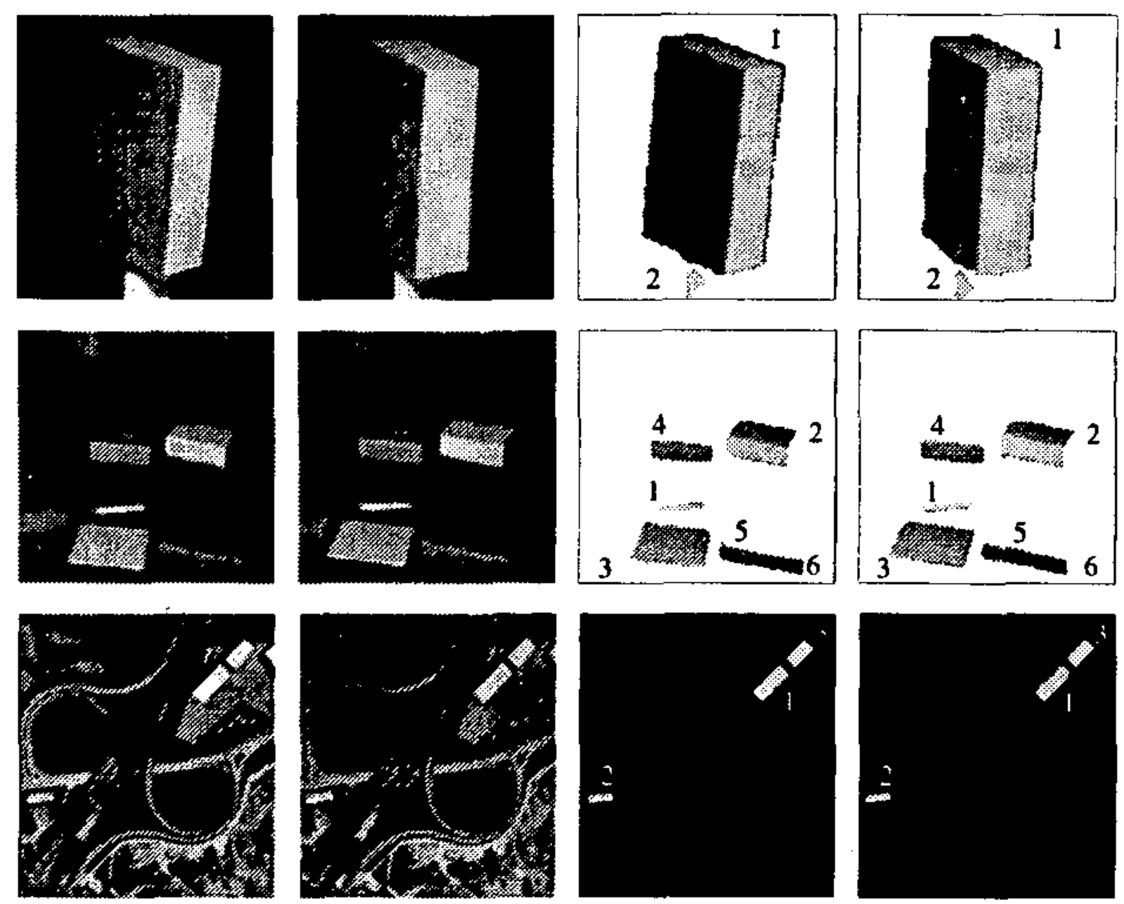
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles



Applications

- Feasible circulation
- Feasible flow
- Perfect matching
- Image segmentation
- Stereo matching



Related Work

General graphs:

- Dijkstra (non-negative lengths) - $O\,(n \log n + m)$
- Bellman-Ford - $O(nm)$

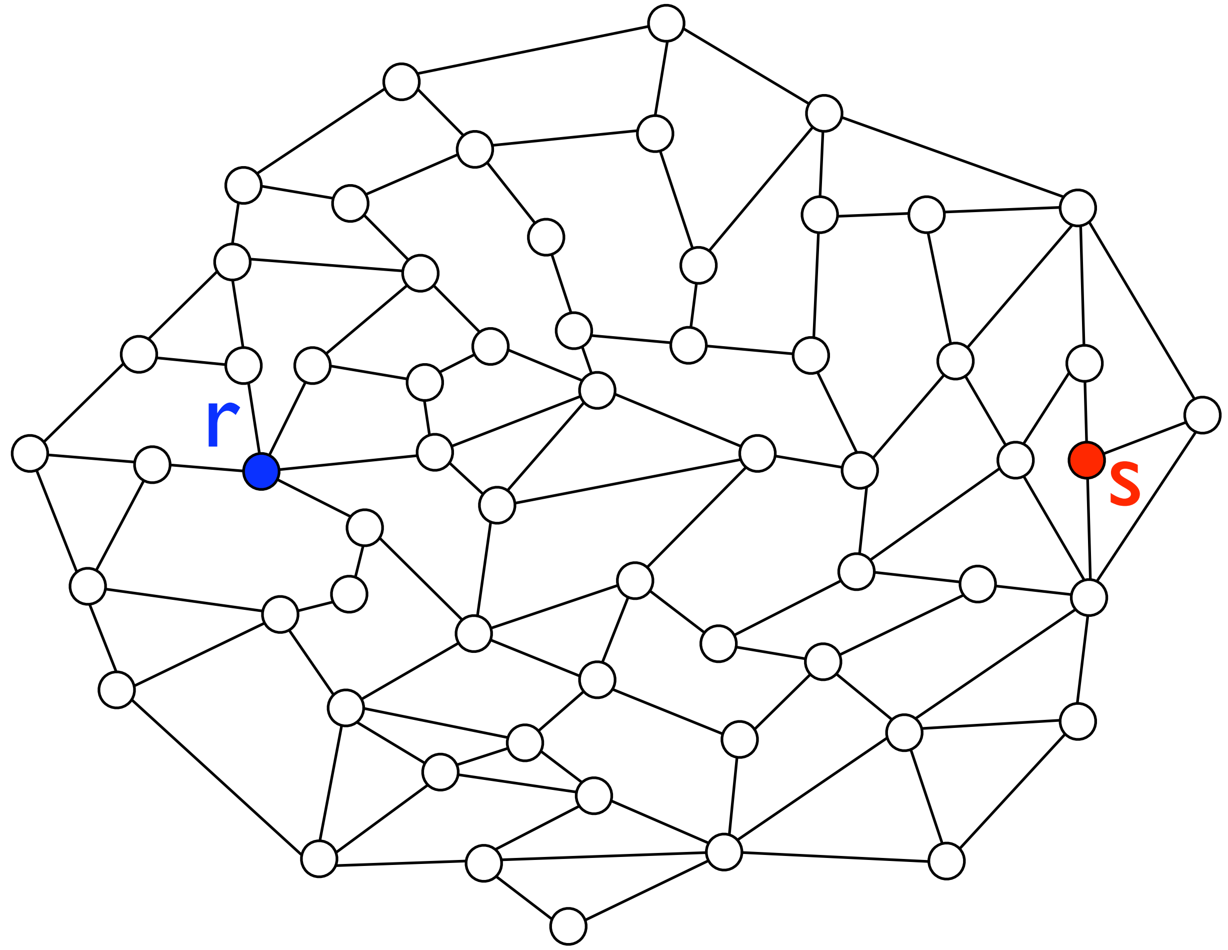
Planar graphs:

- $O(n^{3/2})$ - [Lipton, Rose and Tarjan 1979]
- $O(n^{4/3} \log^{2/3} D)$ - [Henzinger, Klein, Rao, Subramanian 1994]
  also, $O(n)$ for non-negative lengths
- $O(n \log^3 n)$ time $O(n \log n)$ space - [Fakcharoenphol and Rao 2001]

Our Contribution:

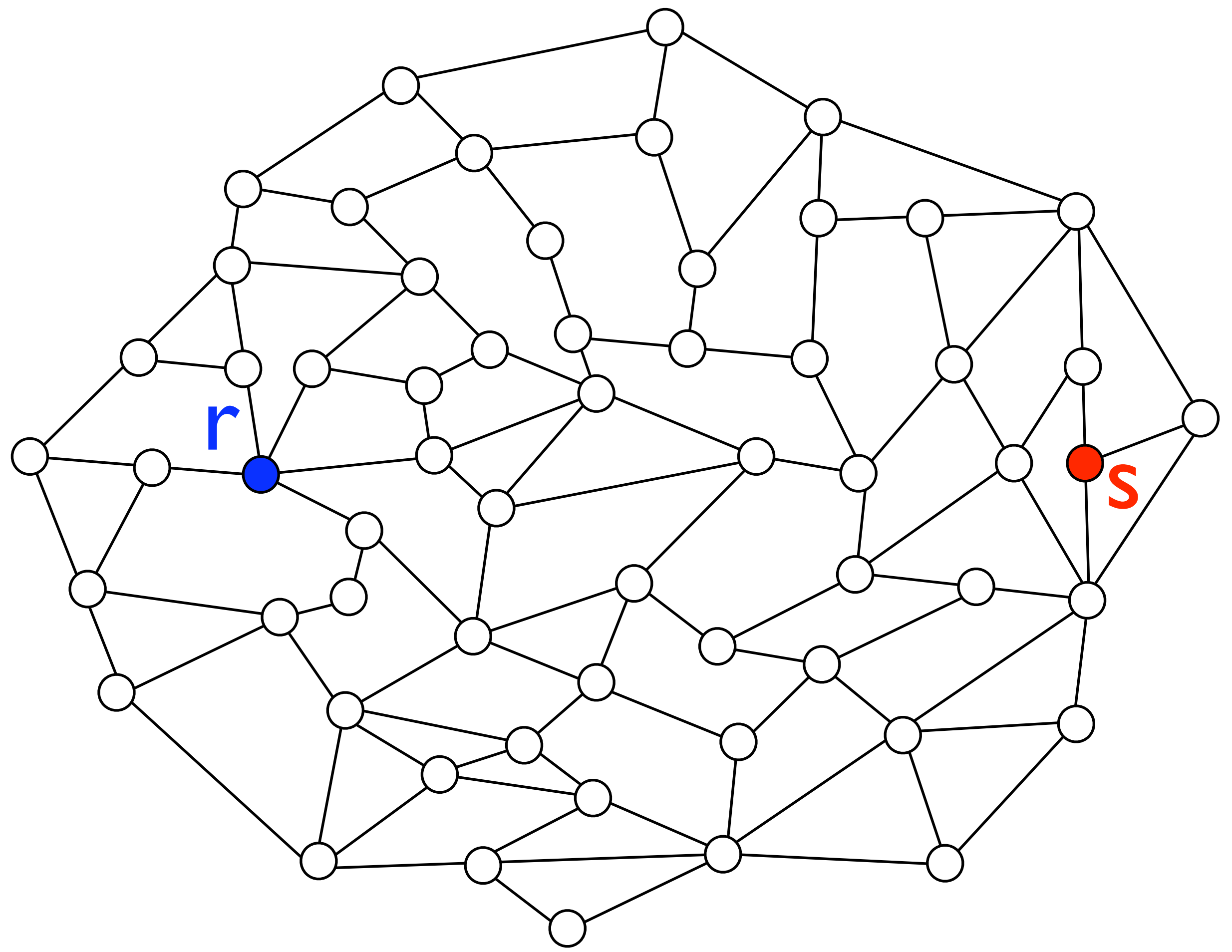
- $O(n \log^2 n)$ time, $O(n)$ space

Rerooting



Rerooting

- We want distances from s
- It suffices to find distances from any node r

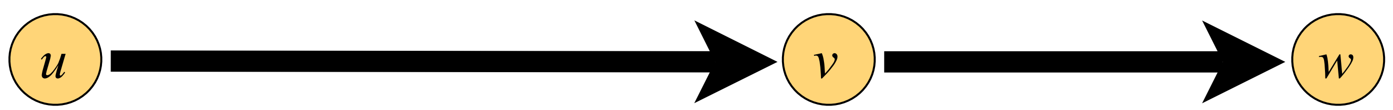


Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$

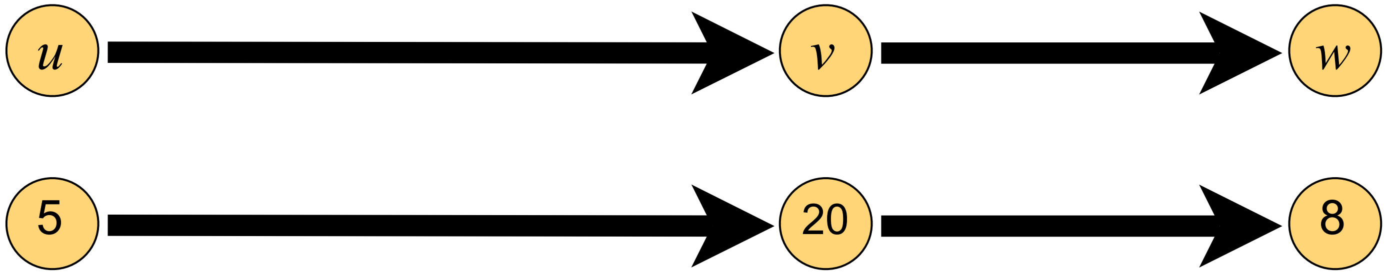
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



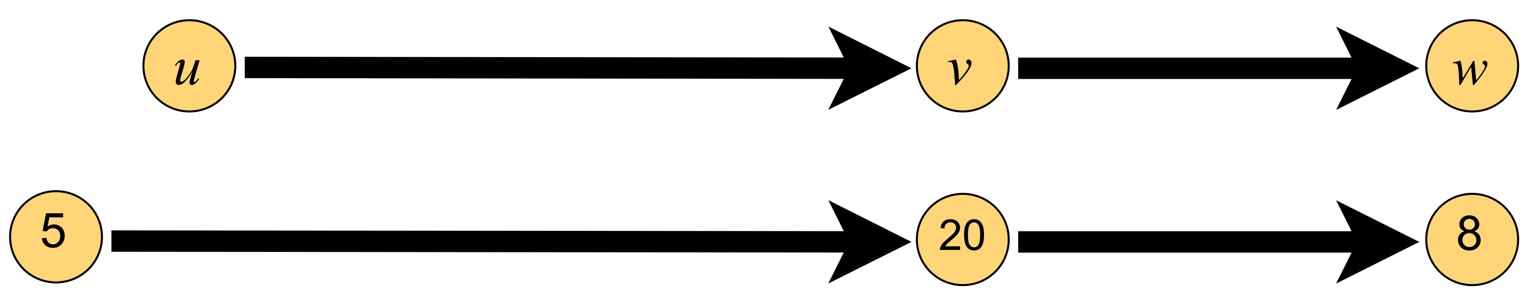
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



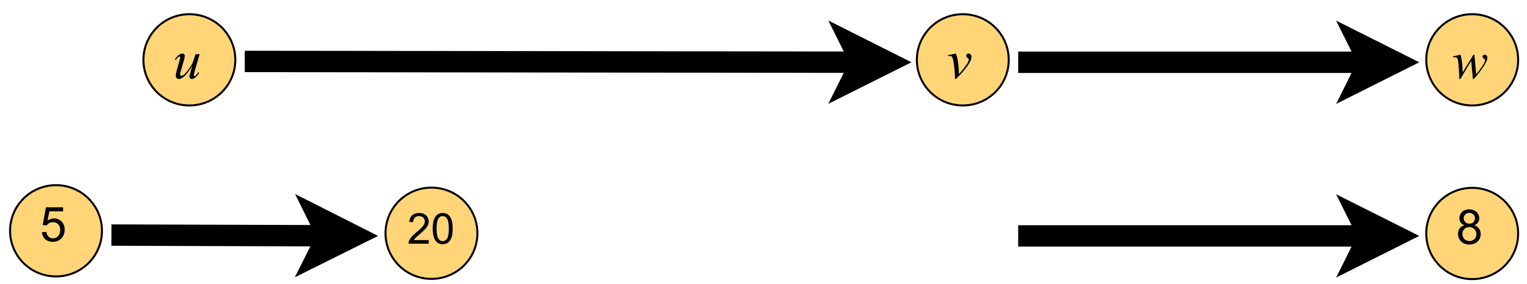
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



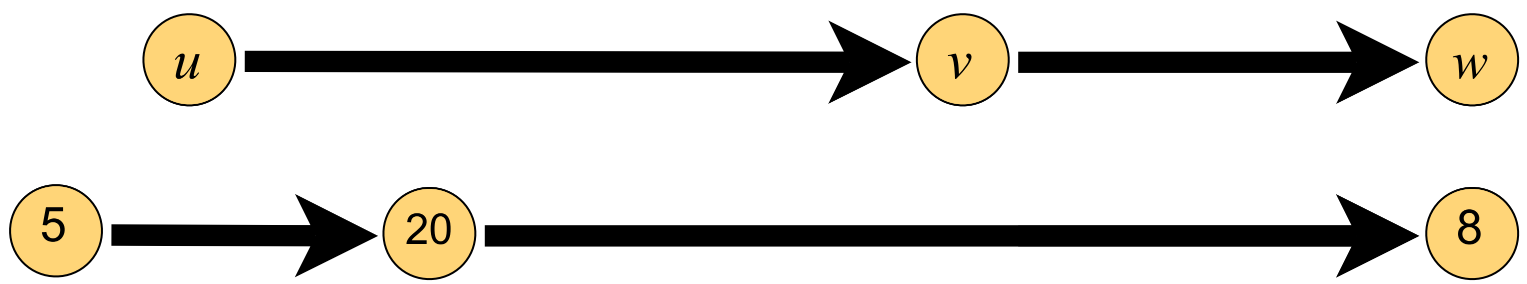
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



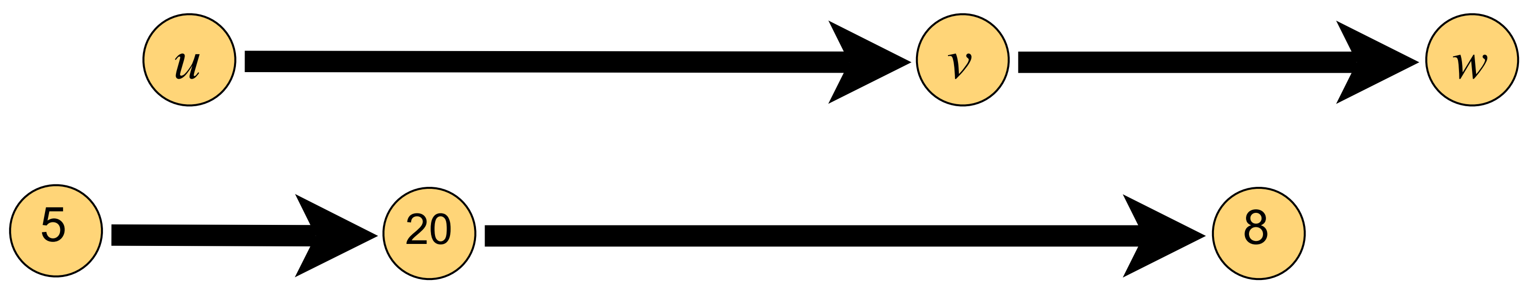
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



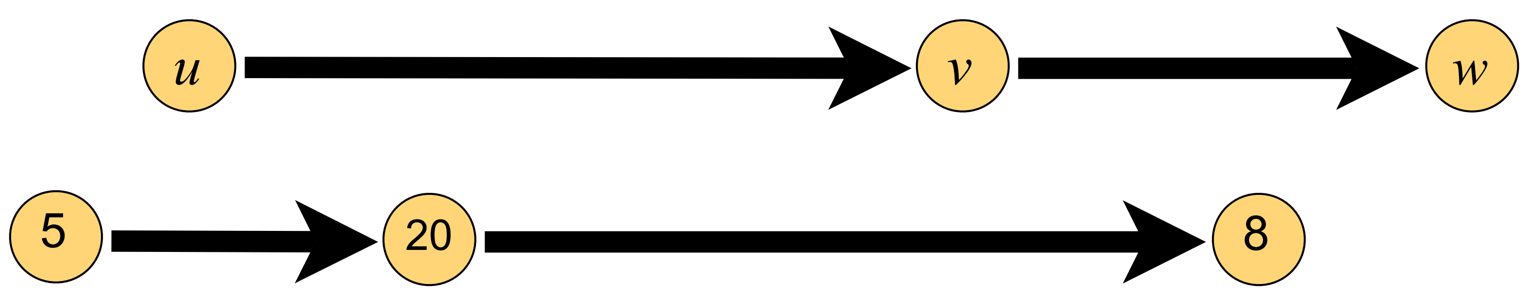
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



Price Functions, Reduced Lengths

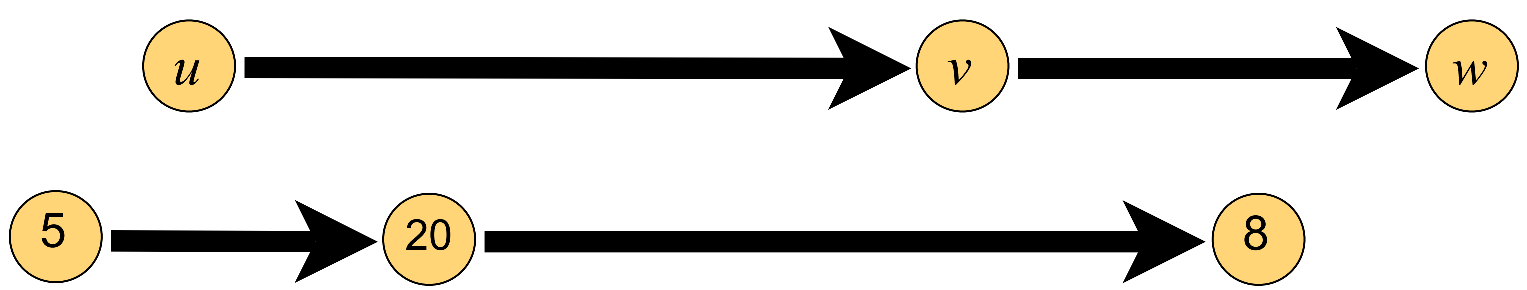
- **Price function**: $\varphi(v)$
- **Reduced length**: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$

Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$



- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$
- $\varphi(v)$ preserves shortest paths

Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$
- $\varphi(v)$ preserves shortest paths

Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$
- $\varphi(v)$ preserves shortest paths

Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$
- $\varphi(v)$ preserves shortest paths

- Price function is feasible if l_φ is non-negative
- Converts a problem with negative lengths to non-negative lengths

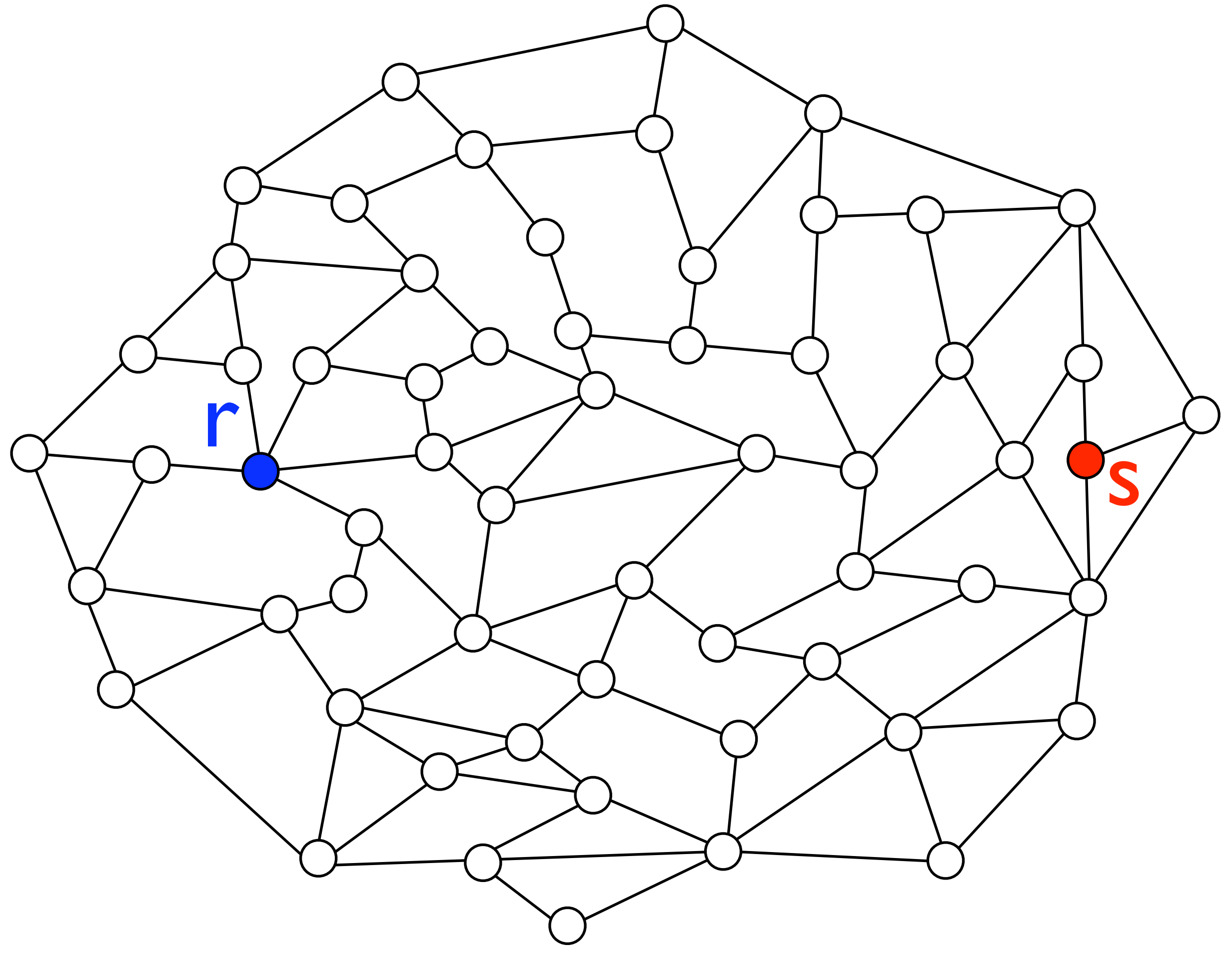
Price Functions, Reduced Lengths

- Price function: $\varphi(v)$
- Reduced length: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
- Length of any s-t path changes by $\varphi(s)$-$\varphi(t)$
- $\varphi(v)$ preserves shortest paths

- Price function is feasible if l_φ is non-negative
- Converts a problem with negative lengths to non-negative lengths
- Single source distances form a feasible price function because $\varphi(u) + l(uv) \geq \varphi(v)$

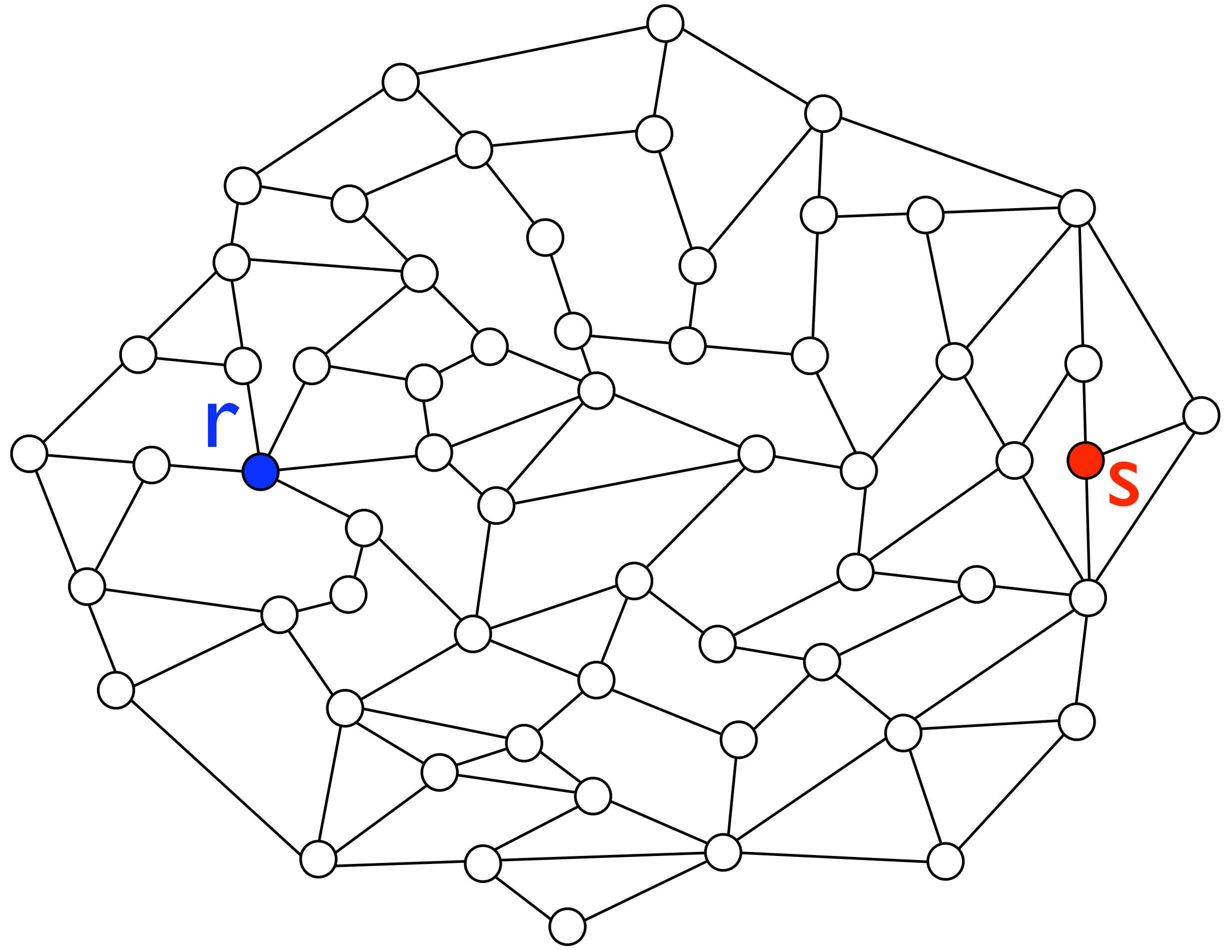
Rerooting

- We want distances from s
- It suffices to find distances from any node r



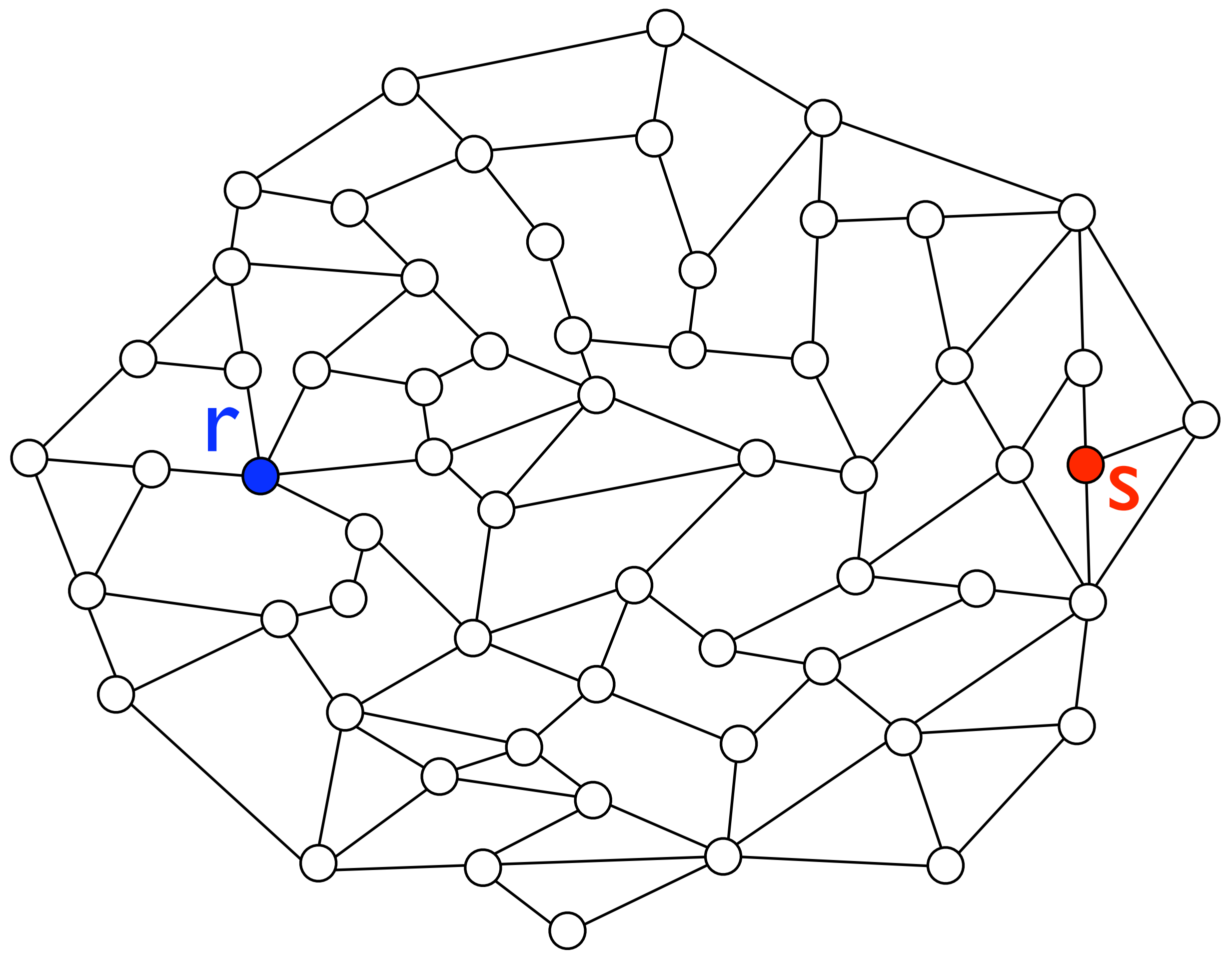
Rerooting

- We want distances from s
- It suffices to find distances from any node r
  - Use distances from r as a feasible price function



Rerooting

- We want distances from s
- It suffices to find distances from any node r
  - Use distances from r as a feasible price function
  - Run Dijkstra's algorithm from s



High-Level View

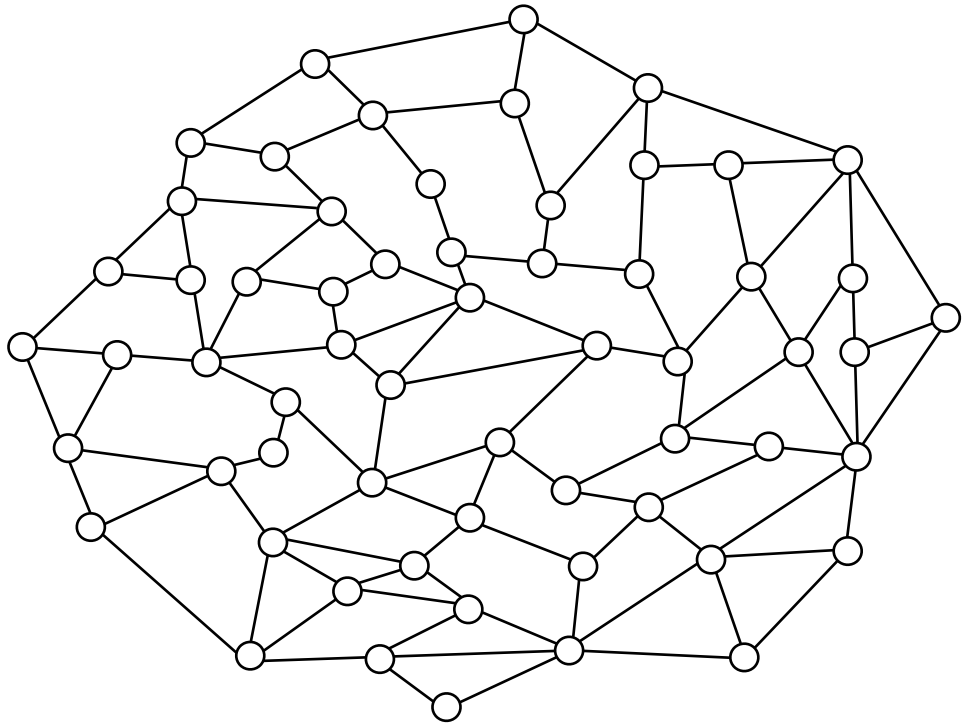
I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

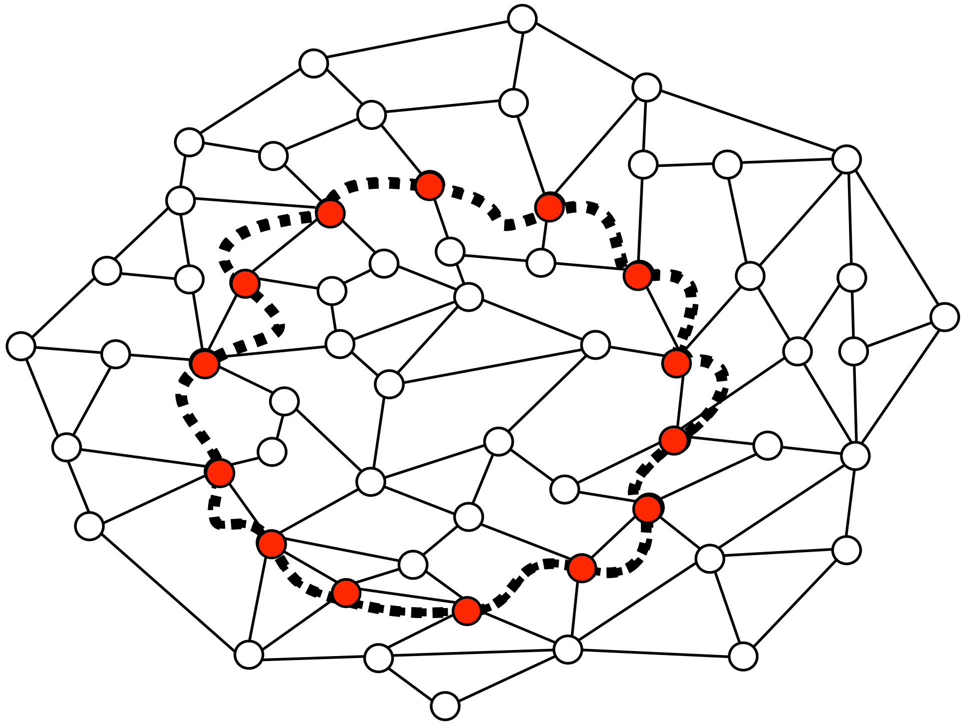
IV. distances from r in G

V. rerooting



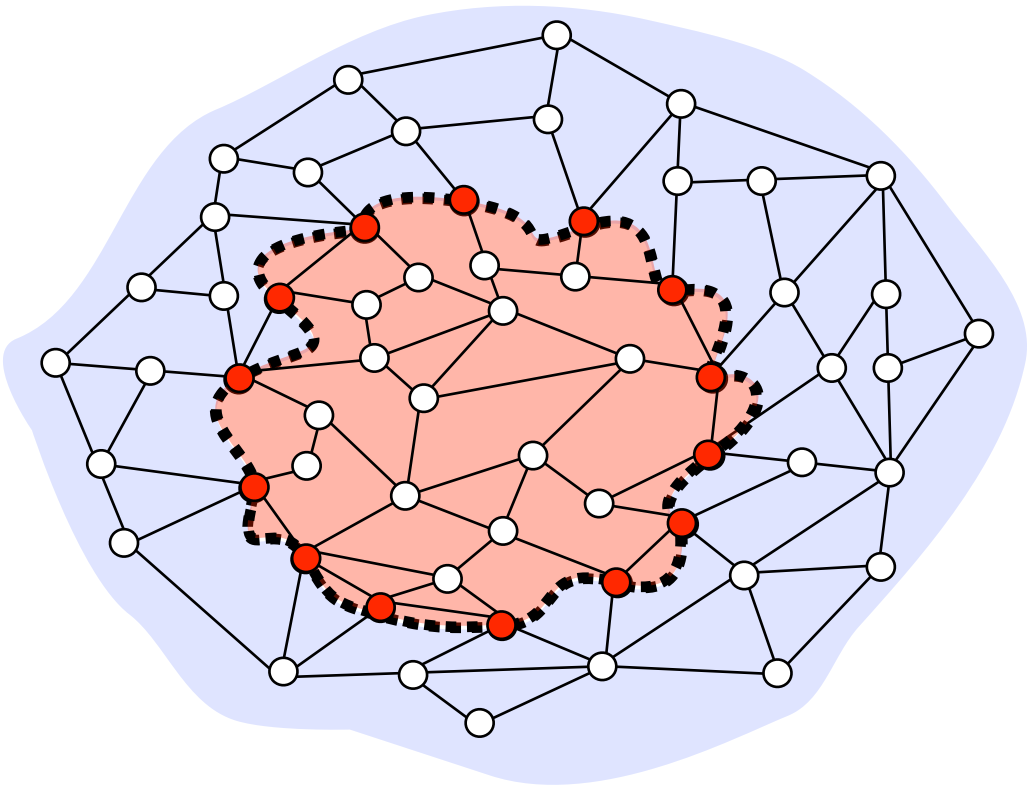
High-Level View

I. recursion
II. boundary to boundary distances in G_i
III. r-to-boundary distances in G
IV. distances from r in G
V. rerooting



High-Level View

I. recursion
II. boundary to boundary distances in G_i
III. r-to-boundary distances in G
IV. distances from r in G
V. rerooting



High-Level View

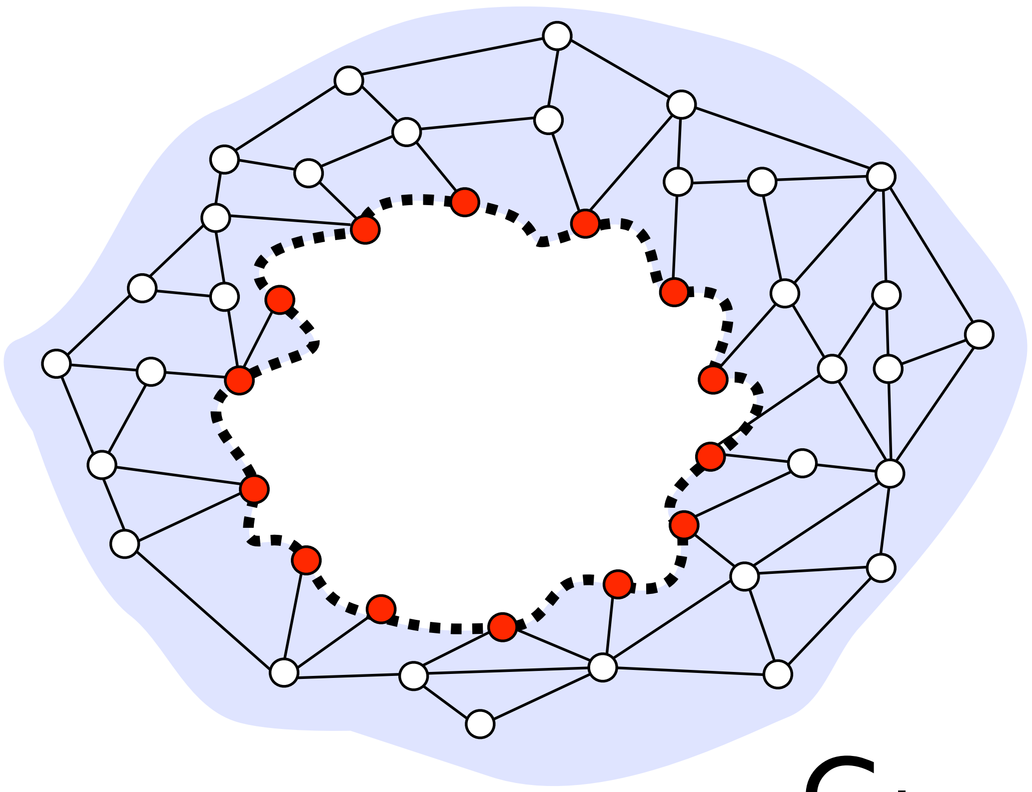
I. recursion

II. boundary to boundary distances in G_i

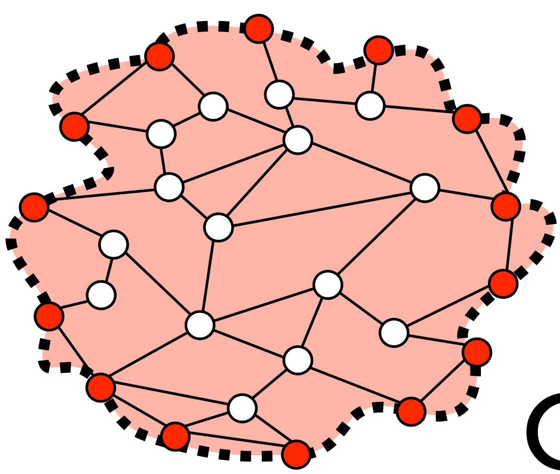
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



G_1



G_0

High-Level View

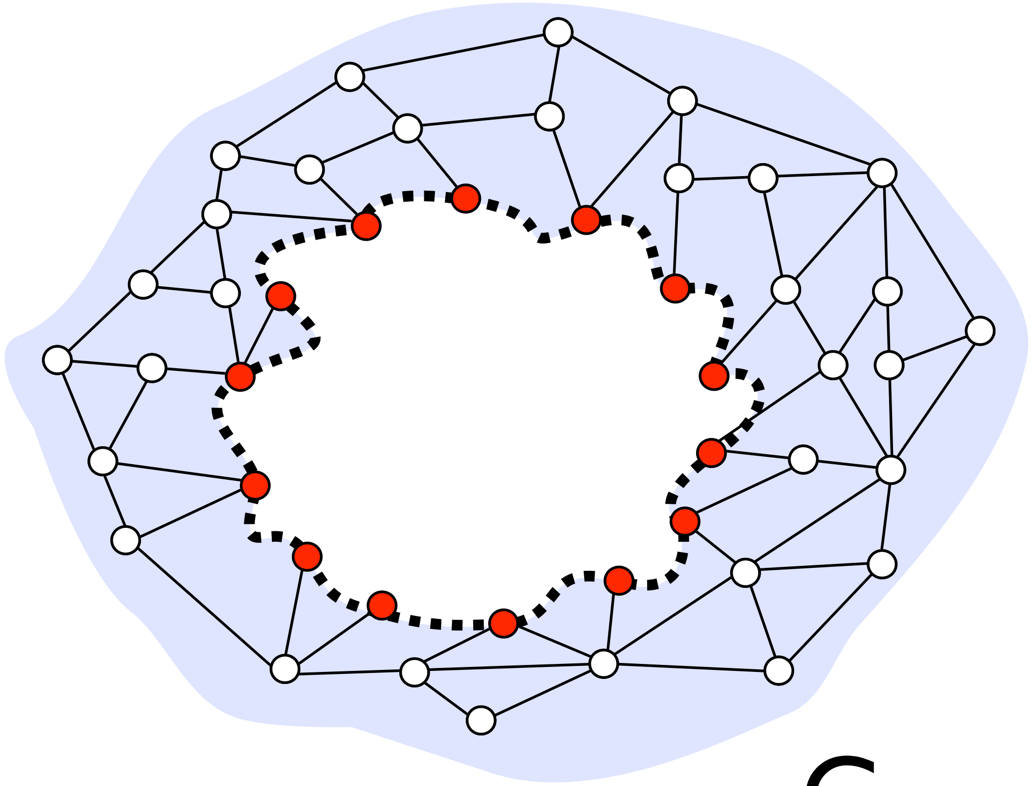
I. recursion

II. boundary to boundary distances in G_i

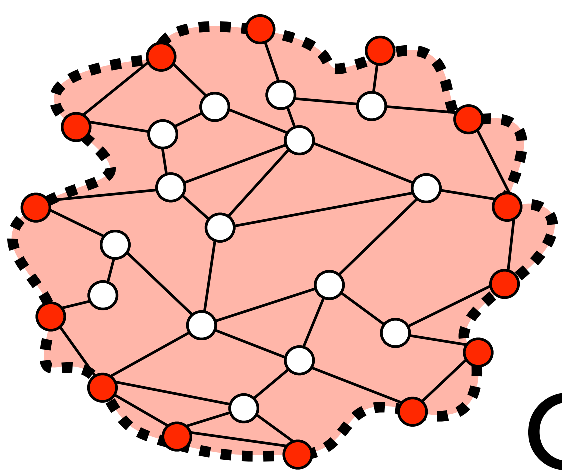
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



G_1



G_0

High-Level View

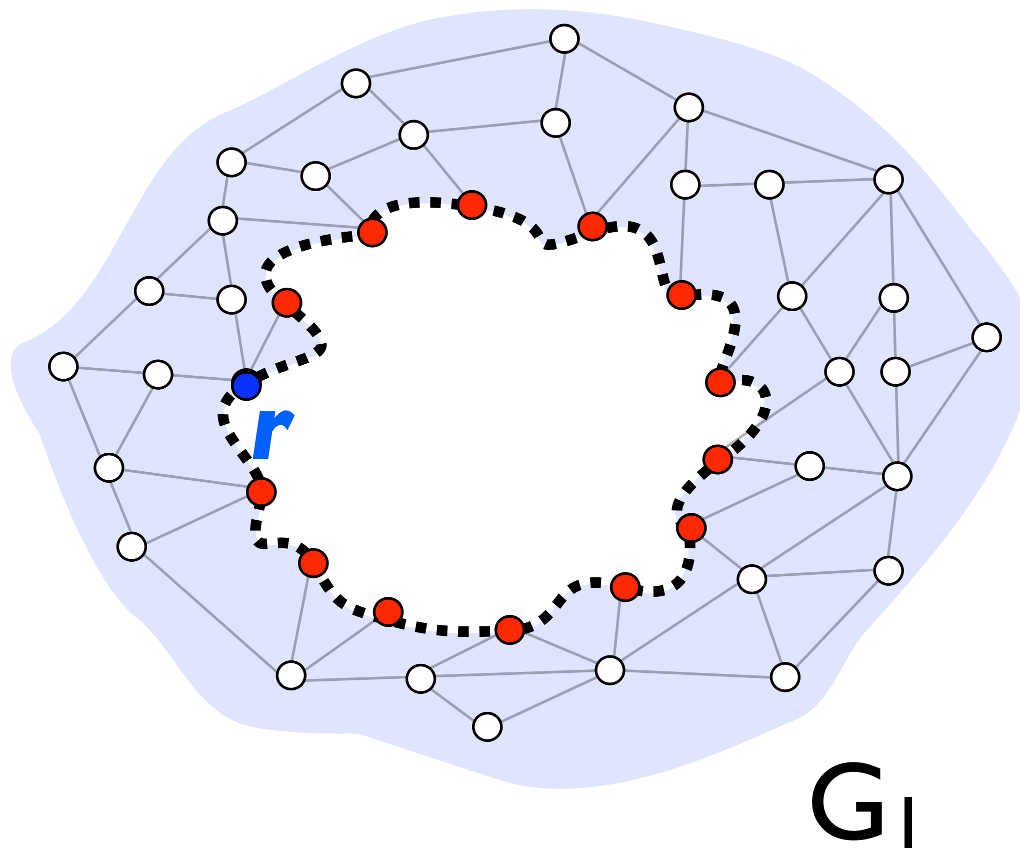
I. recursion

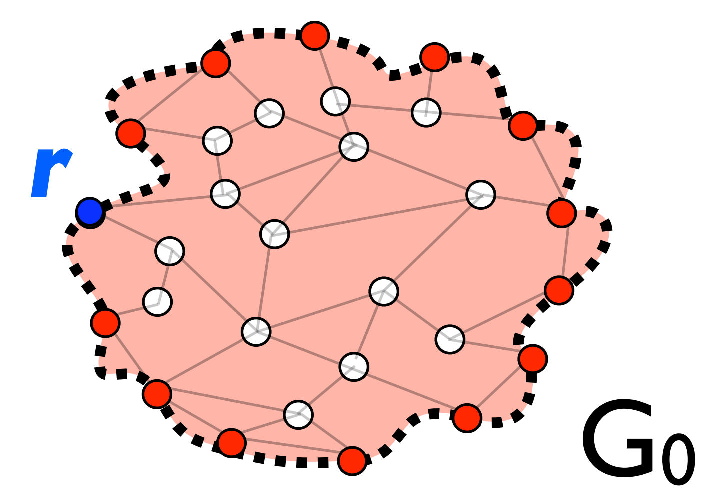
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

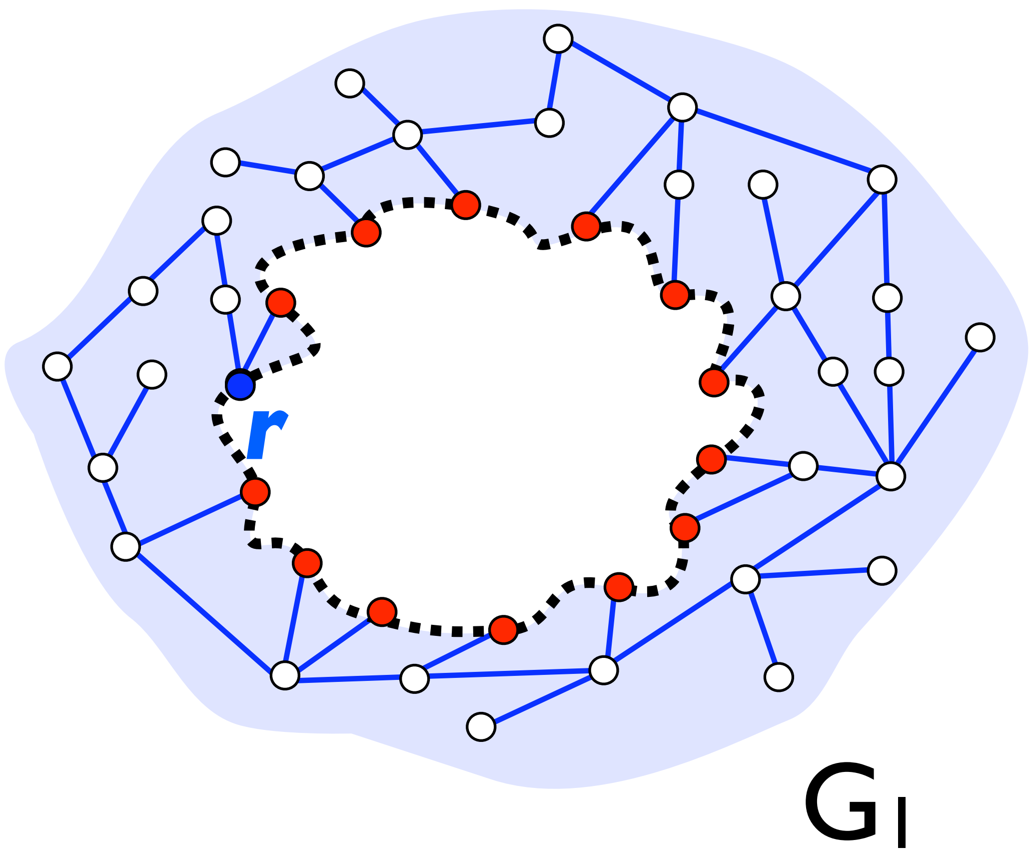
I. recursion

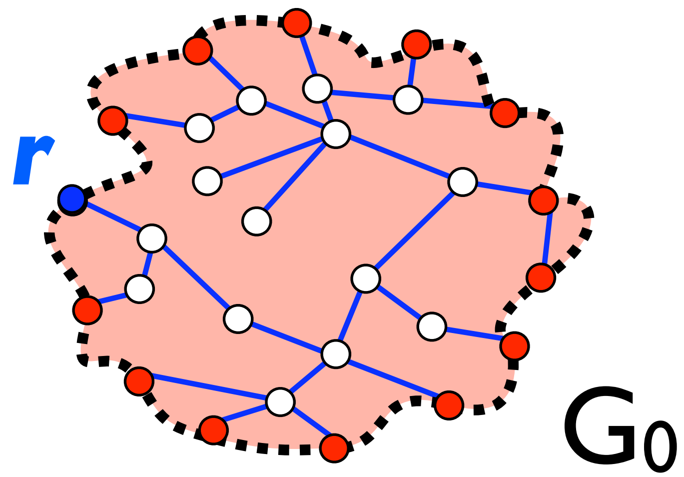
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

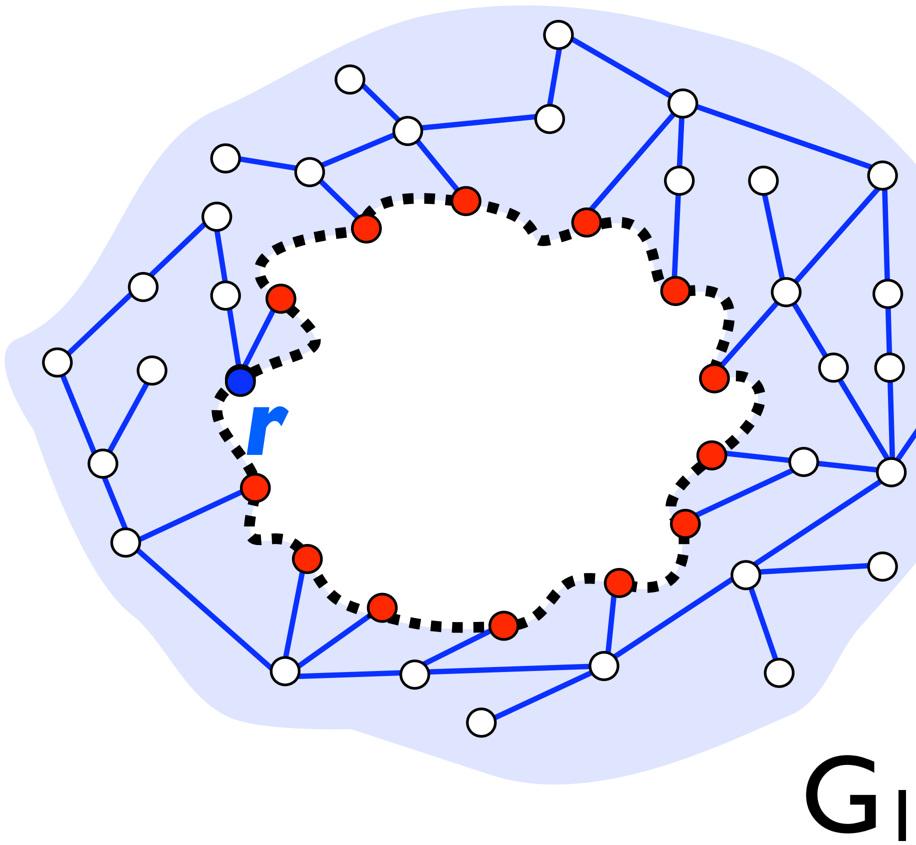
I. recursion

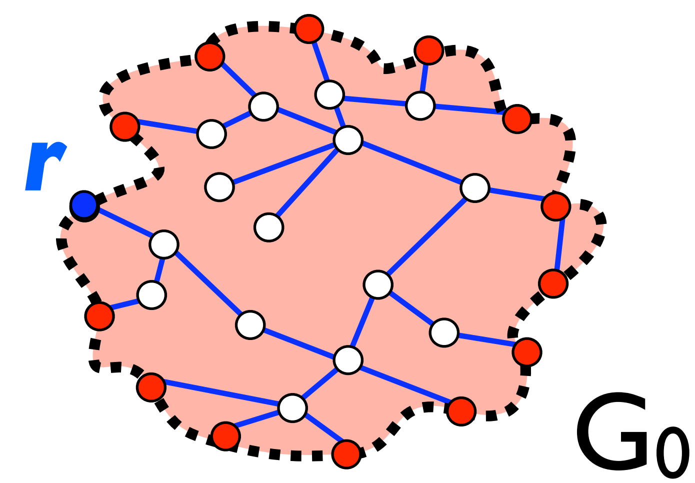
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

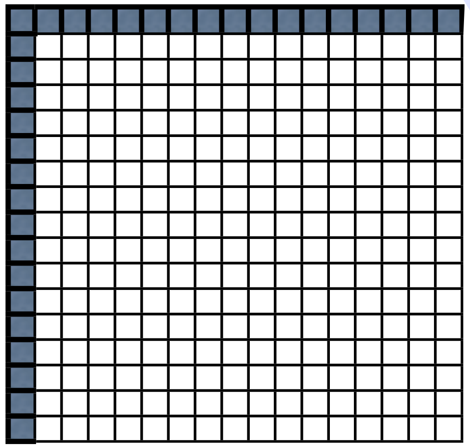
I. recursion

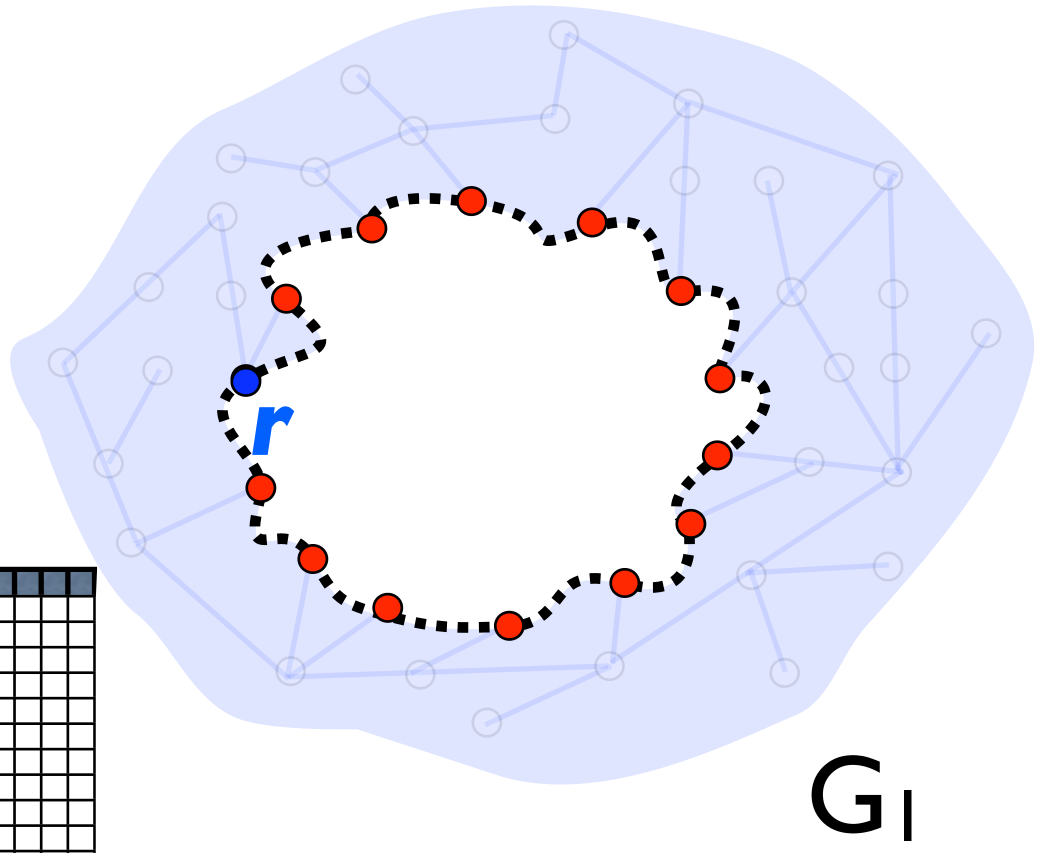
II. boundary to boundary distances in G_i

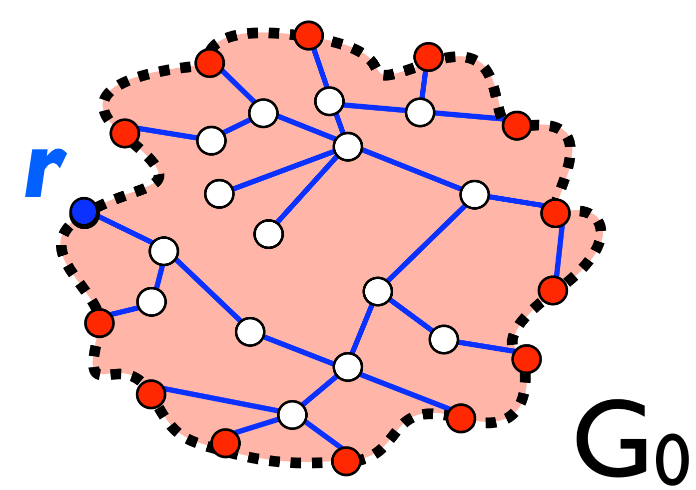
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting







High-Level View

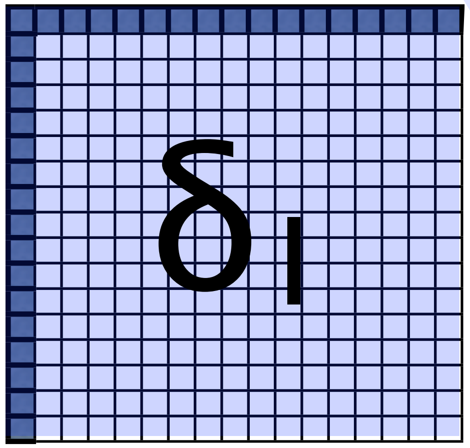
I. recursion

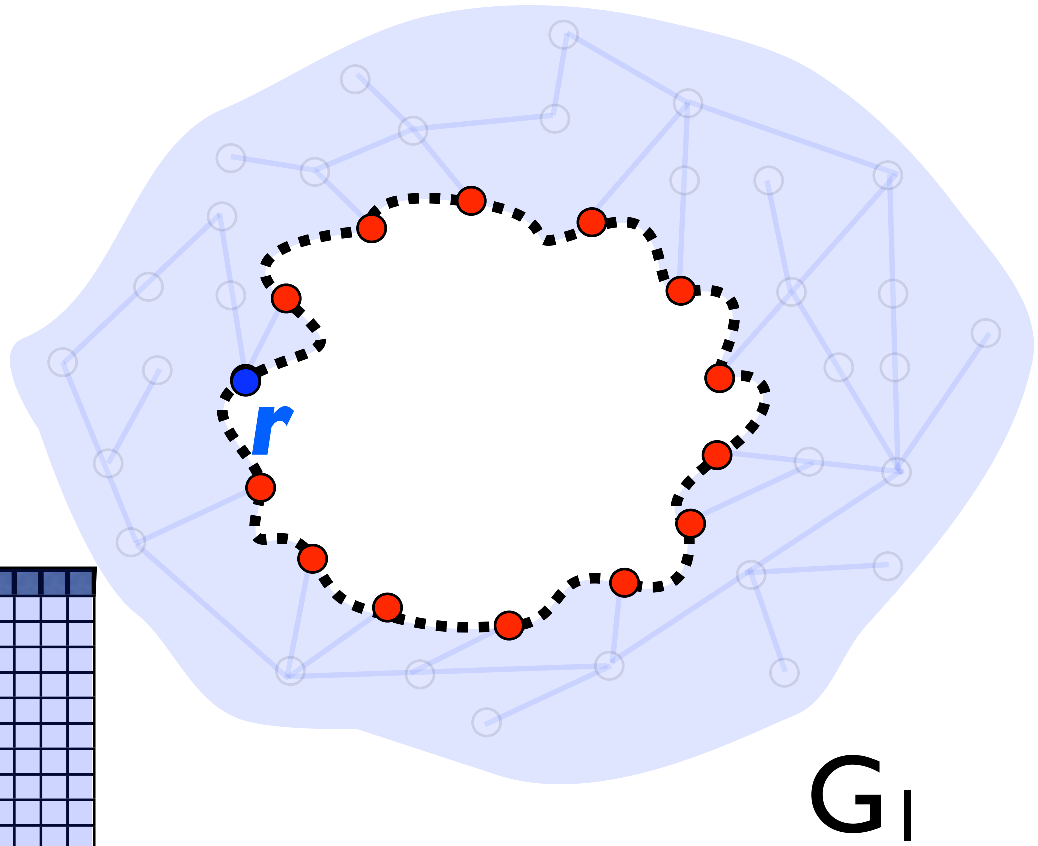
II. boundary to boundary distances in G_i

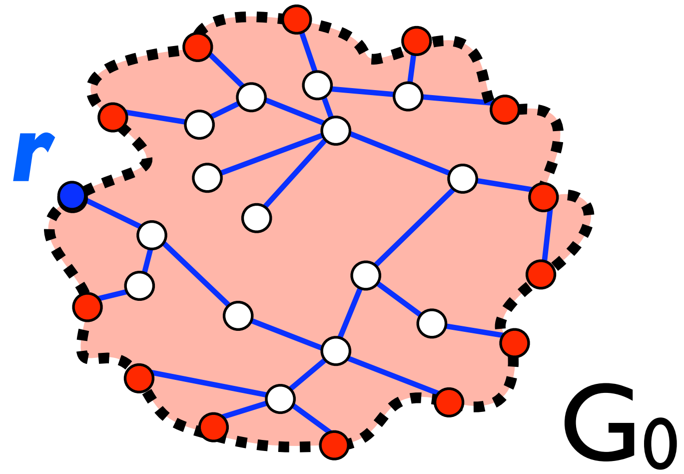
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting







High-Level View

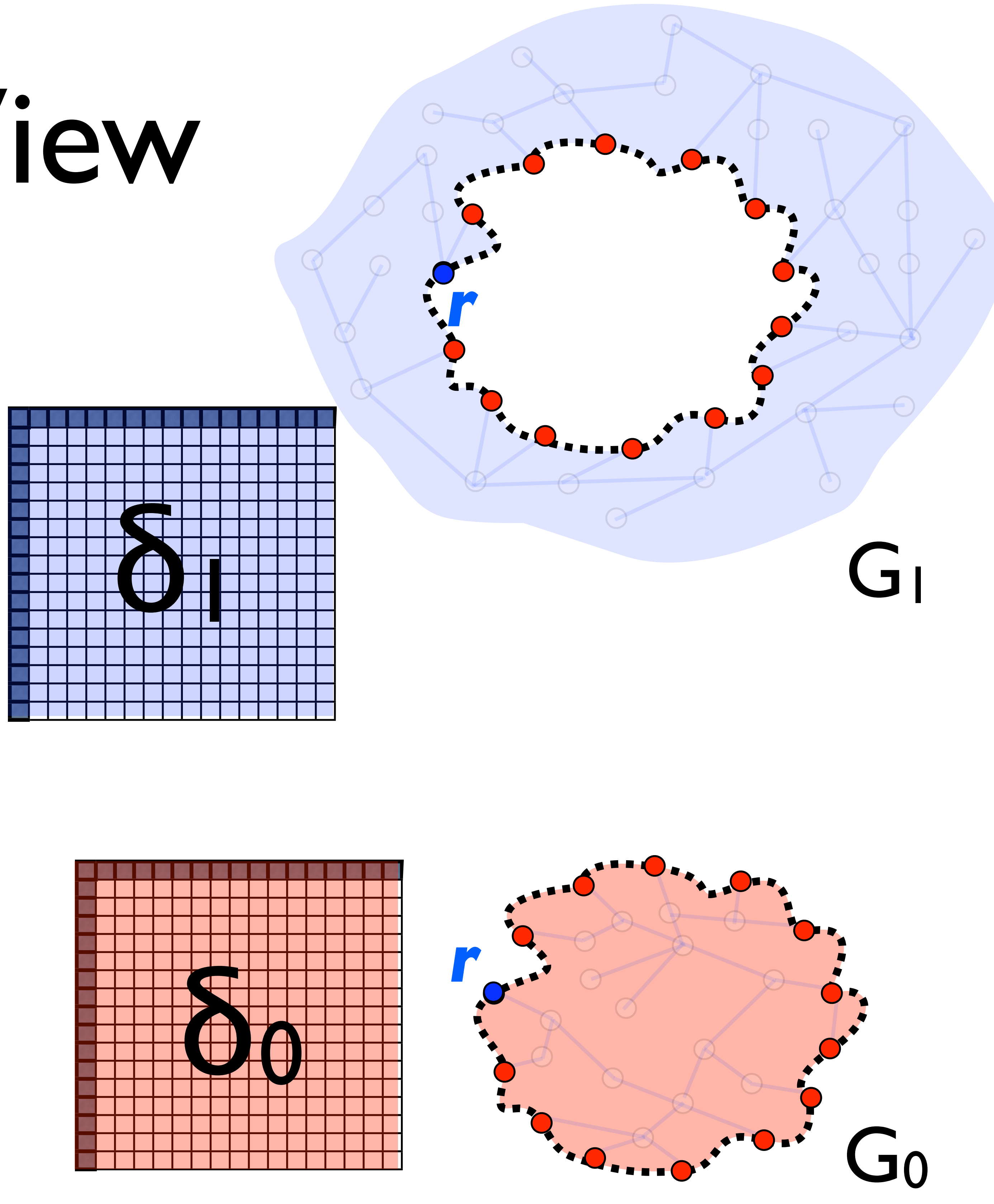
I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



High-Level View

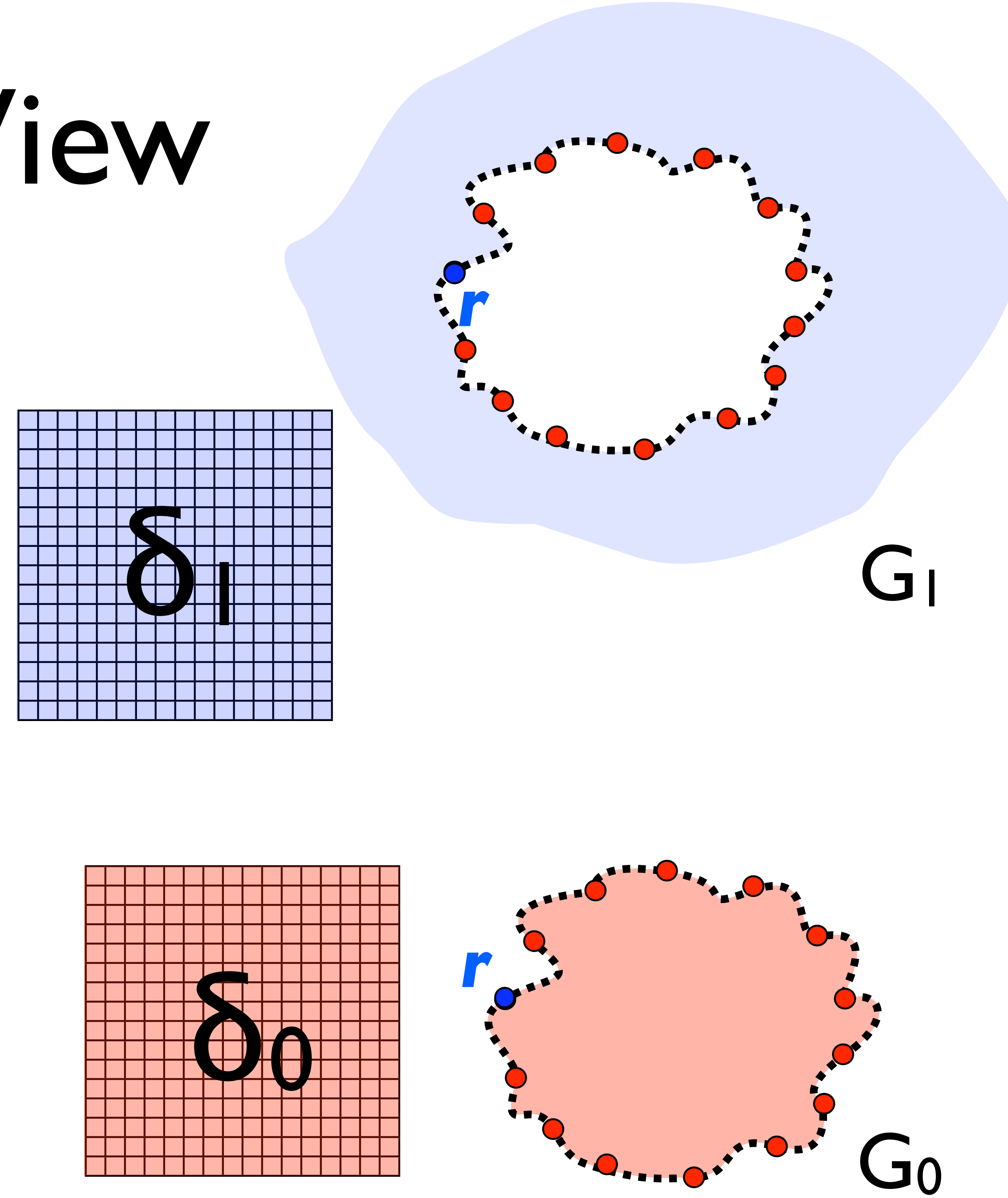
I. recursion

II. boundary to boundary distances in G_i

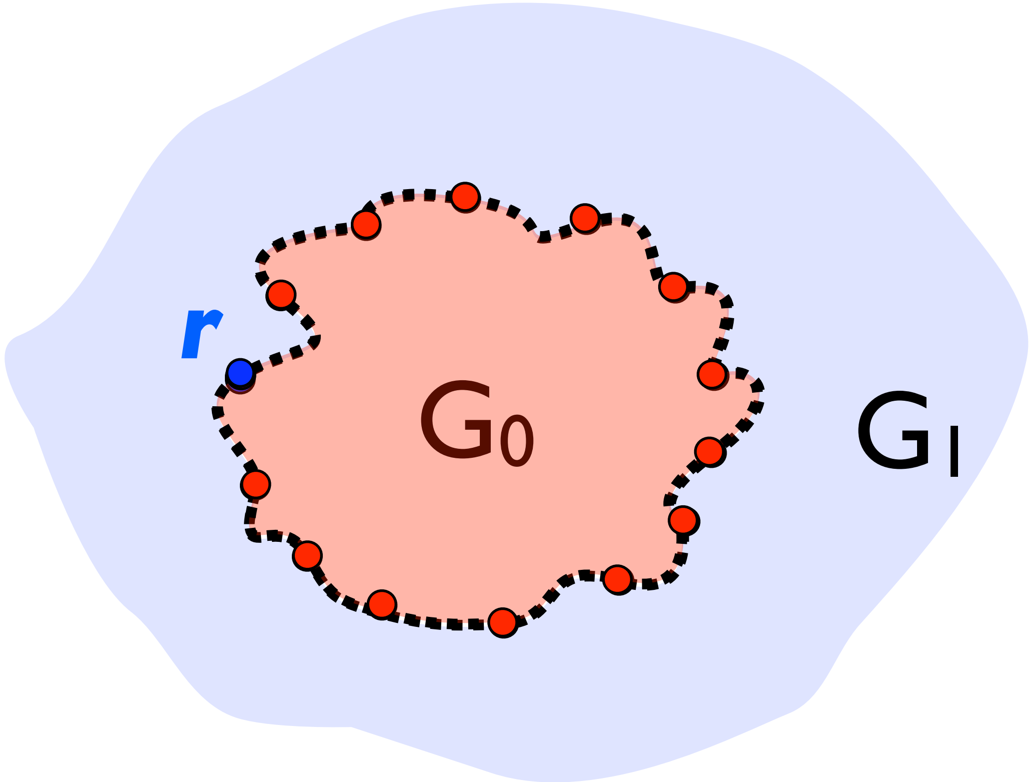
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



High-Level View



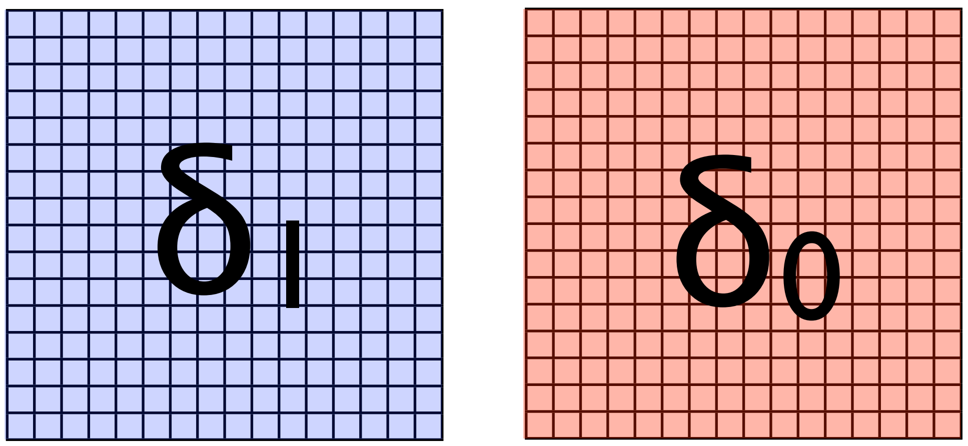
I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



High-Level View

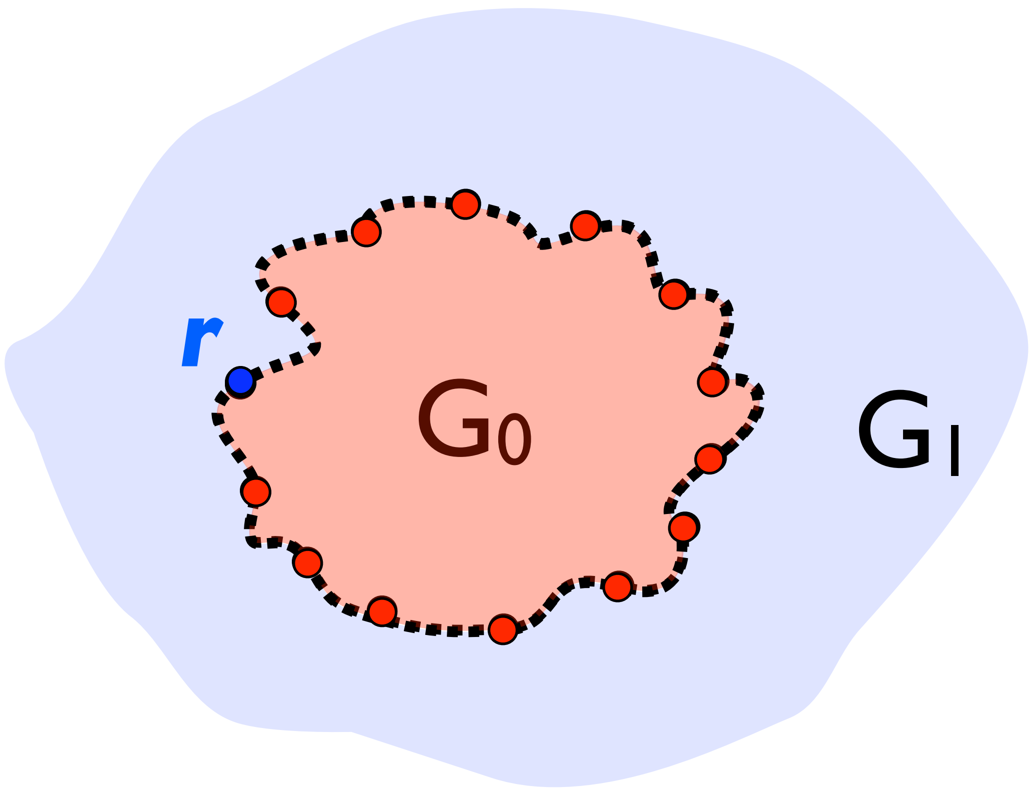
I. recursion

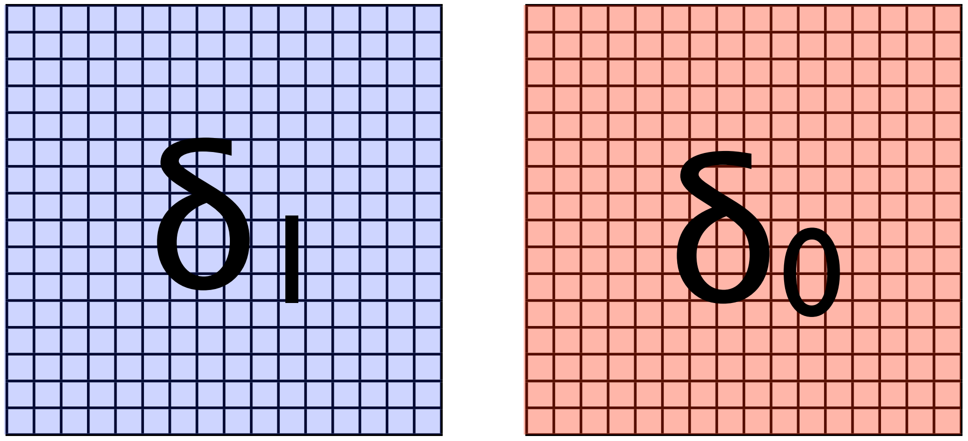
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

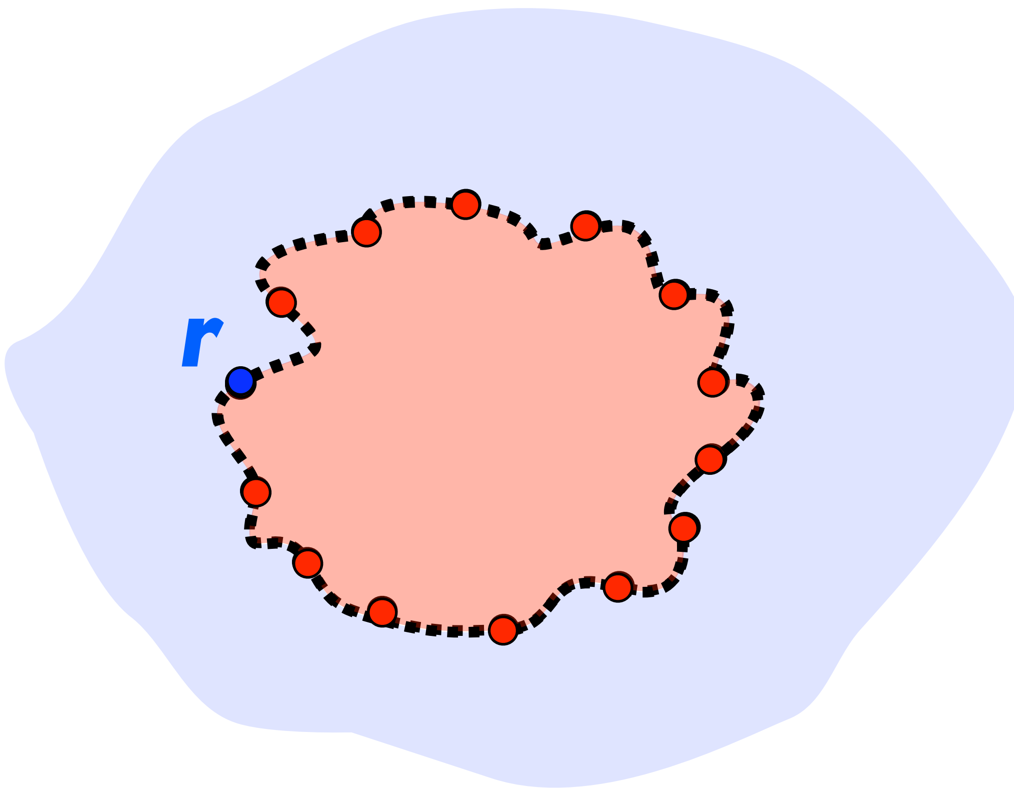
I. recursion

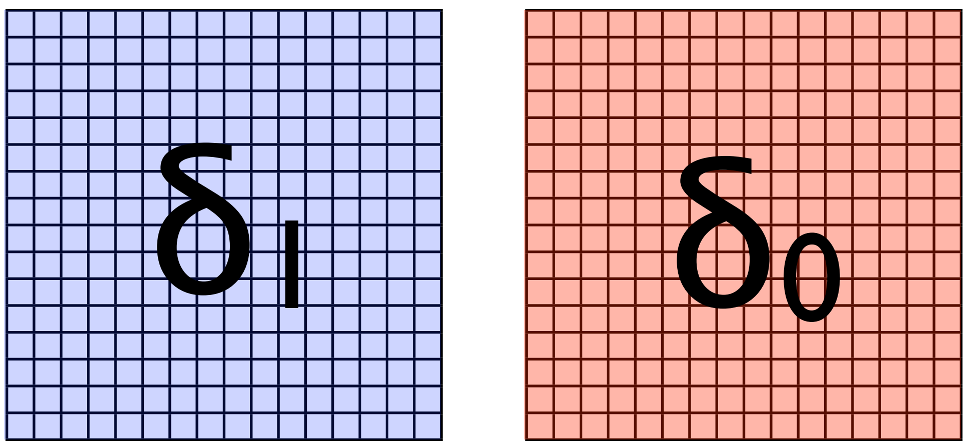
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

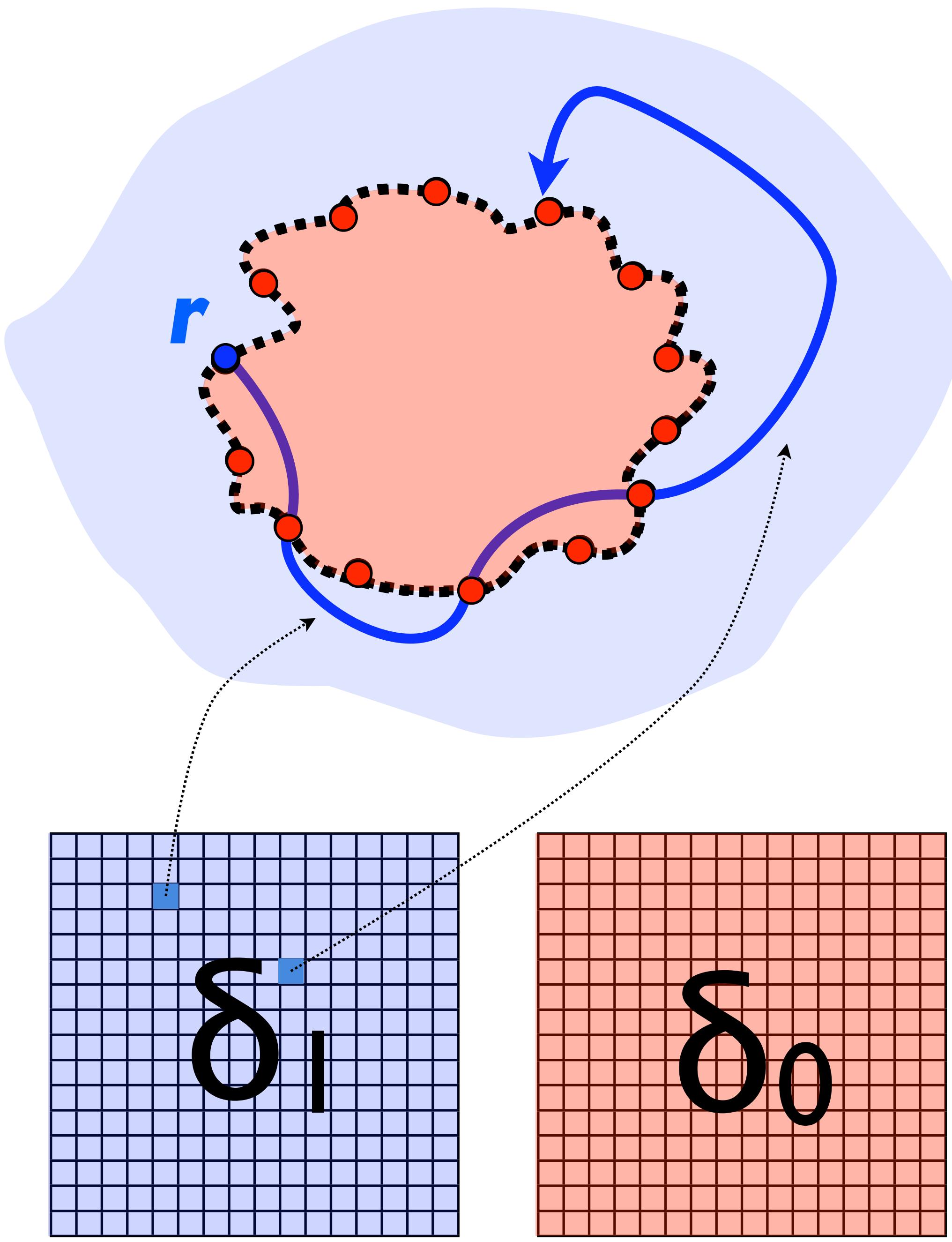
I. recursion

II. boundary to boundary distances in G_i

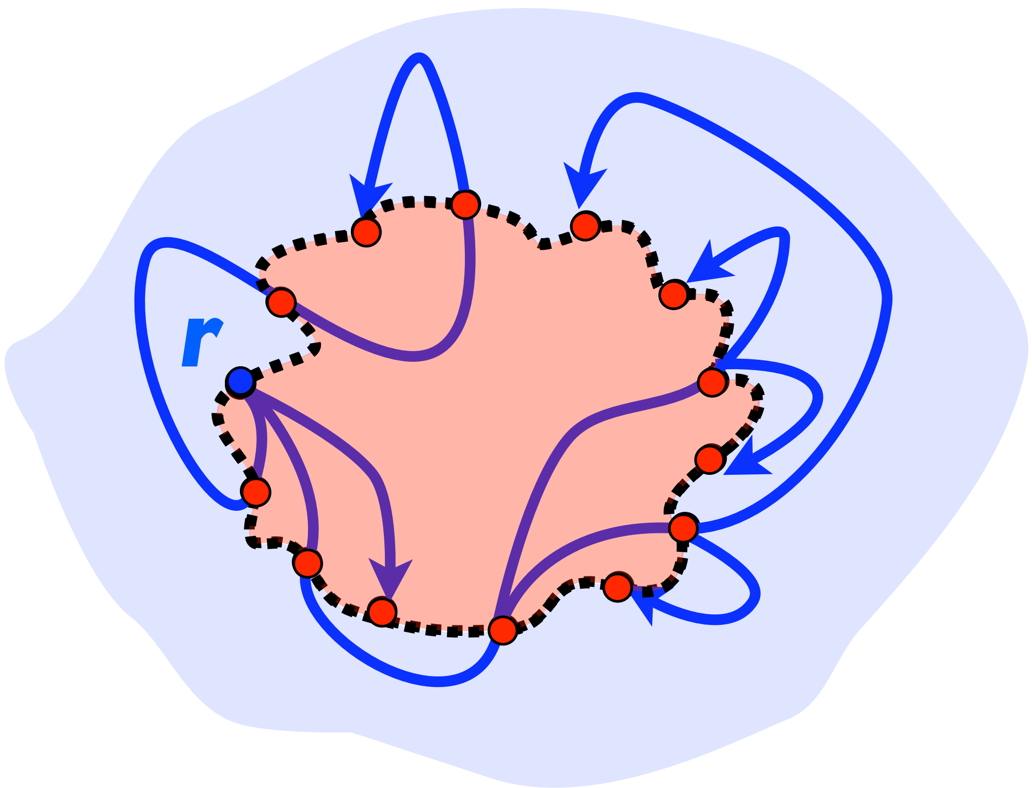
III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting

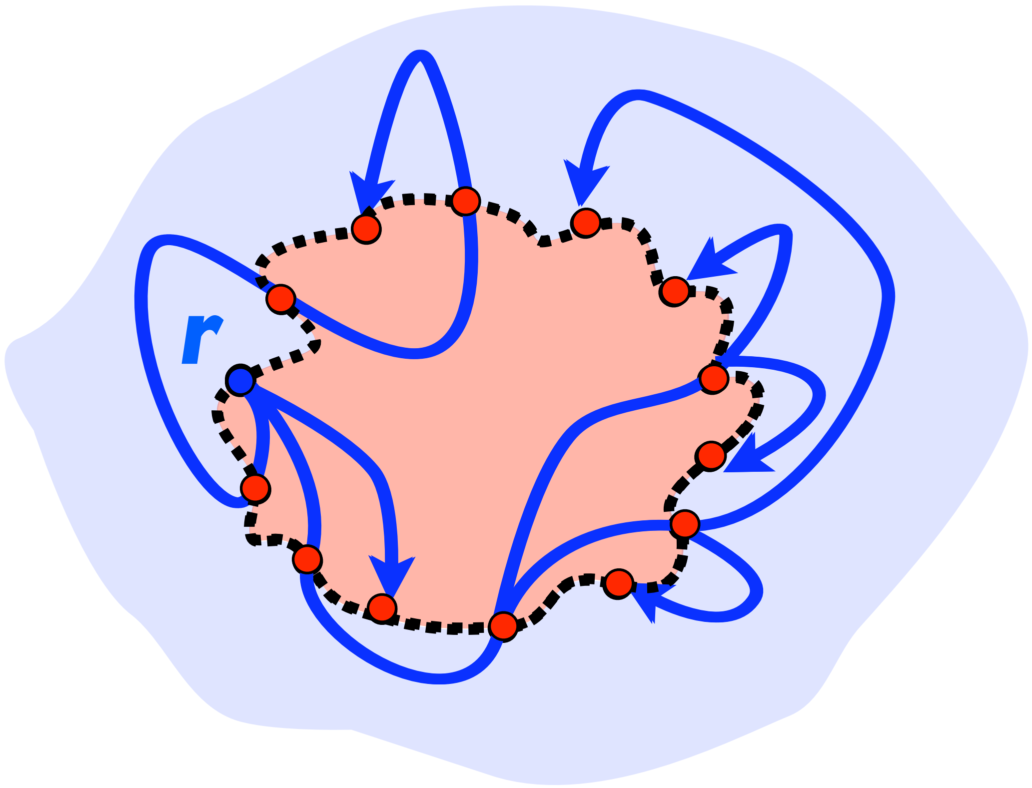


High-Level View



I. recursion
II. boundary to boundary distances in G_i
III. r-to-boundary distances in G
IV. distances from r in G
V. rerooting

High-Level View



I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting

High-Level View

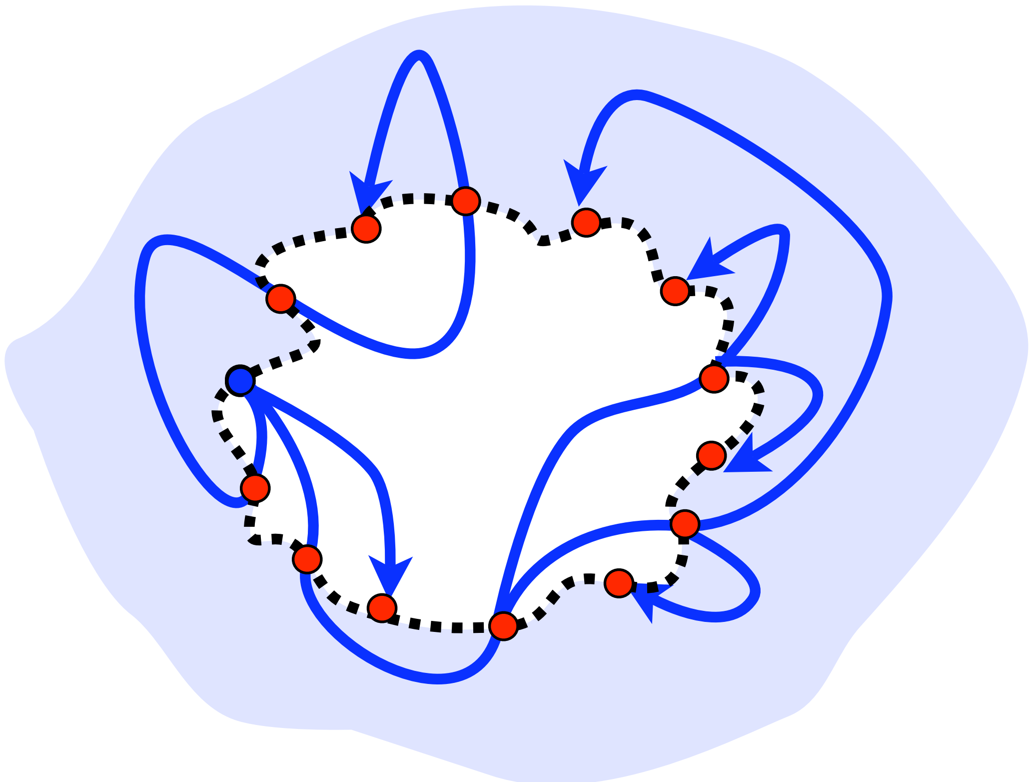
I. recursion

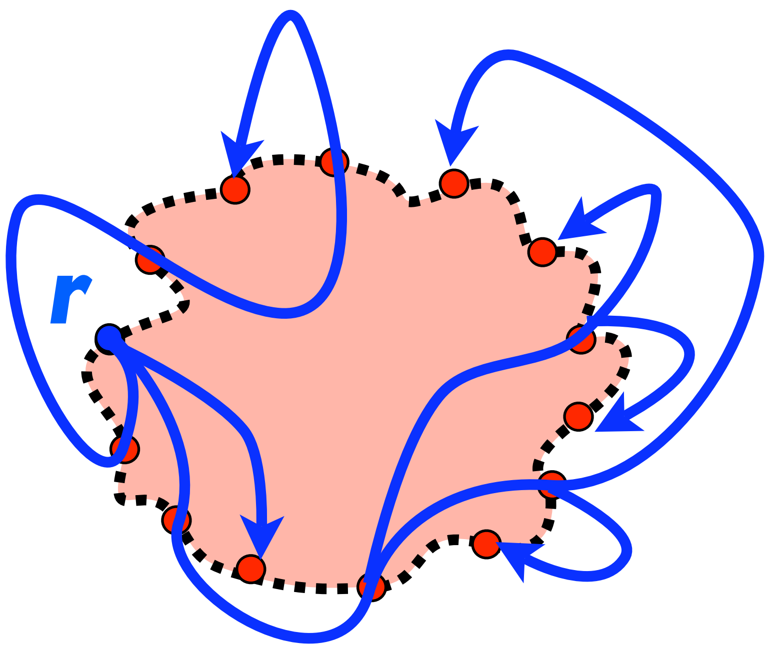
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

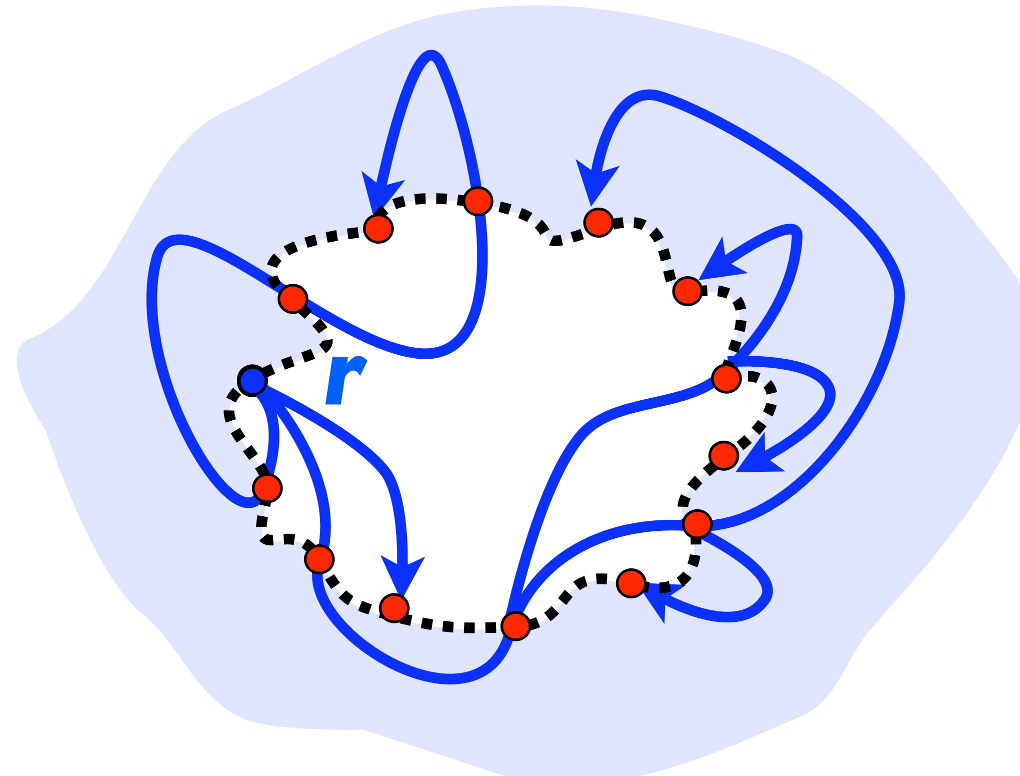
I. recursion

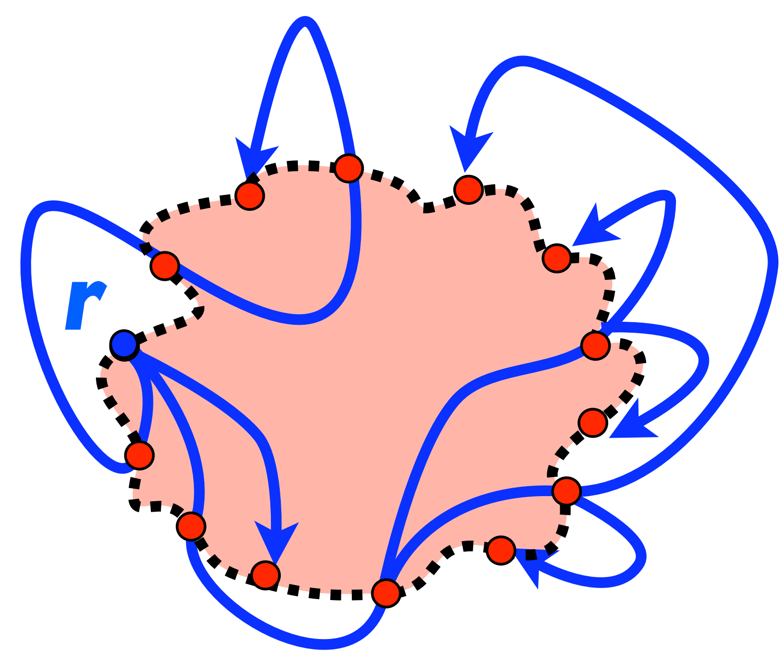
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

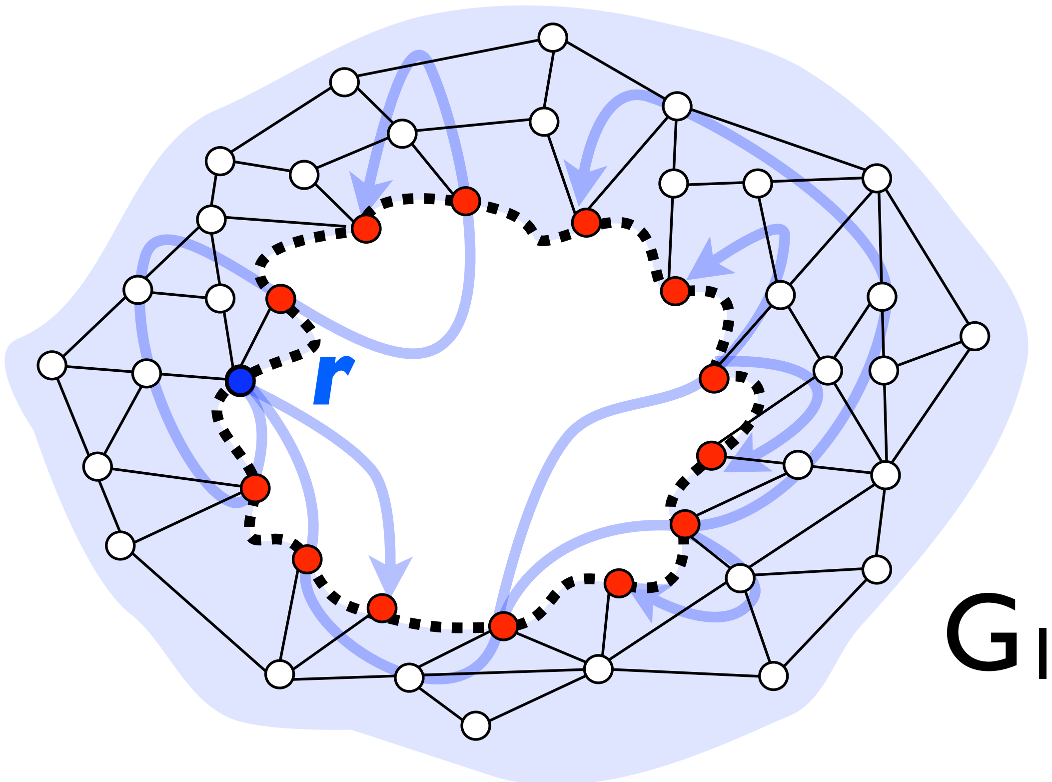
I. recursion

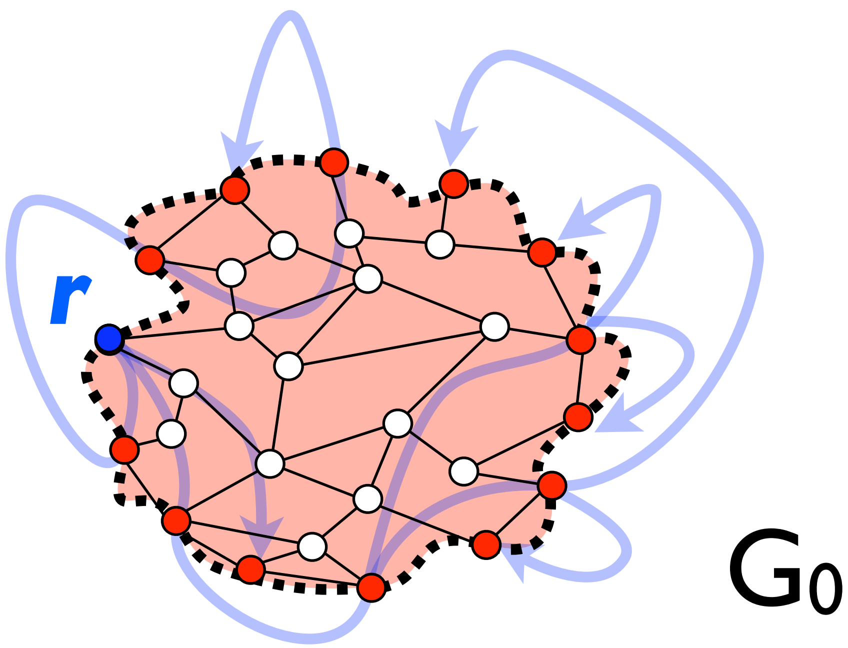
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

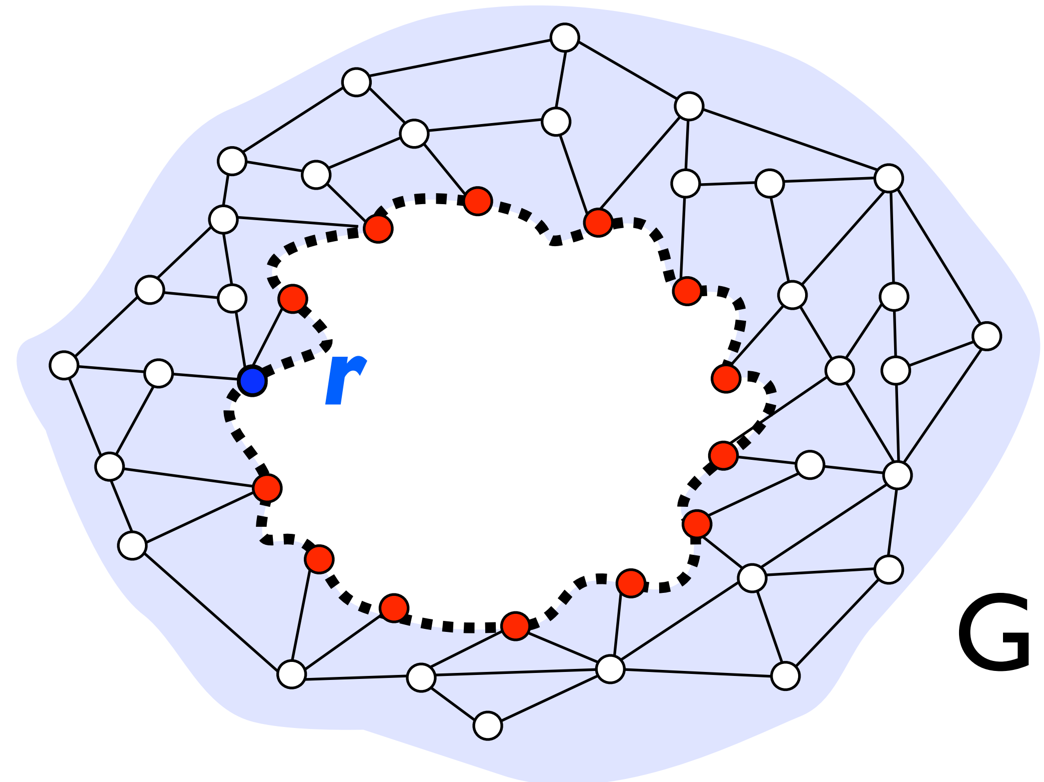
I. recursion

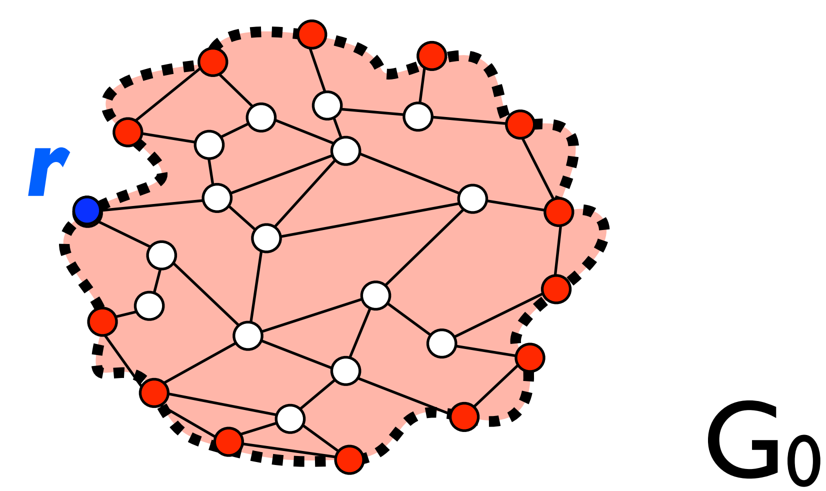
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

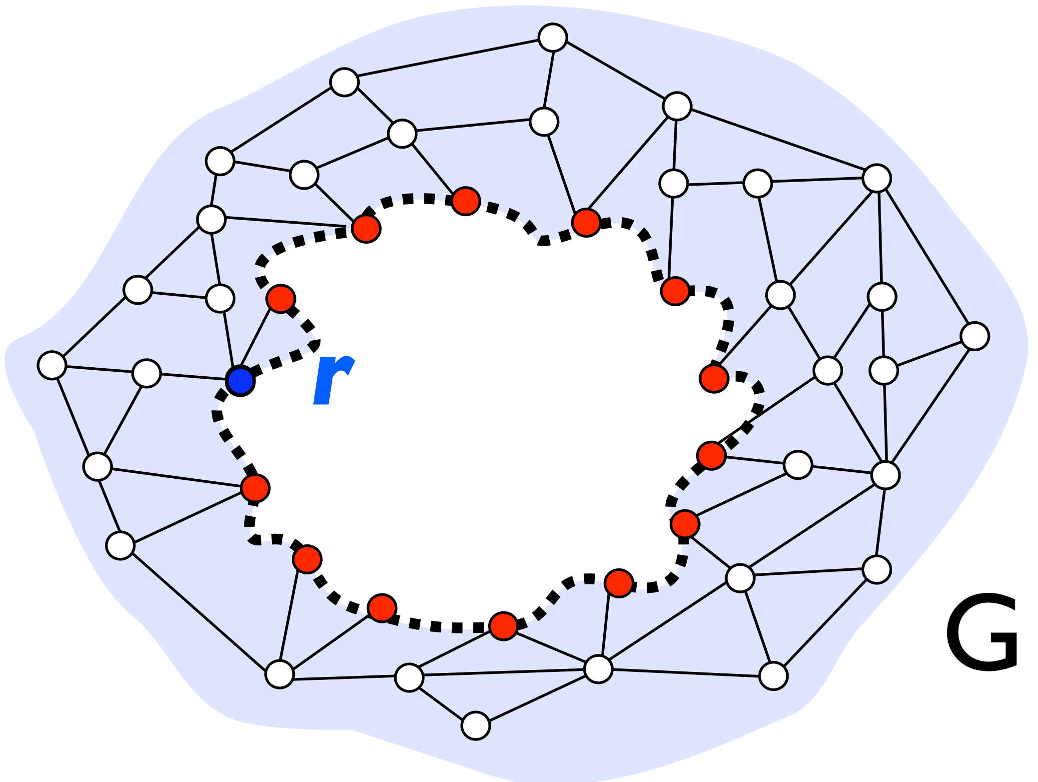
I. recursion

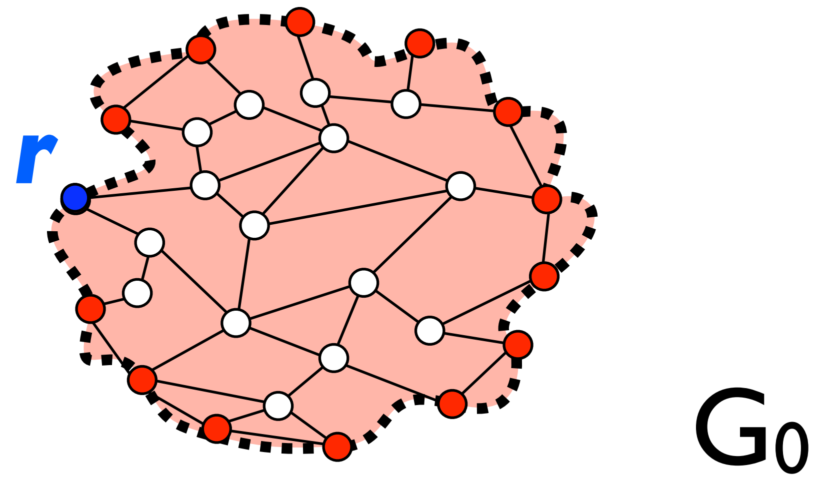
II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting





High-Level View

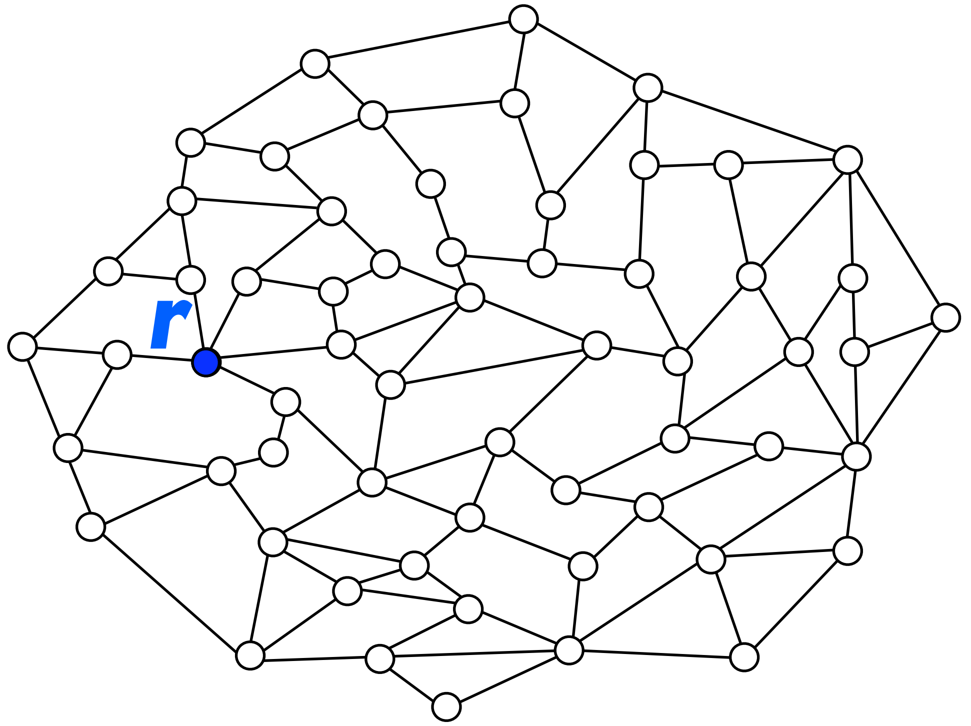
I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

V. rerooting



High-Level View

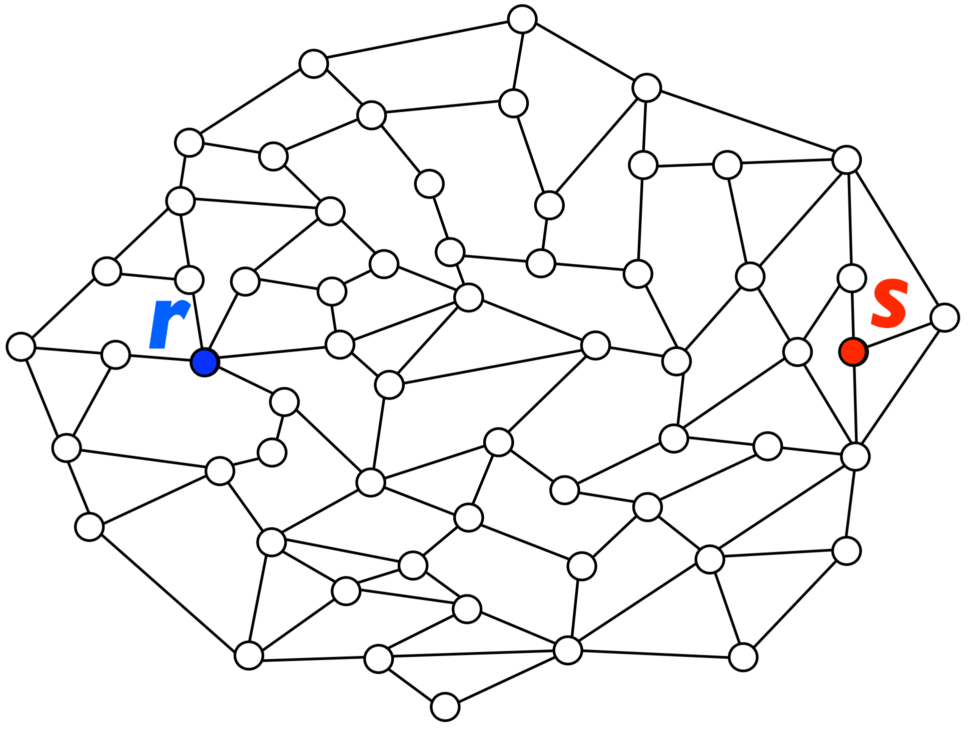
I. recursion

II. boundary to boundary distances in G_i

III. r-to-boundary distances in G

IV. distances from r in G

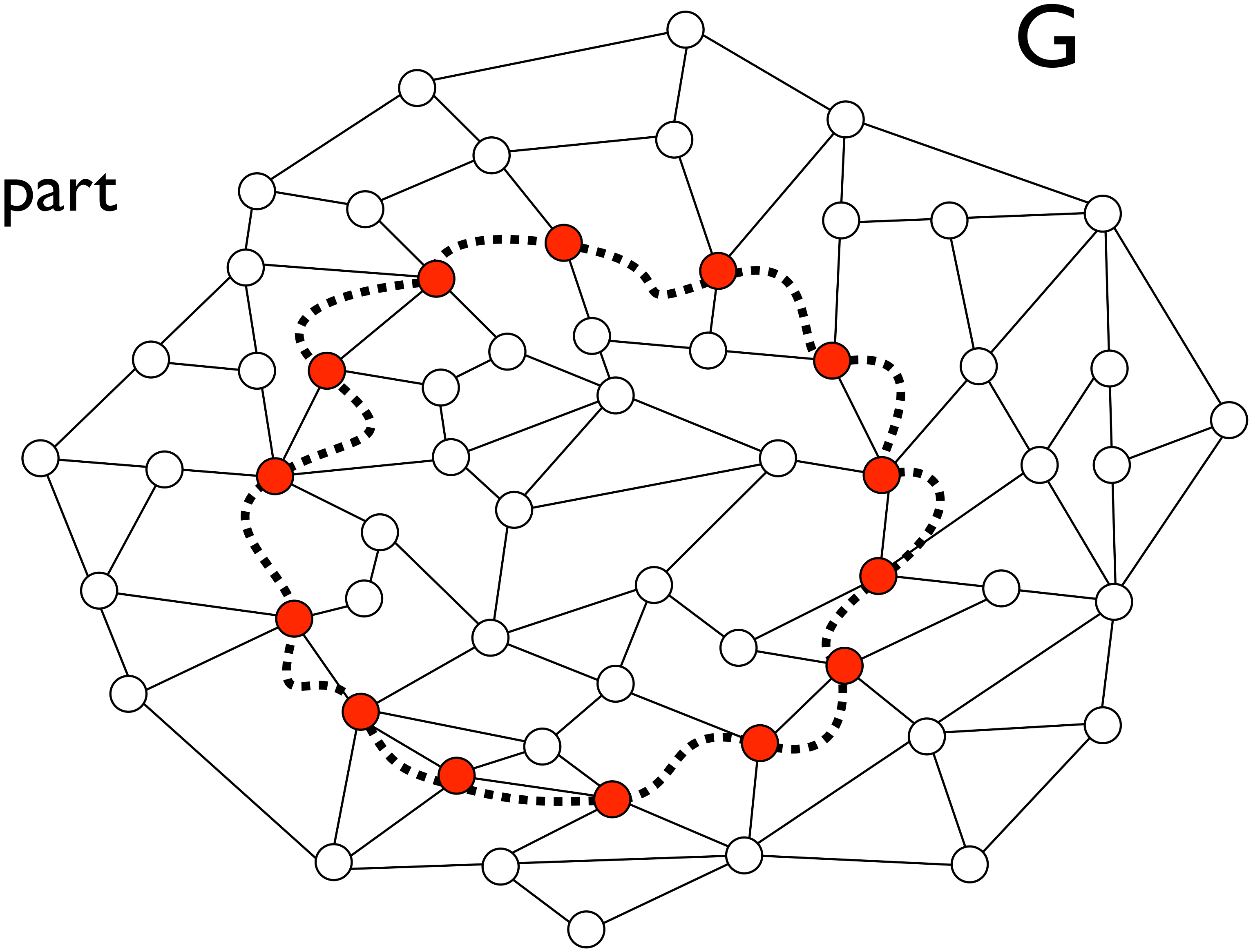
V. rerooting



I. Recursive Step

Planar Separator:

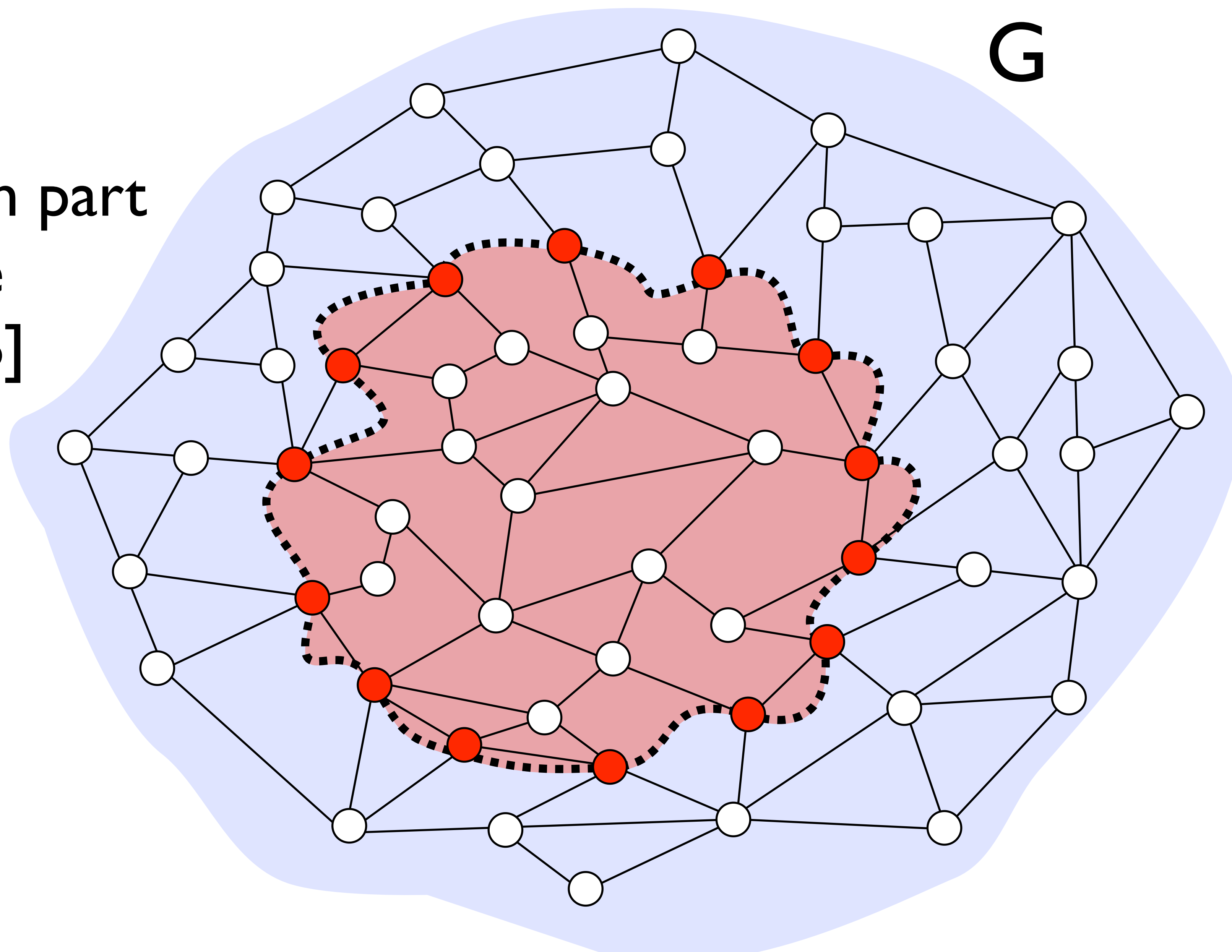
- $O\left(\sqrt{n}\right)$ boundary nodes
- At most $2n/3$ nodes in each part
- Can be found in $O\left(n\right)$ time [Lipton-Tarjan 79, Miller 86]



I. Recursive Step

Planar Separator:

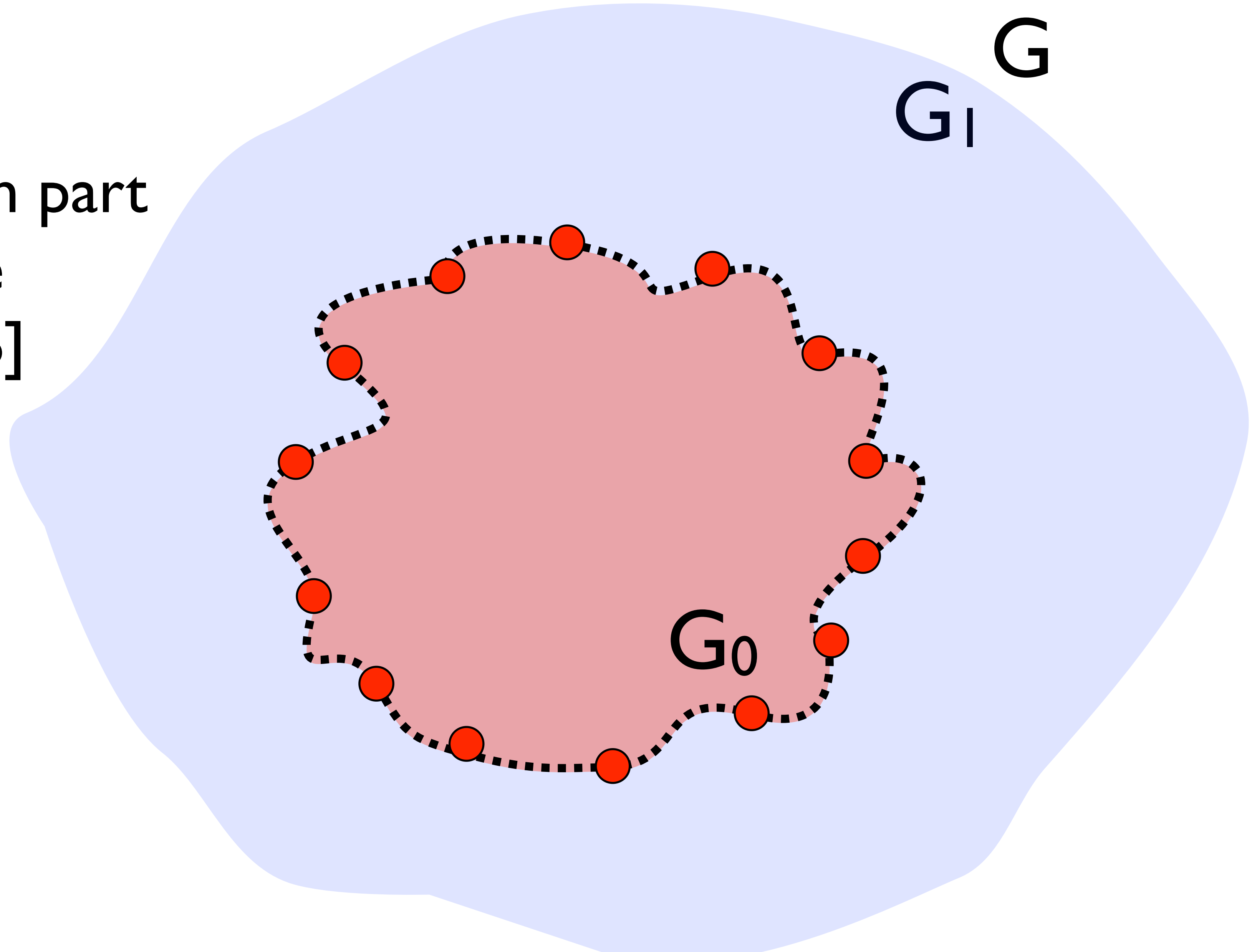
- $O\left(\sqrt{n}\right)$ boundary nodes
- At most $2n/3$ nodes in each part
- Can be found in $O\left(n\right)$ time [Lipton-Tarjan 79, Miller 86]



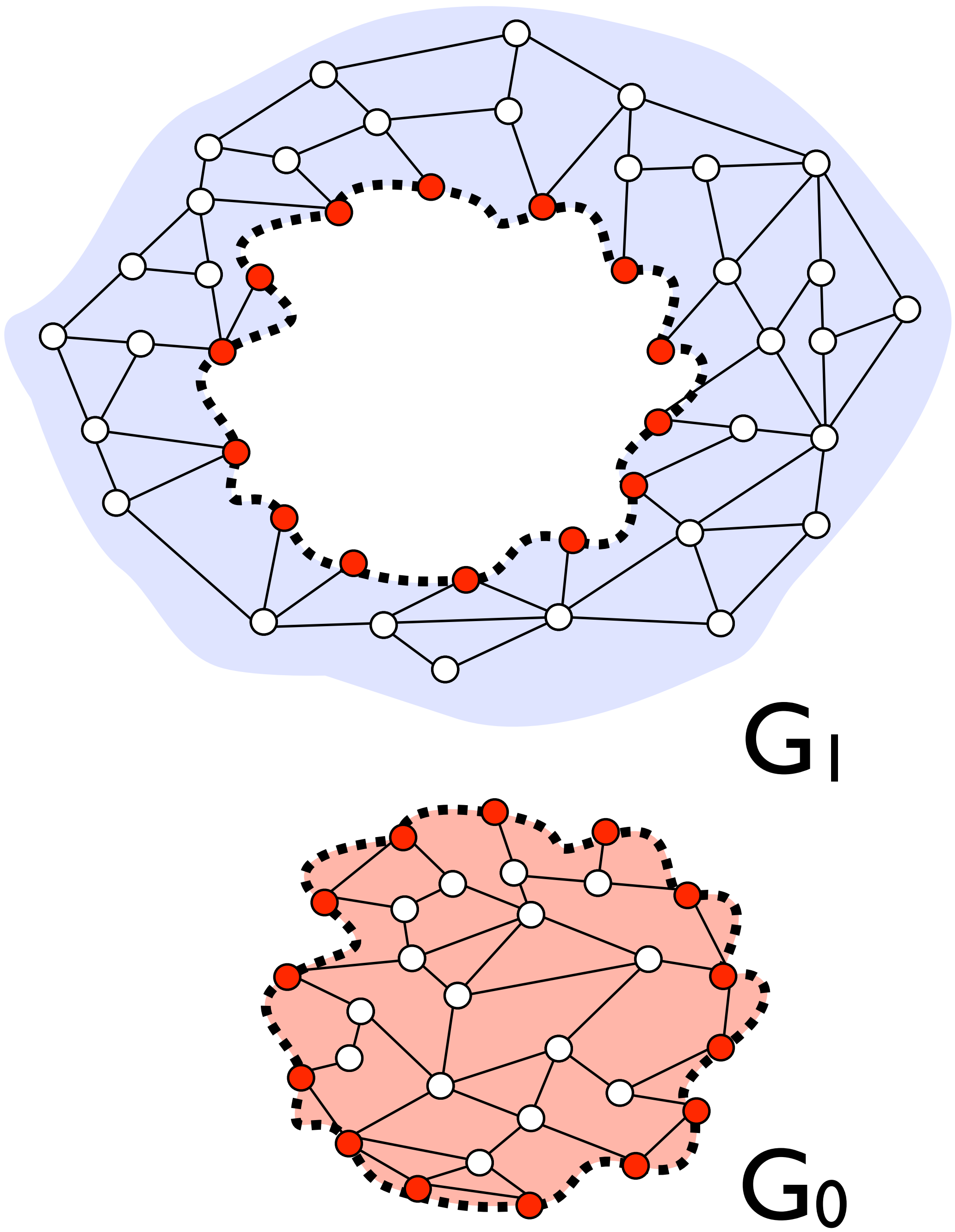
I. Recursive Step

Planar Separator:

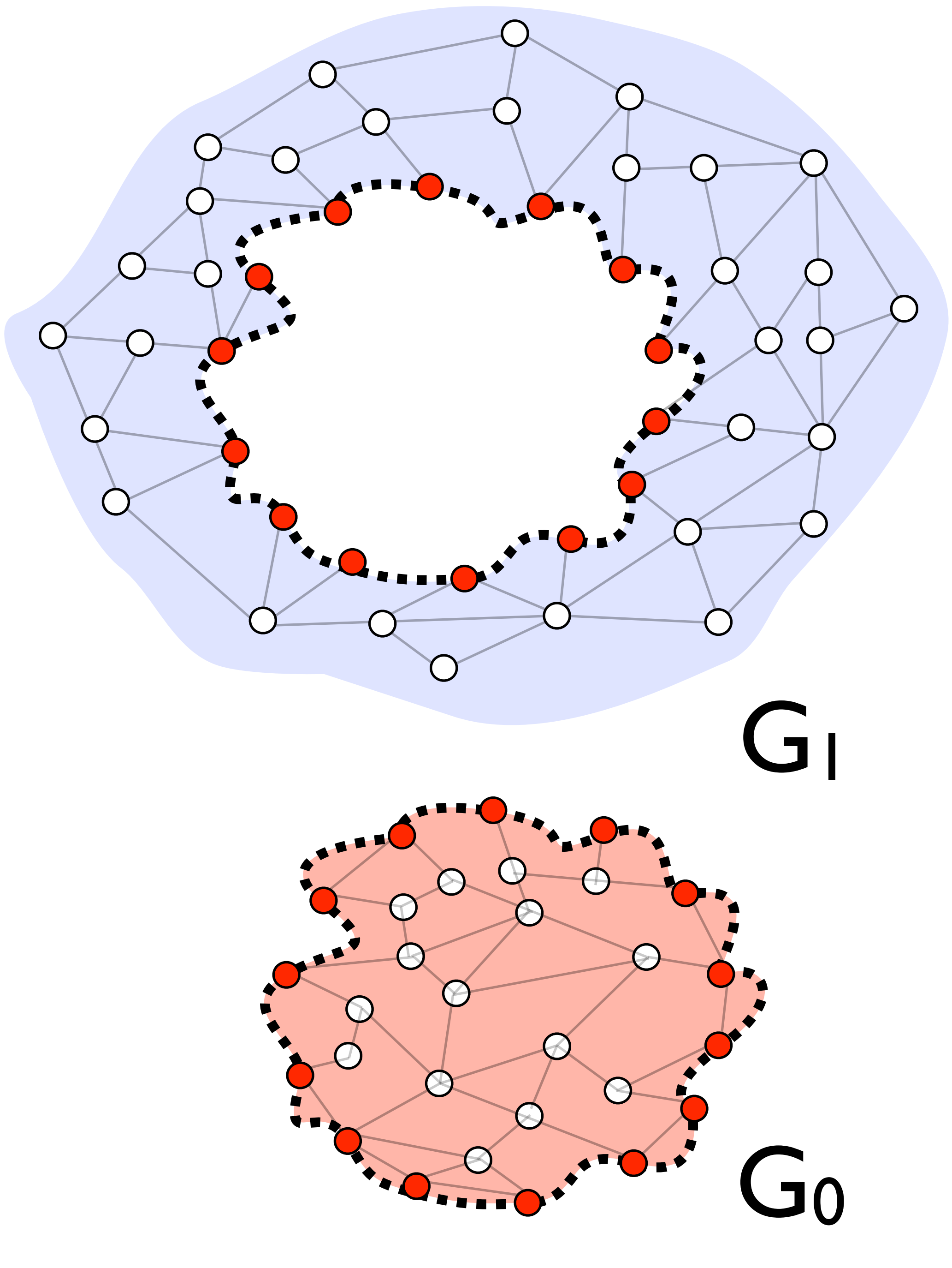
- $O\left(\sqrt{n}\right)$ boundary nodes
- At most $2n/3$ nodes in each part
- Can be found in $O\left(n\right)$ time [Lipton-Tarjan 79, Miller 86]



I. Recursive Step

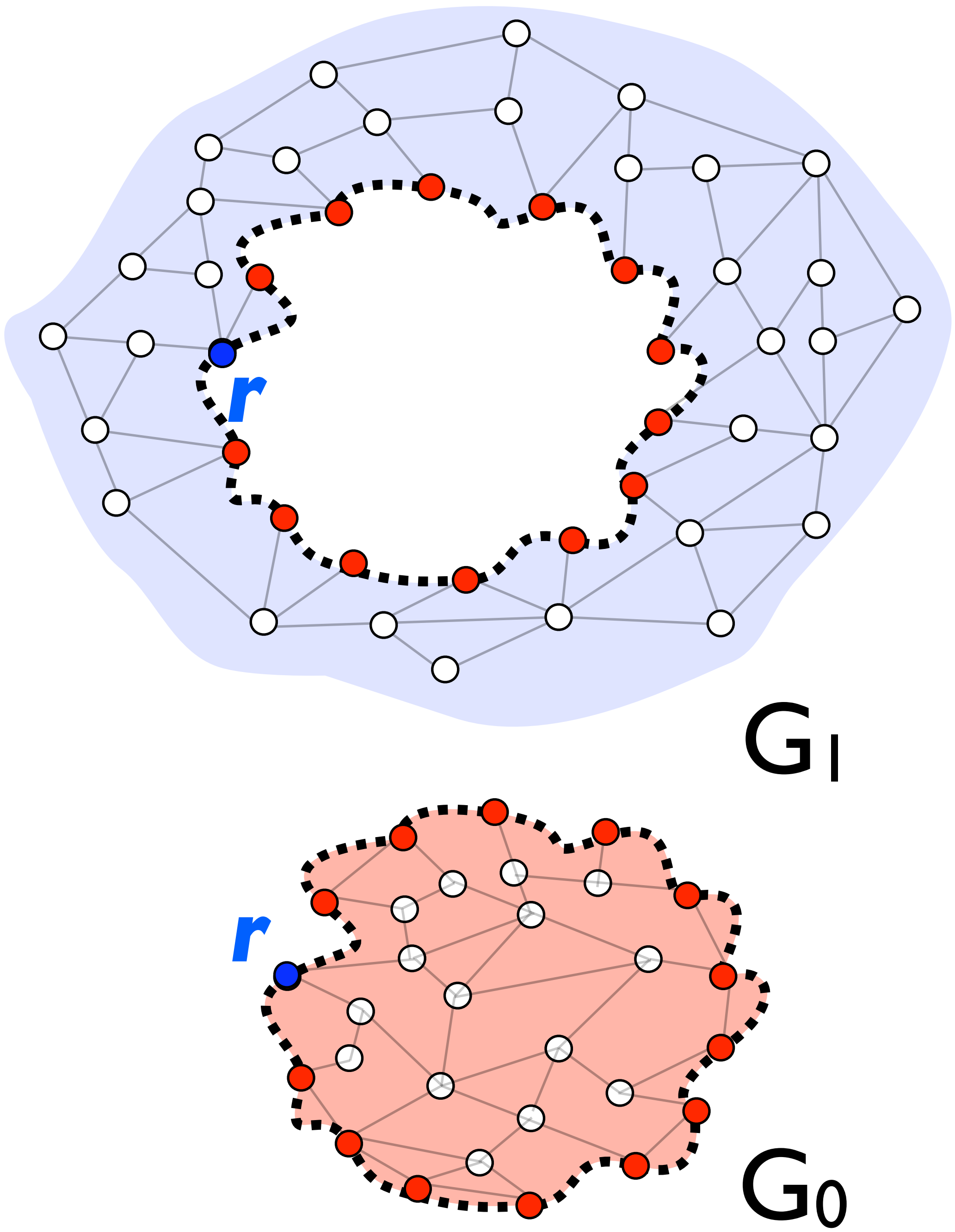


I. Recursive Step



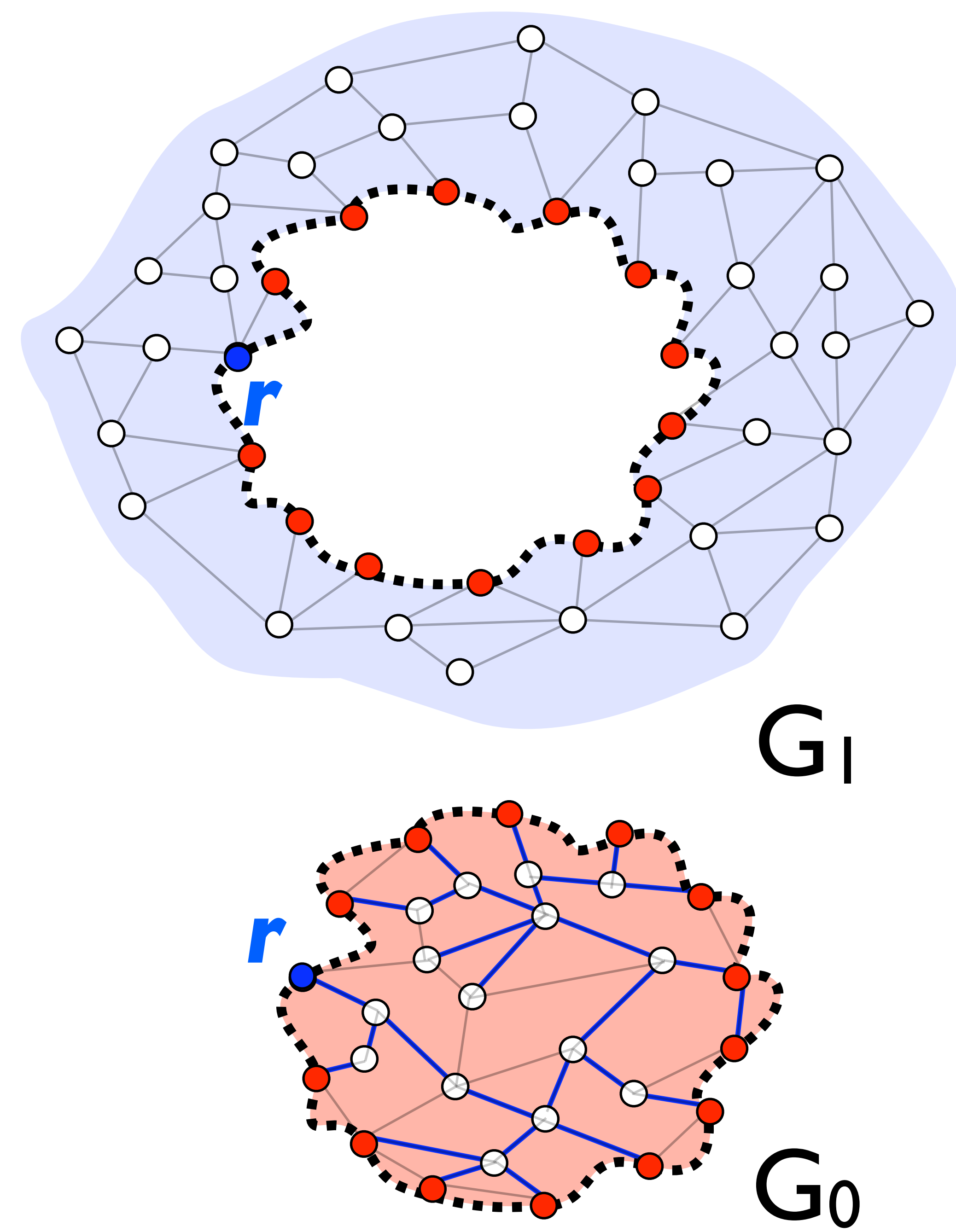
1. Recursive Step

- Choose an arbitrary boundary node r



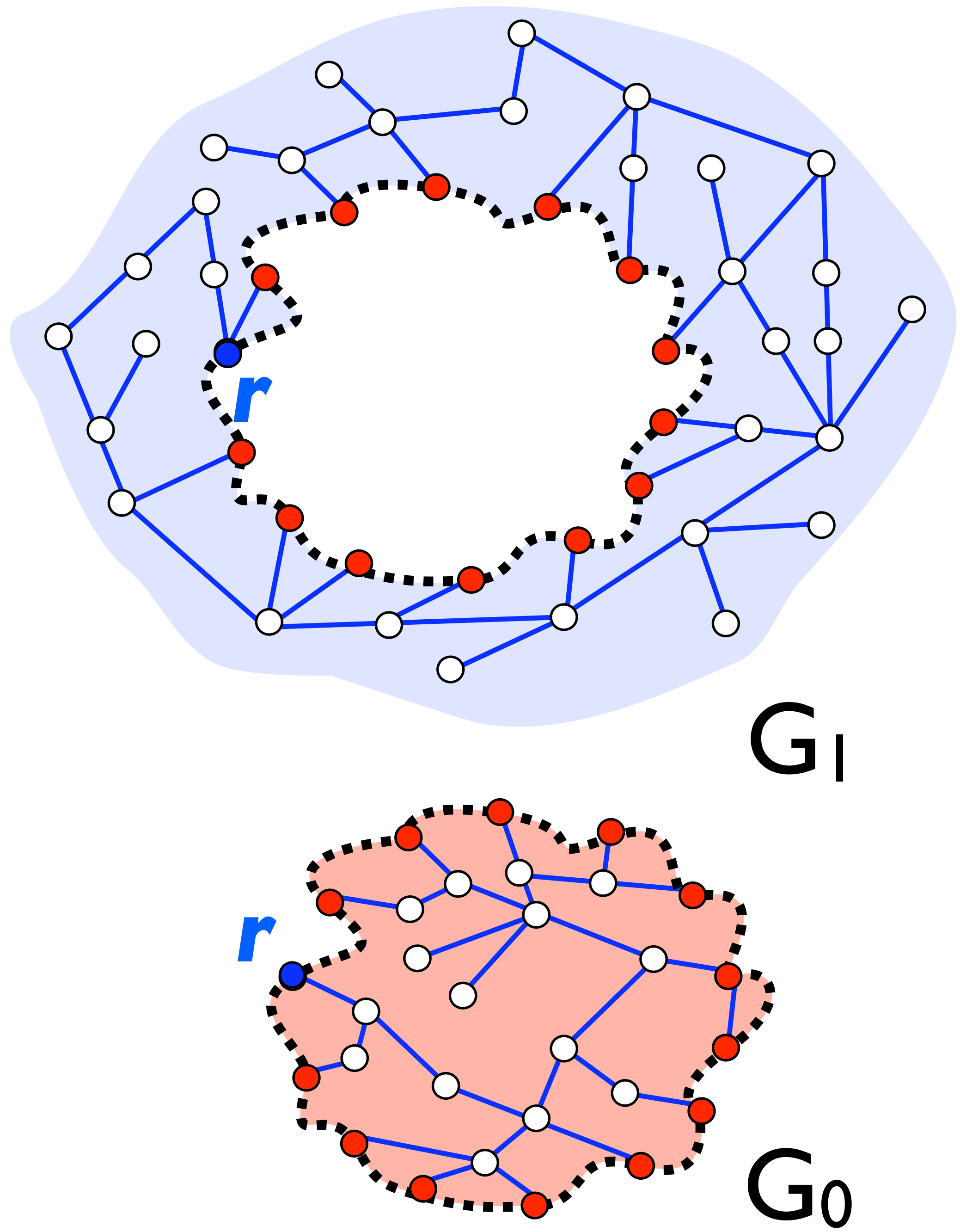
I. Recursive Step

- Choose an arbitrary boundary node r
- Recursively compute distances from r within G_0



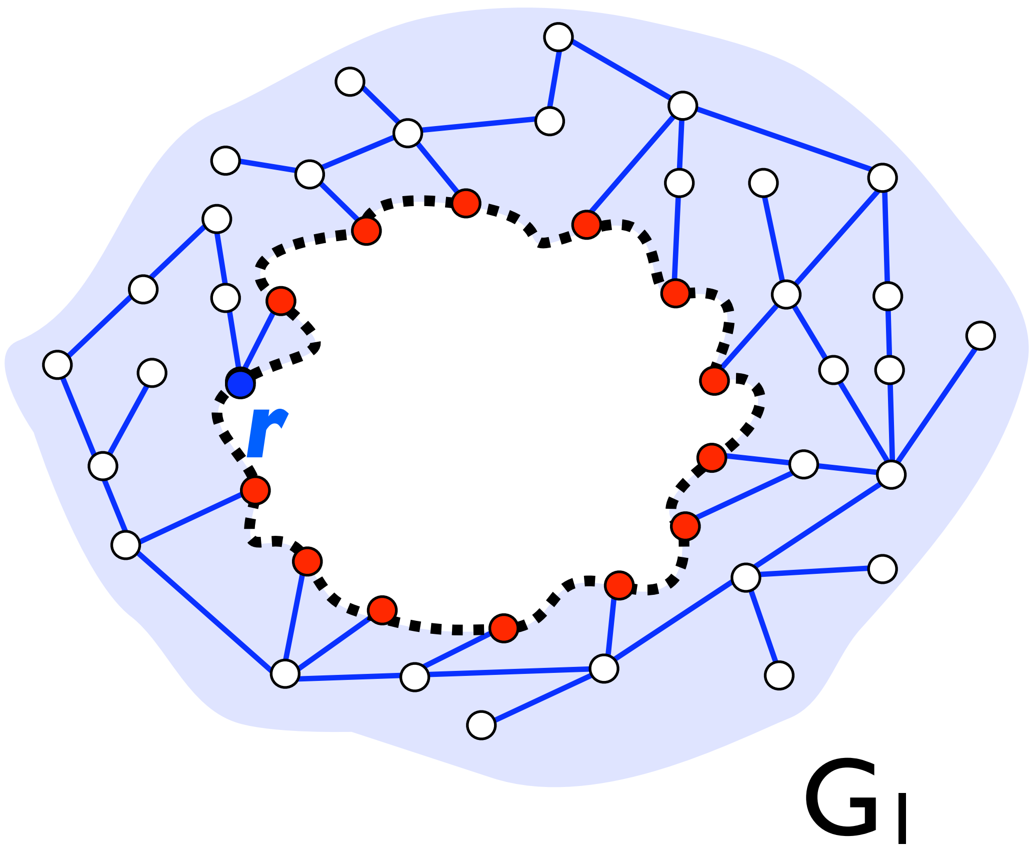
I. Recursive Step

- Choose an arbitrary boundary node r
- Recursively compute distances from r within G_0
- Recursively compute distances from r within G_1



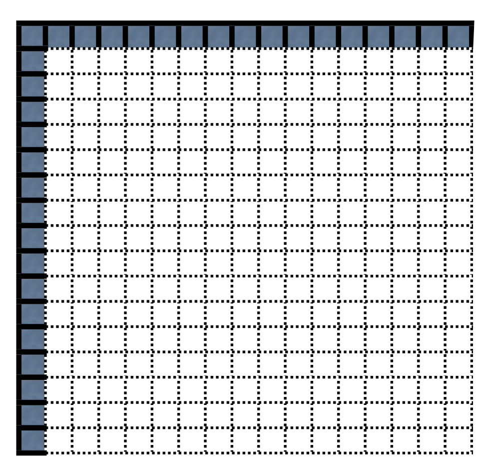
II. Boundary-to-Boundary Distances in G_i

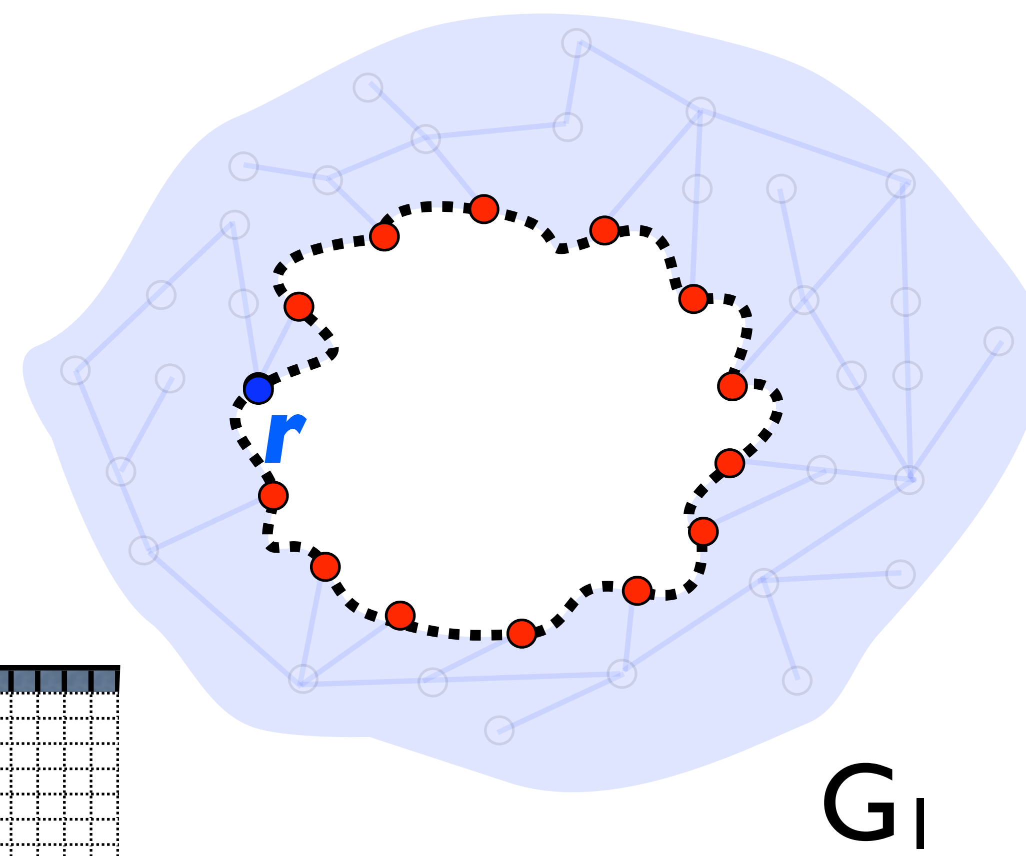
- Compute all boundary-to-boundary distances within G_1
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-r distances in G_1
- Repeat for G_0



II. Boundary-to-Boundary Distances in G_i

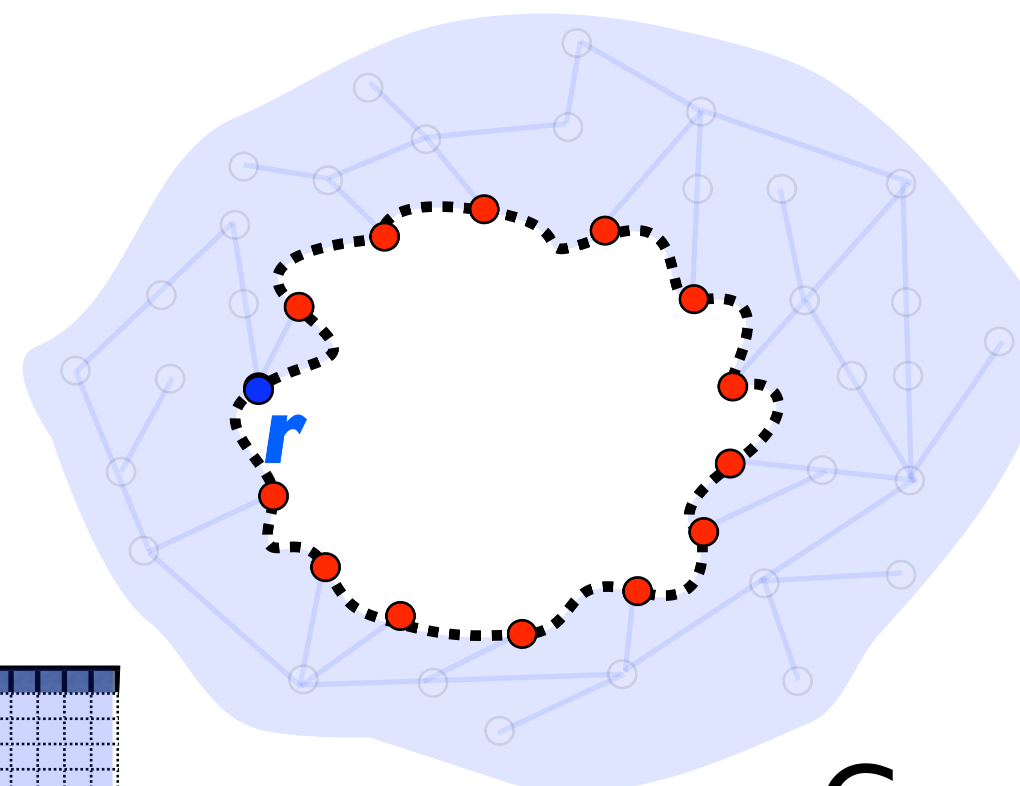
- Compute all boundary-to-boundary distances within G_1
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-r distances in G_1
- Repeat for G_0



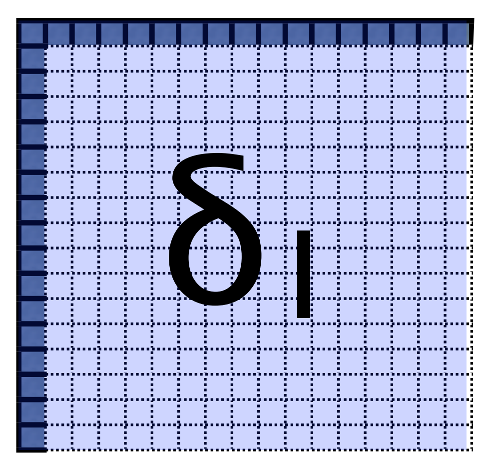


II. Boundary-to-Boundary Distances in G_i

- Compute all boundary-to-boundary distances within G_1
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-r distances in G_1
- Repeat for G_0

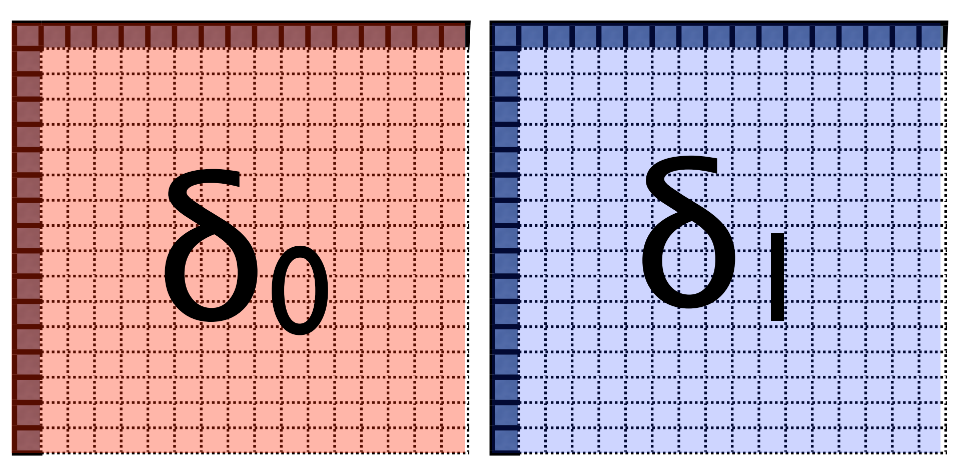


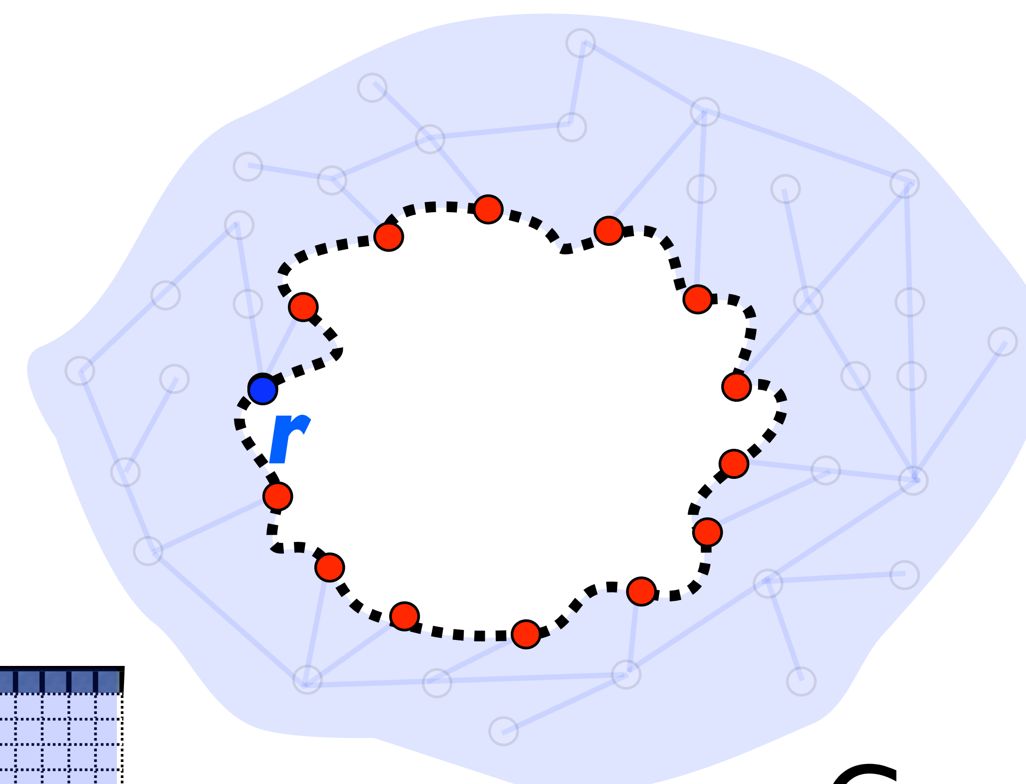
G_1



II. Boundary-to-Boundary Distances in G_i

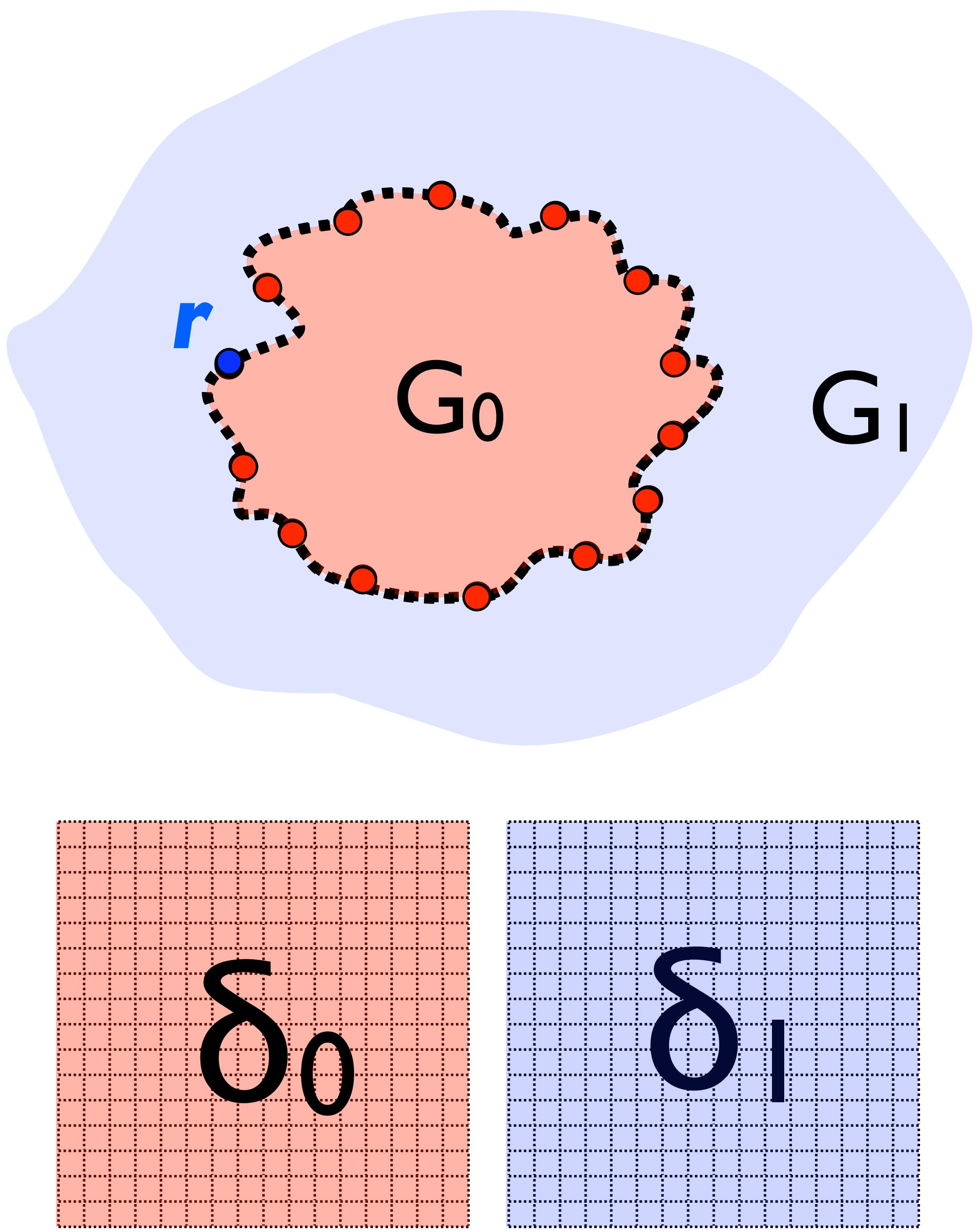
- Compute all boundary-to-boundary distances within G_1
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-r distances in G_1
- Repeat for G_0





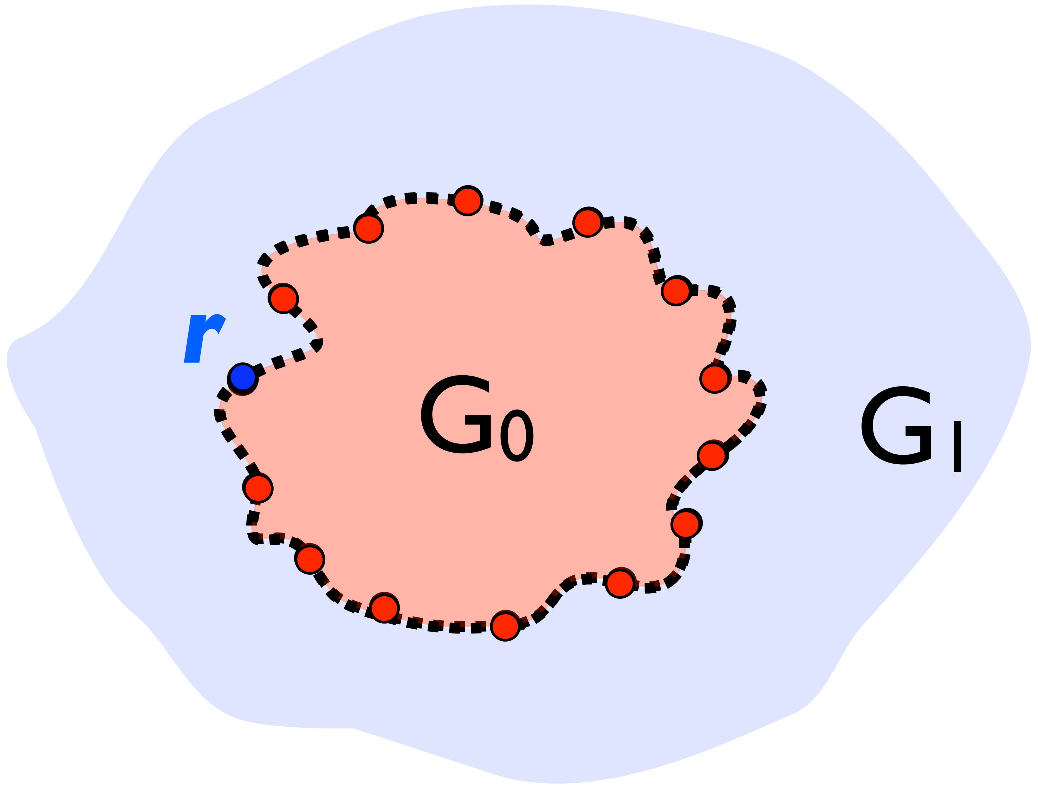
G_1

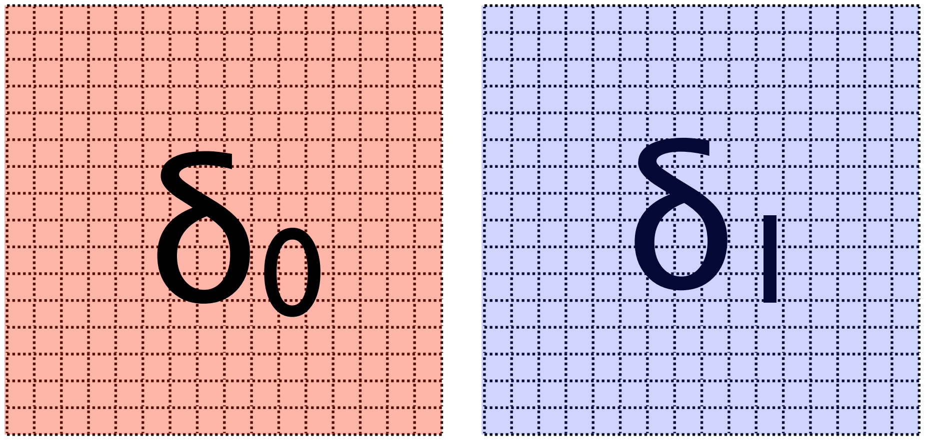
III. r-to-boundary Distances in G



III. r-to-boundary Distances in G

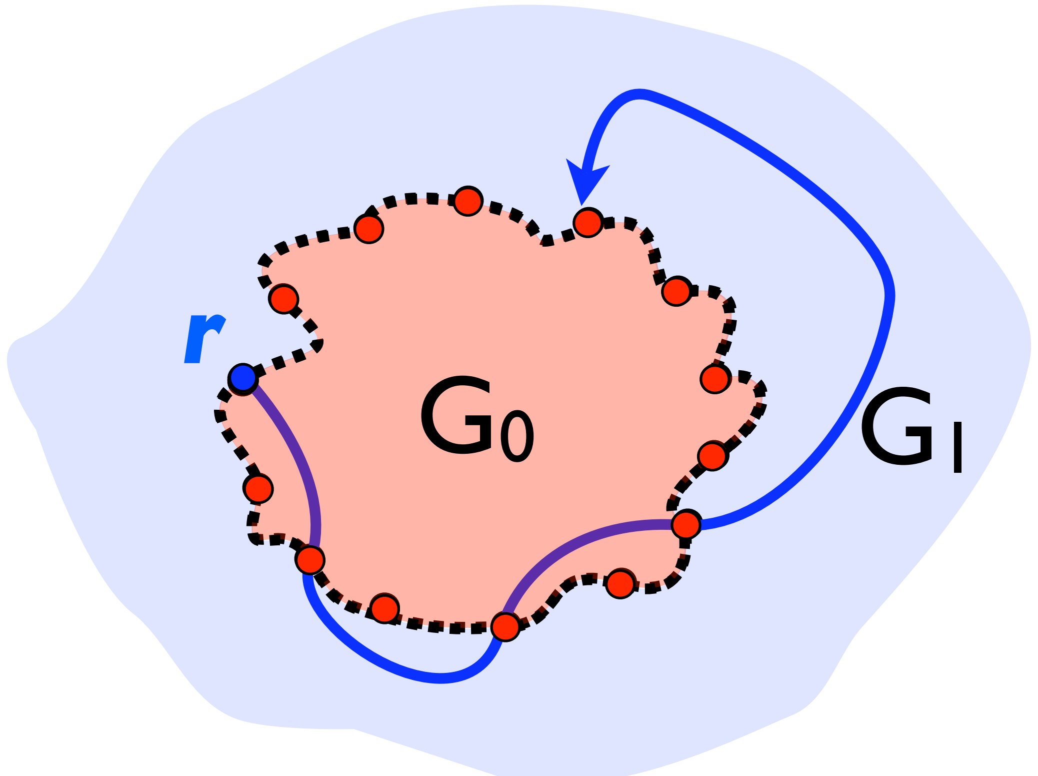
- Compute distances from r to all boundary nodes

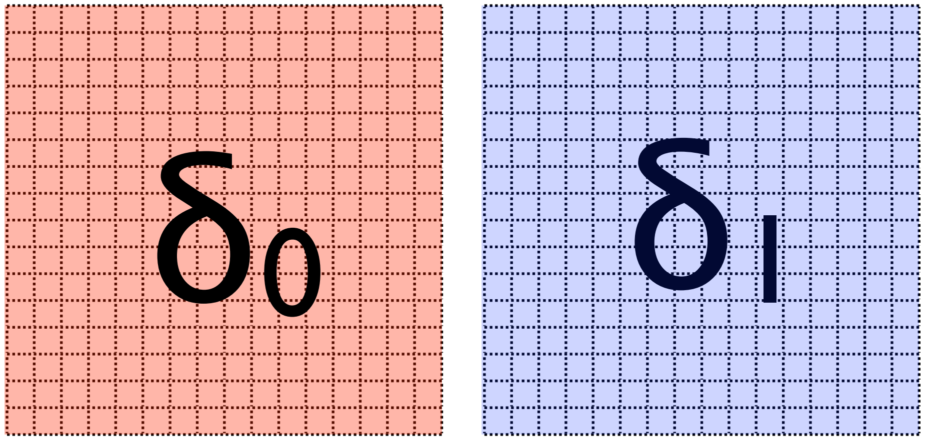




III. r-to-boundary Distances in G

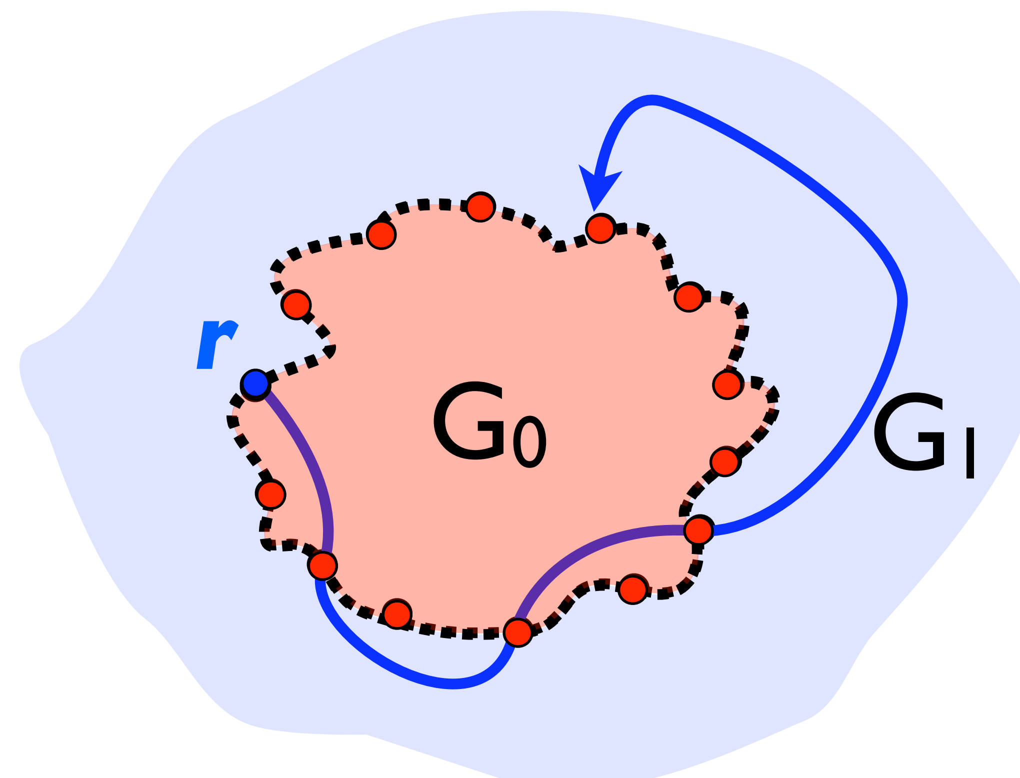
- Compute distances from r to all boundary nodes
  - Shortest path in G consists of alternating boundary-to-boundary shortest paths in G_0 and G_1



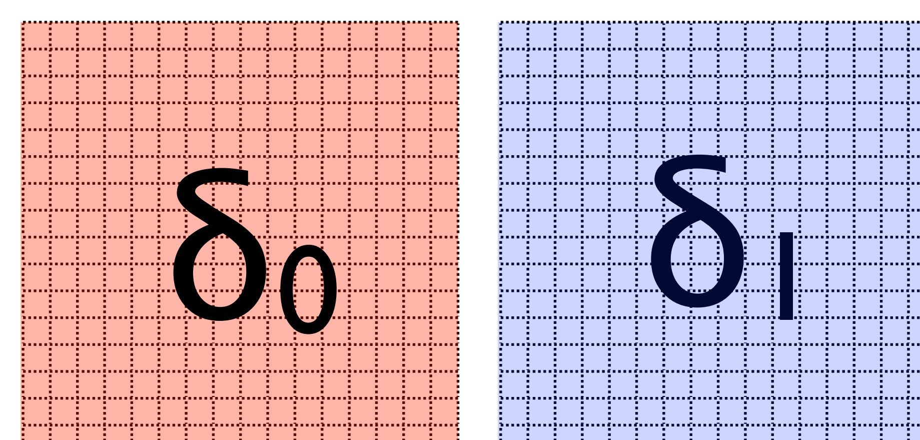


III. r-to-boundary Distances in G

- Compute distances from r to all boundary nodes
  - Shortest path in G consists of alternating boundary-to-boundary shortest paths in G_0 and G_1
  - "Bellman-Ford" using just boundary-to-boundary distances



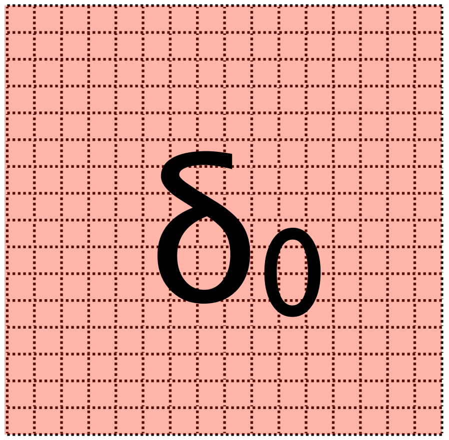
$$\forall v \;\; e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w,v]\}$$

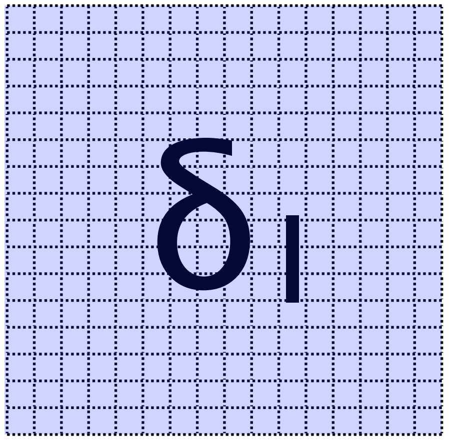


III. r-to-boundary Distances in G

$$\forall v \;\; e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w, v]\}$$

- All iterations in $O\left(n^{3/2}\right)$ [Lipton-Rose-Tarjan 1979]
- δ has a Monge non-crossing property [Fakcharoenphol-Rao 2001] ⇒ $O\left(n \log^2 n\right)$ time
- We show: $O\left(n\alpha(n)\right)$ time

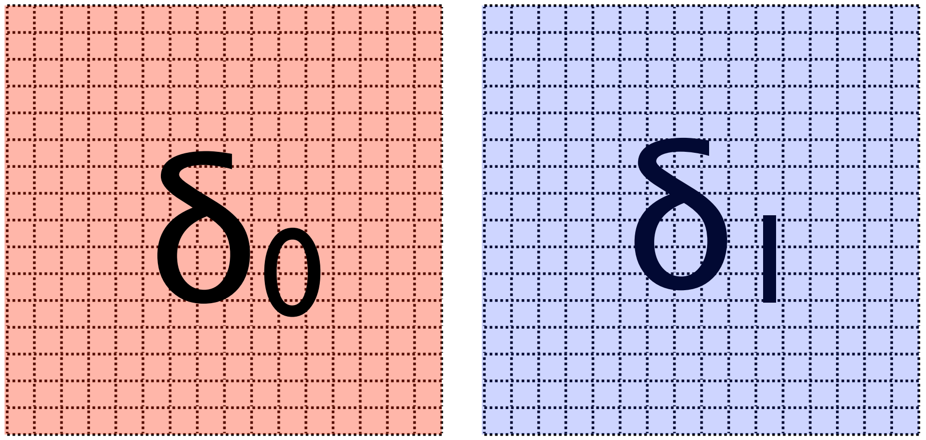




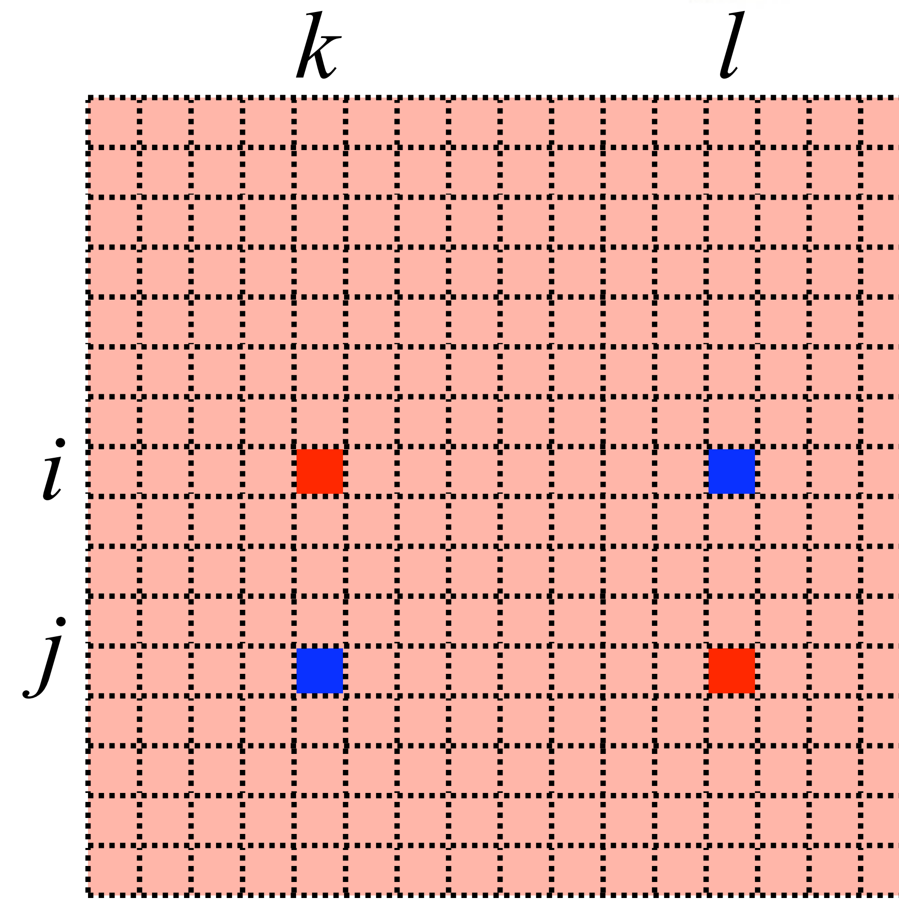
III. r-to-boundary Distances in G

$$\forall v \;\; e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w, v]\}$$

- Think of a matrix whose *w,v* element is
- We want to find all column minima of this matrix

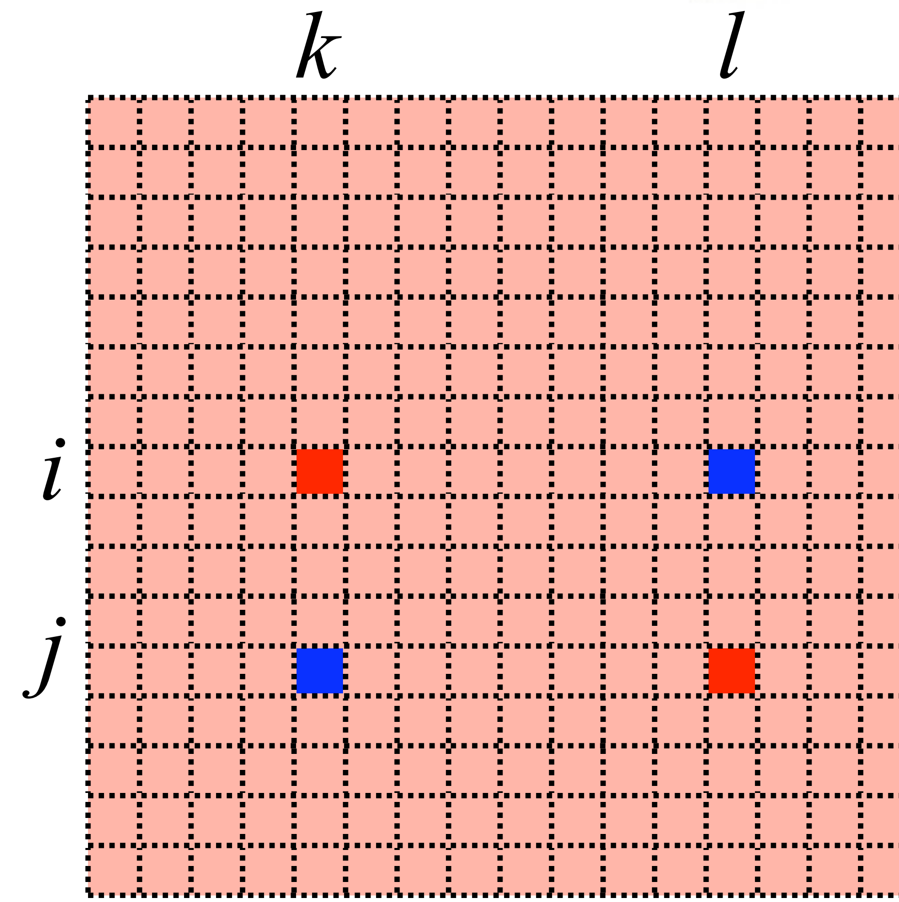


Monge Matrices



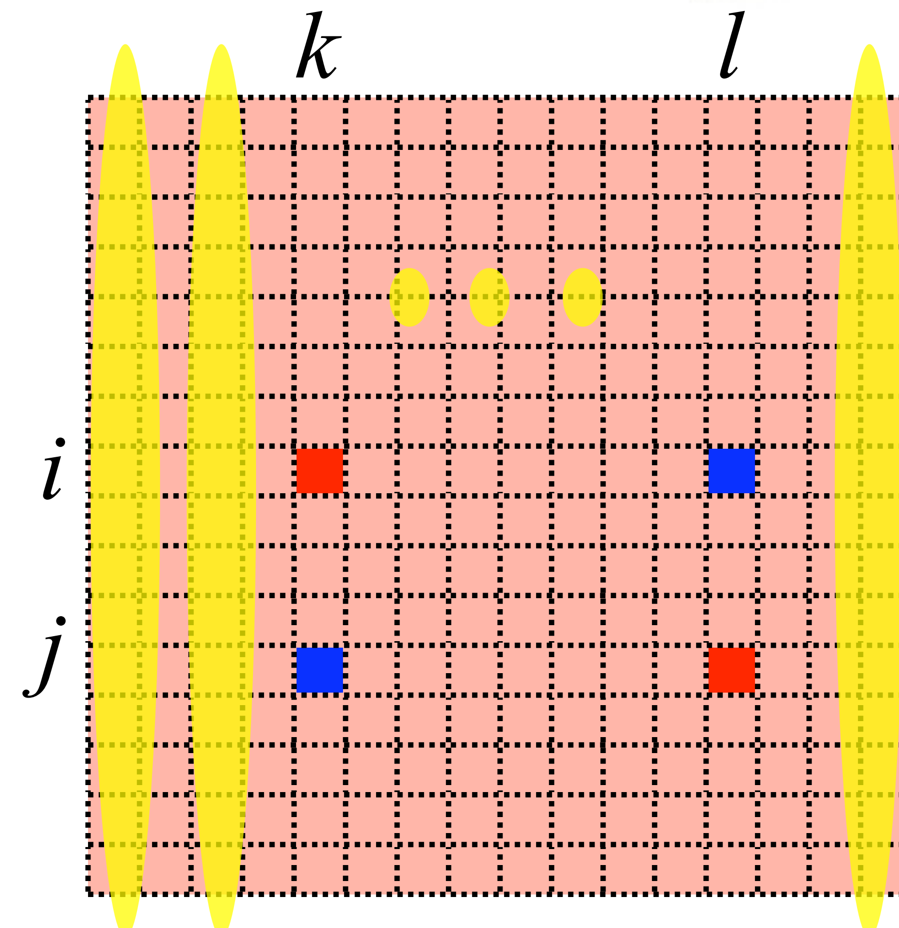
Monge Matrices

- A matrix is Monge if for any $i \leq j, k \leq l$
  $\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$



Monge Matrices

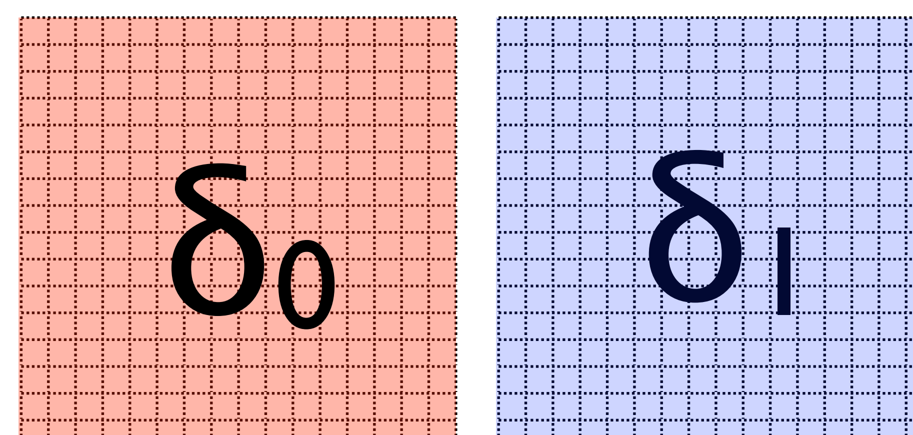
- A matrix is Monge if for any $i \leq j, k \leq l$
  $\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$
- All column minima of an n x n Monge matrix can be found in $O(n)$ time [SMAWK 1989]



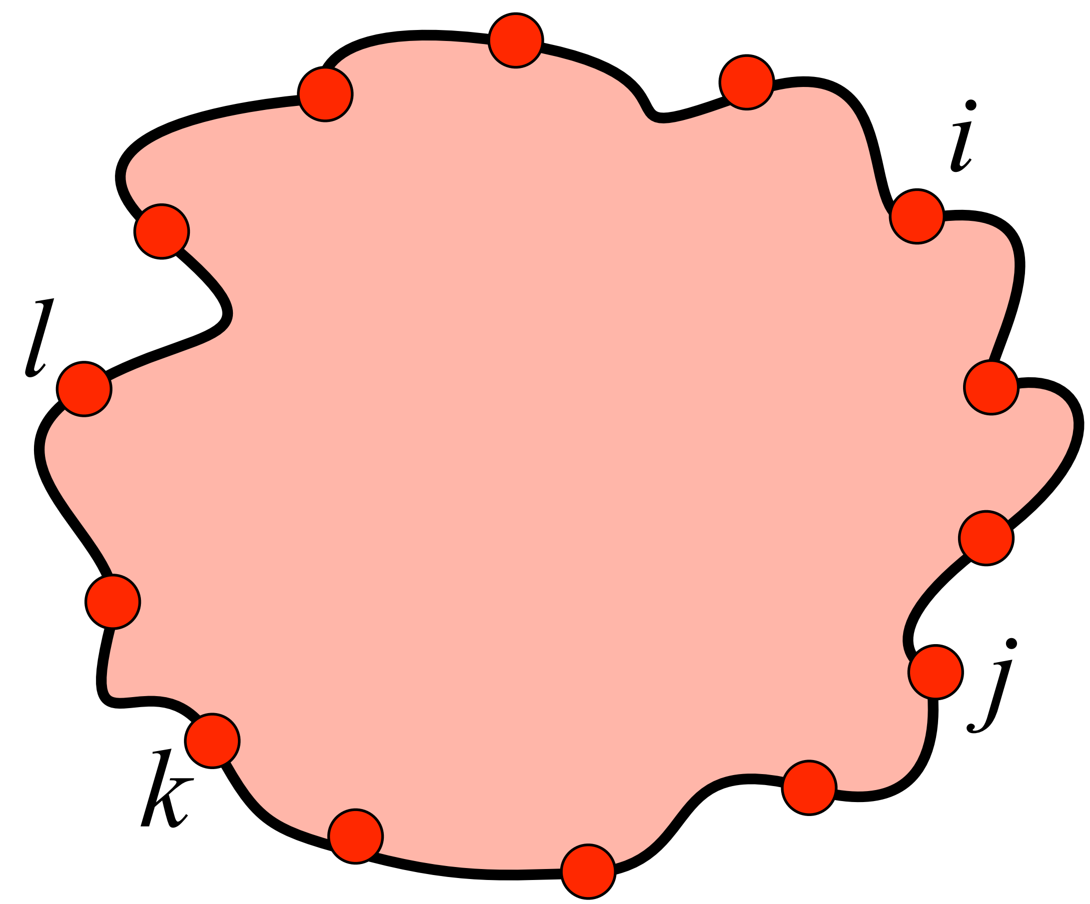
III. r-to-boundary Distances in G

$$\forall v \;\; e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w,v]\}$$

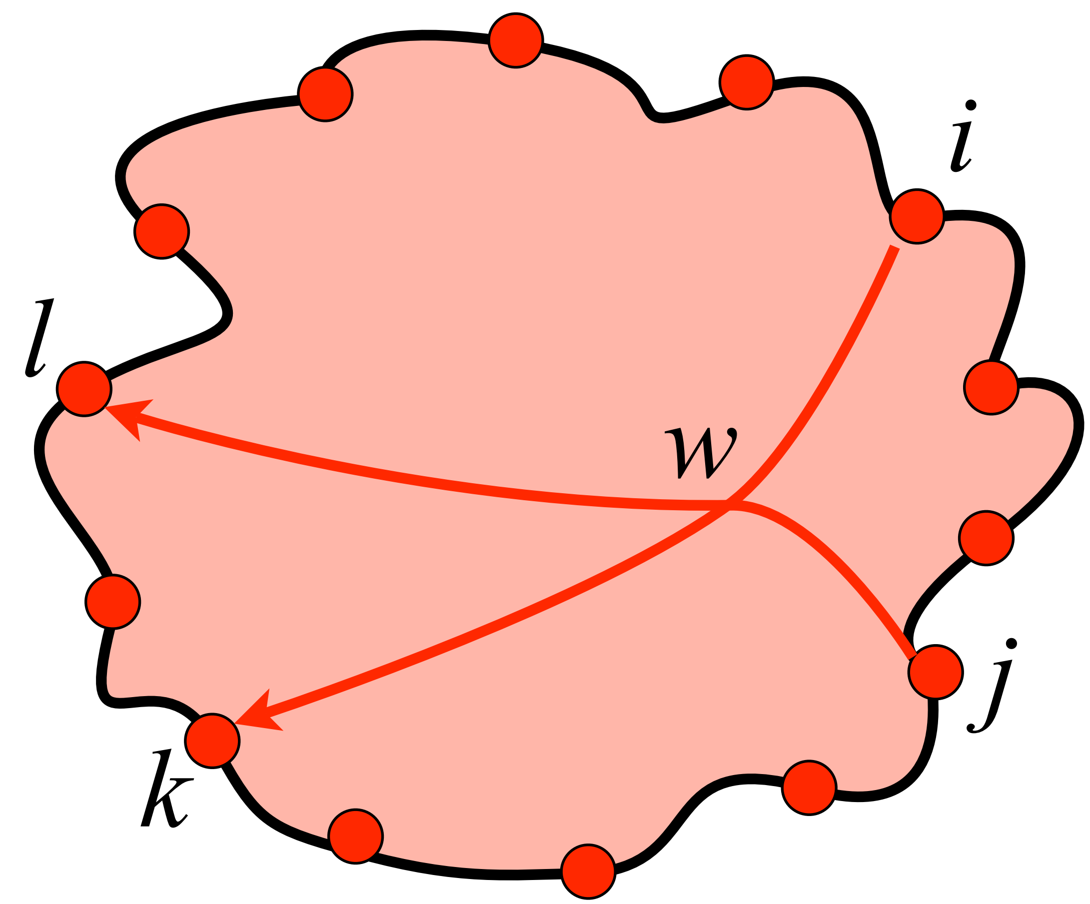
- Think of a matrix whose *w,v* element is
- We want to find all column minima of this matrix
- Show that this matrix is Monge



Crossings and the Monge Property

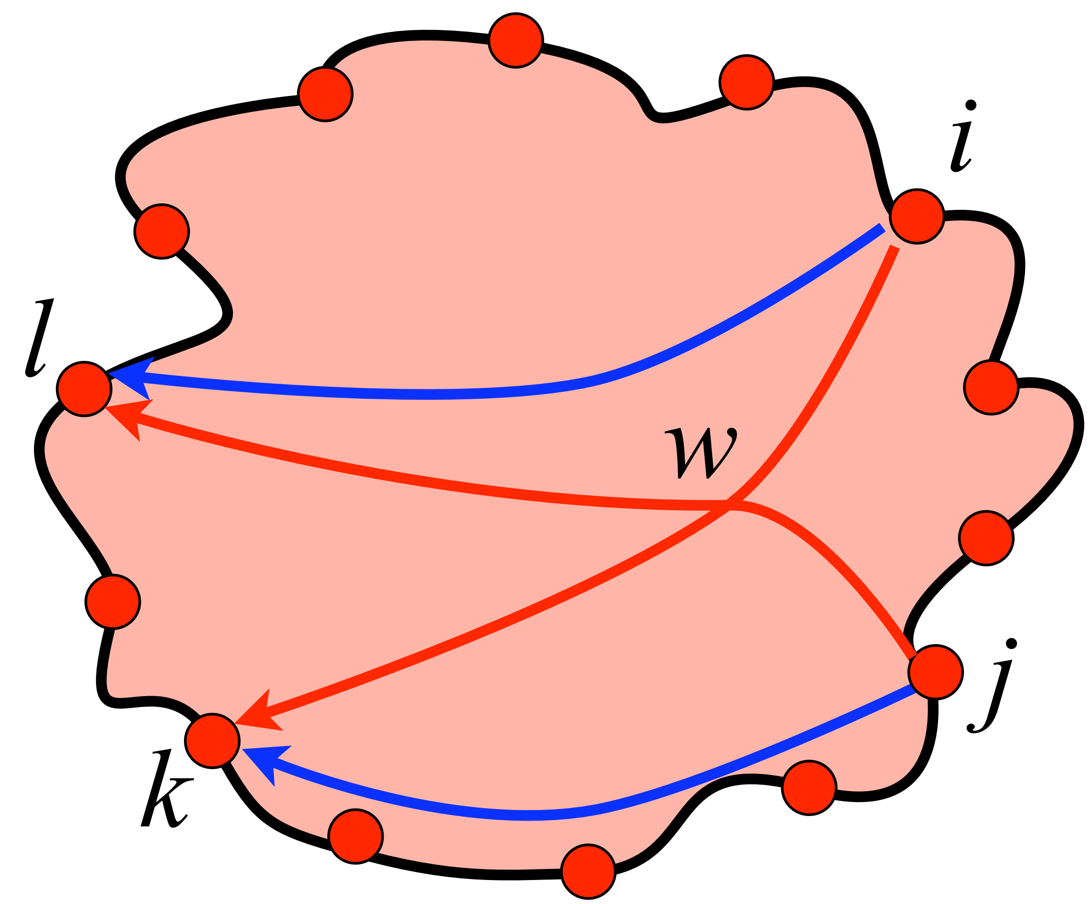


Crossings and the Monge Property



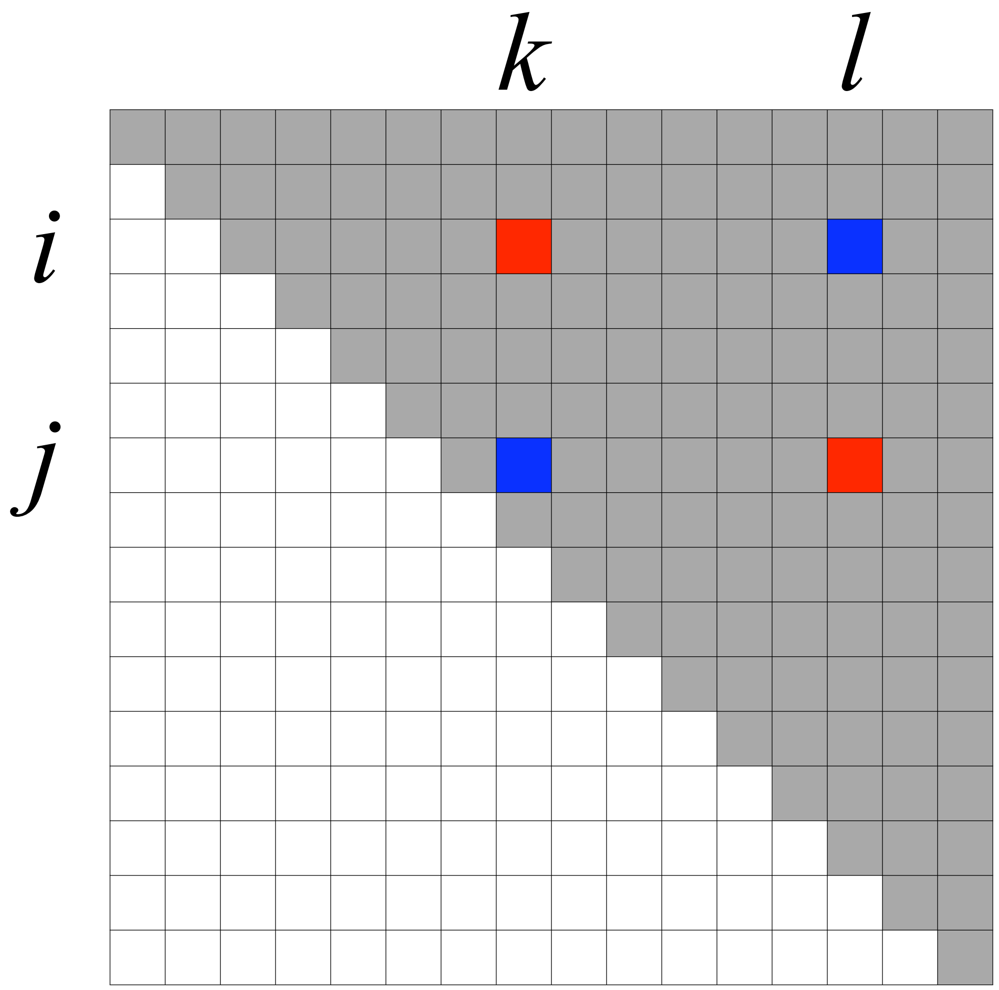
$\delta(i,k) + \delta(j,l)$

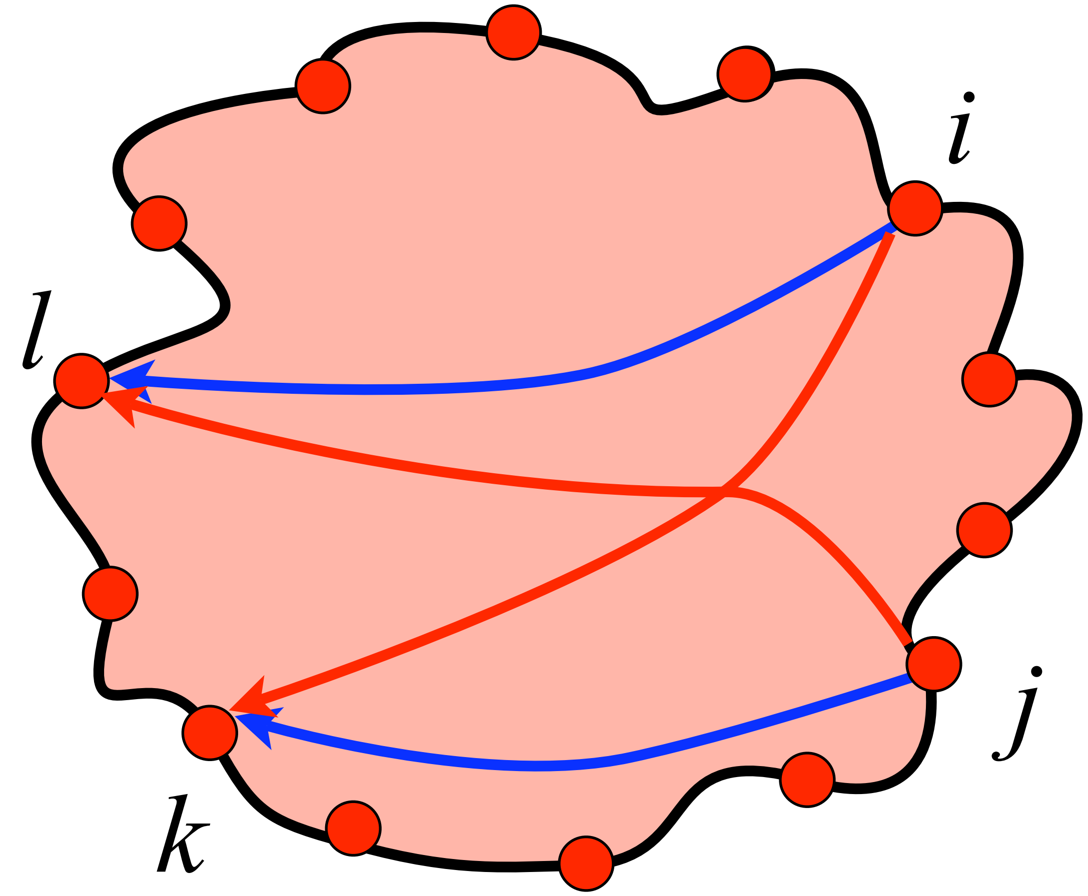
Crossings and the Monge Property



$$\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$$

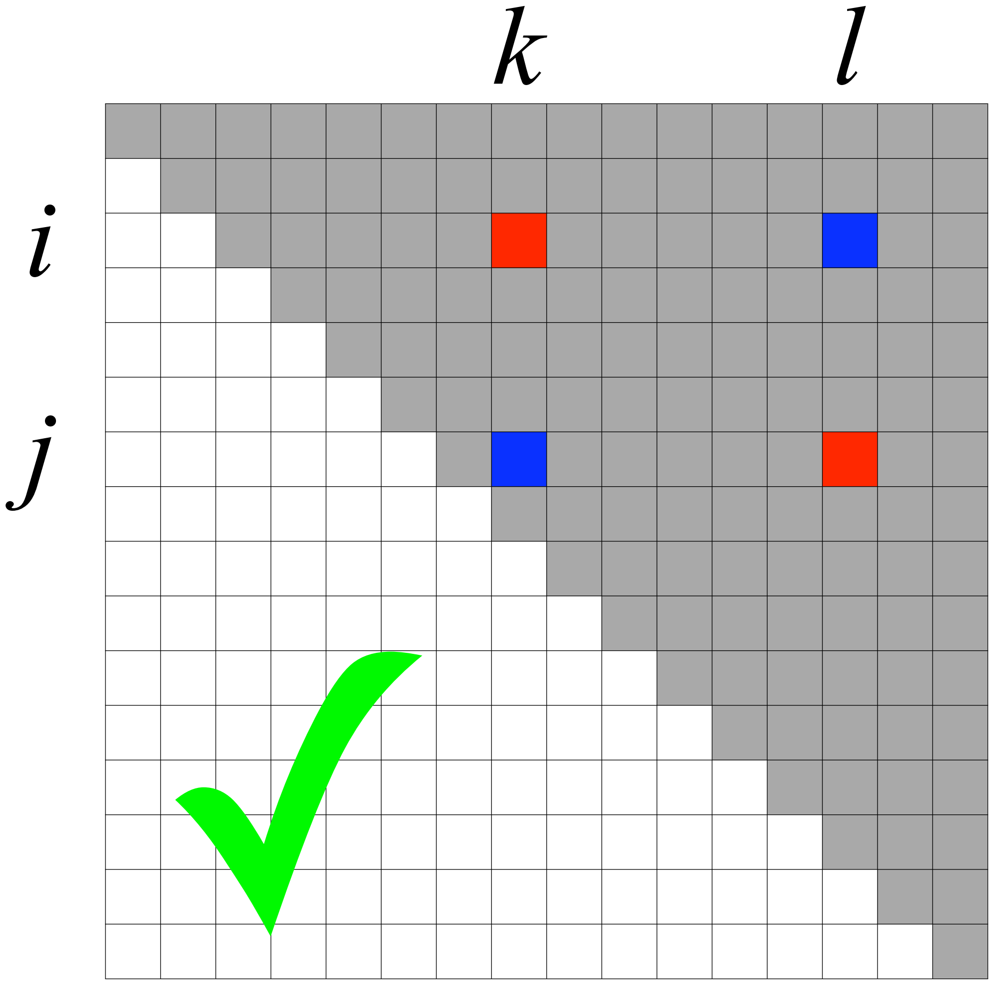
Crossings and the Monge Property

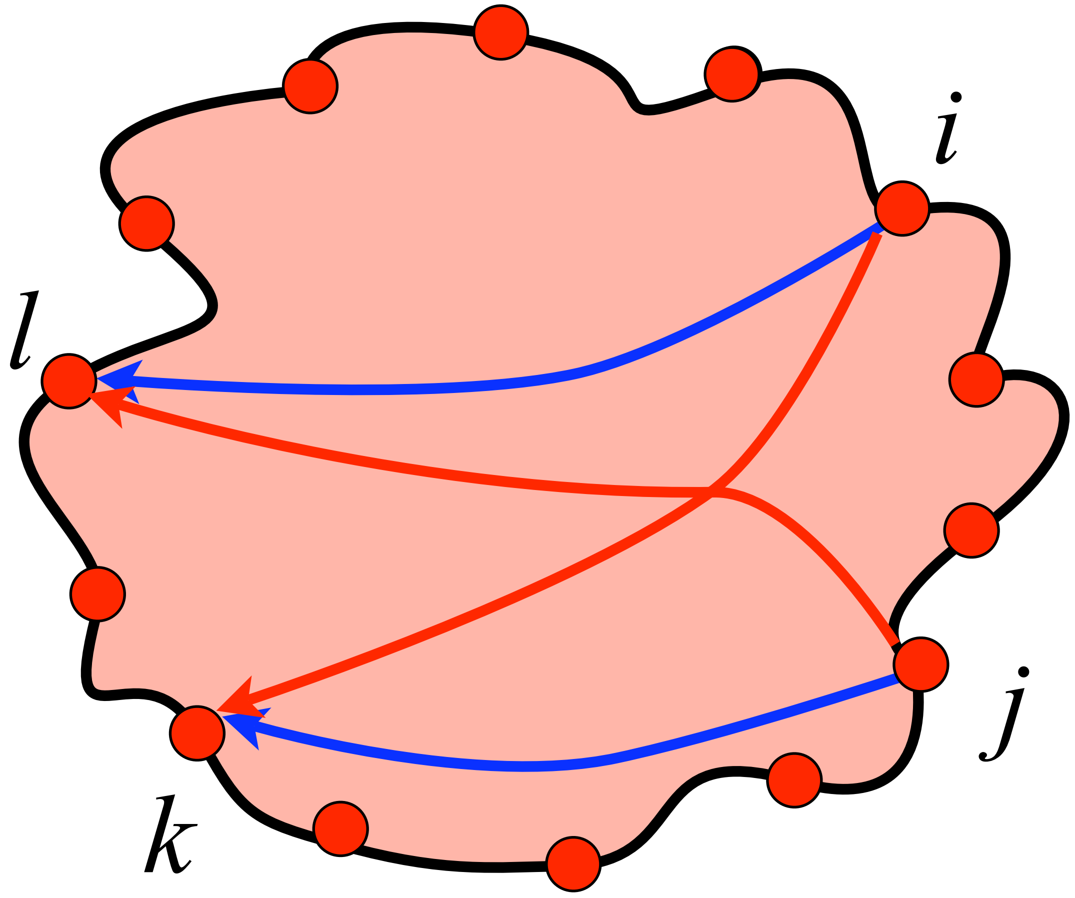




$$\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$$

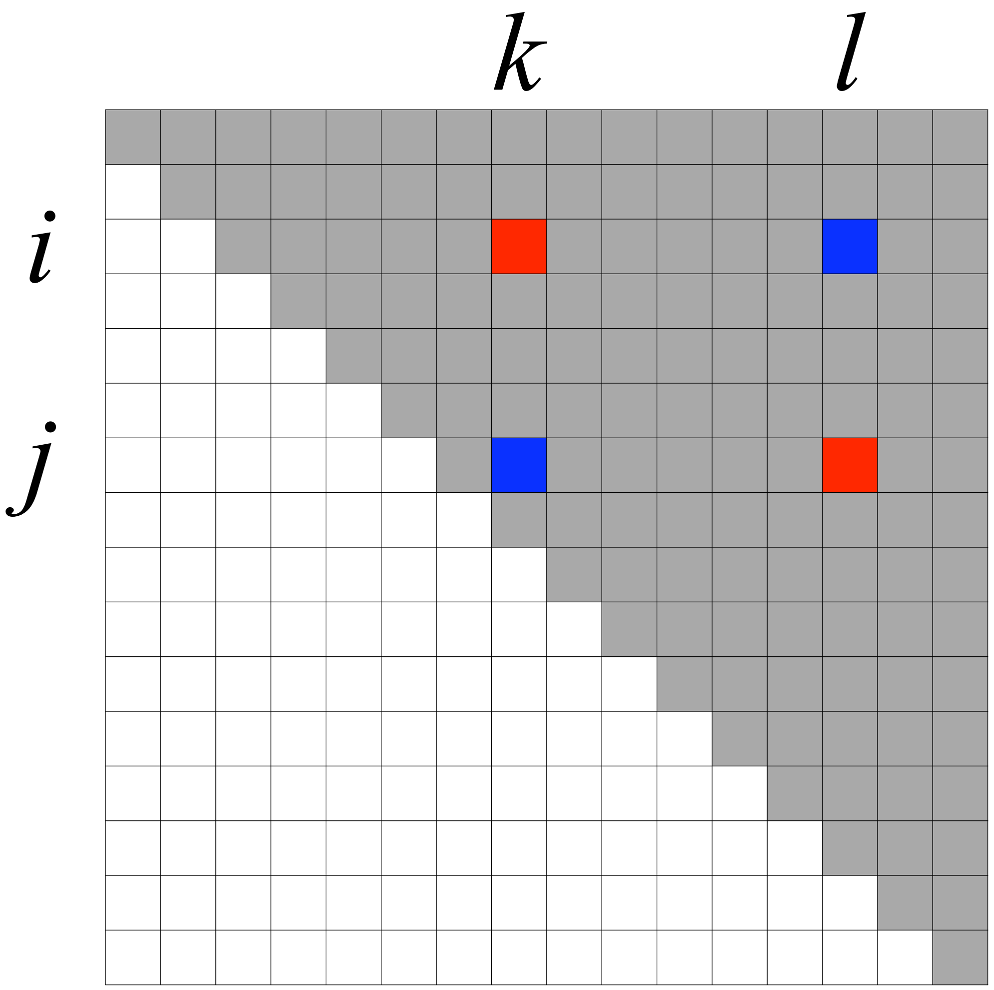
Crossings and the Monge Property

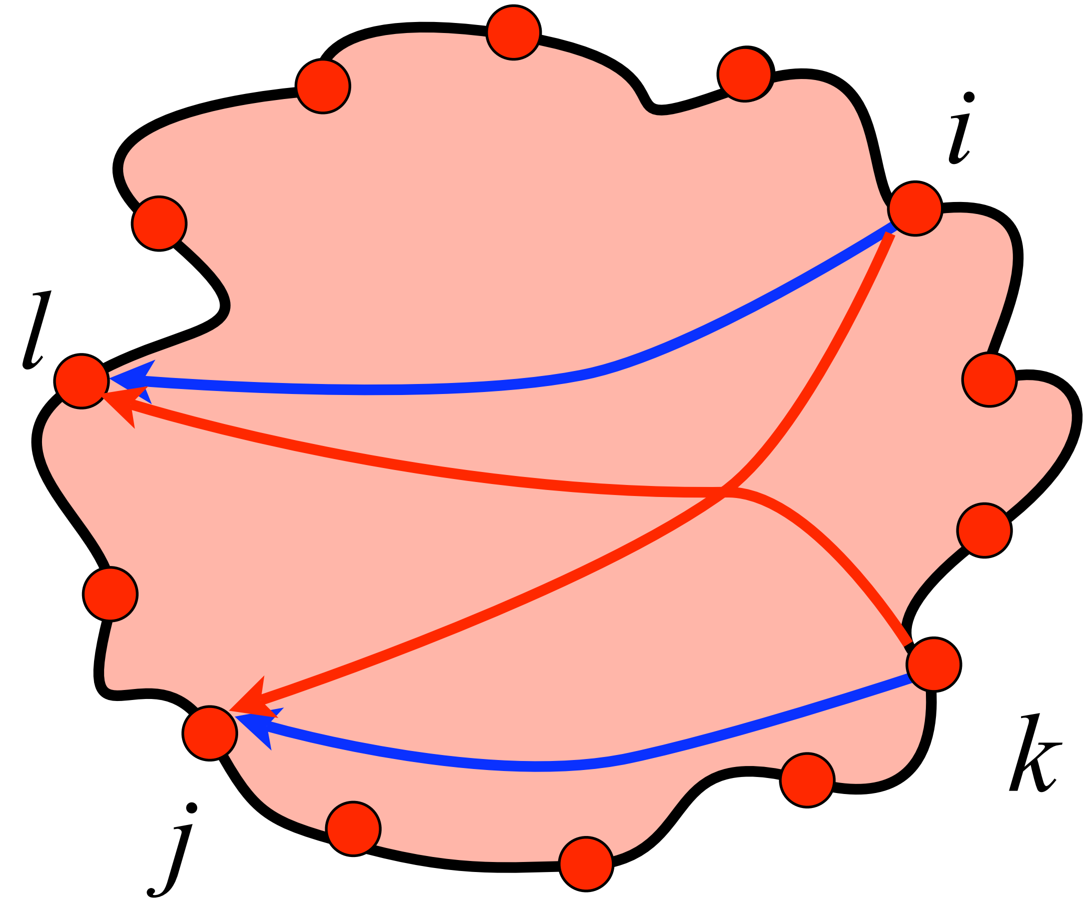




$\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$

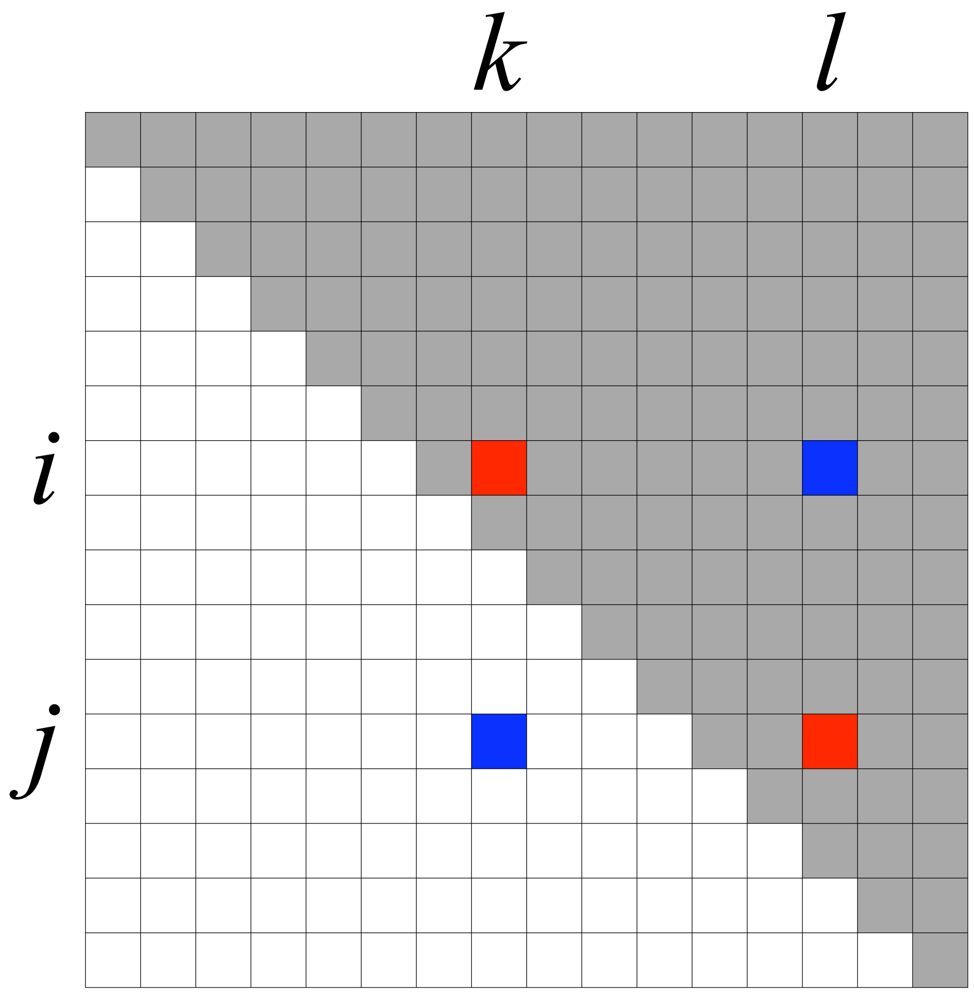
Crossings and the Monge Property

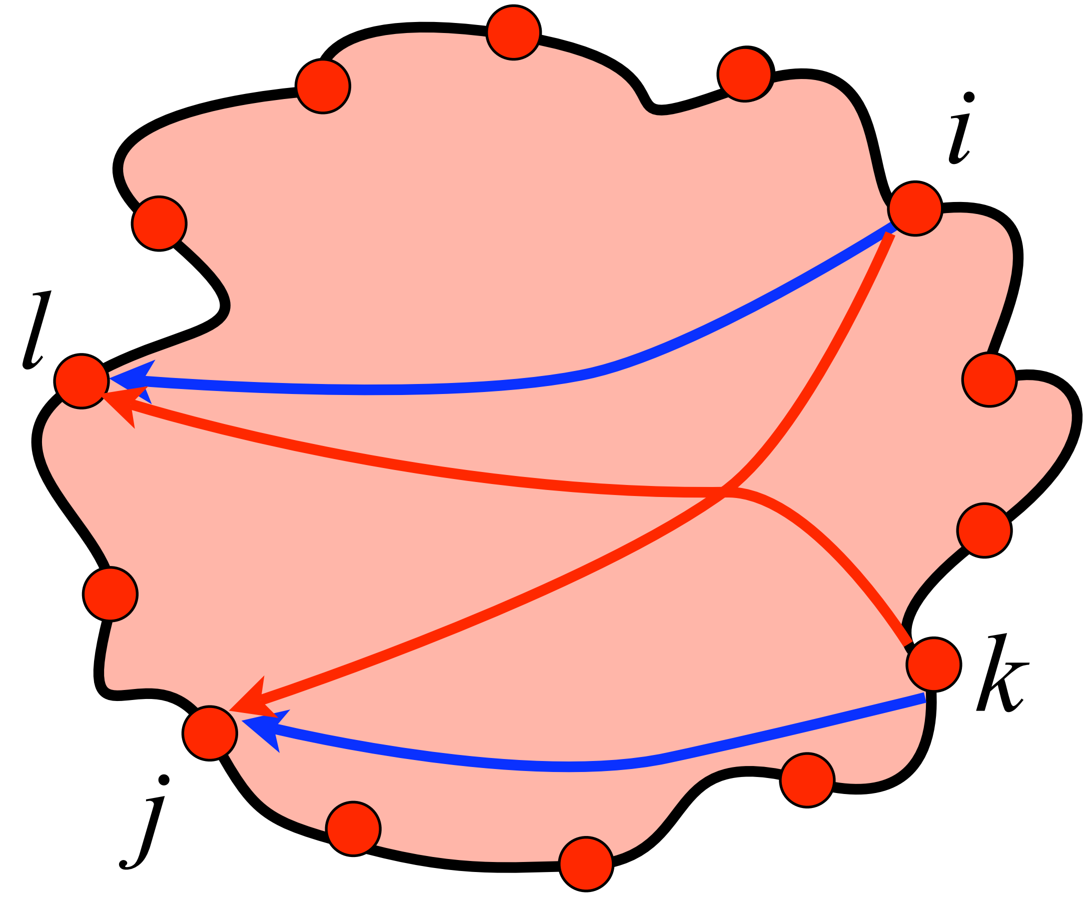




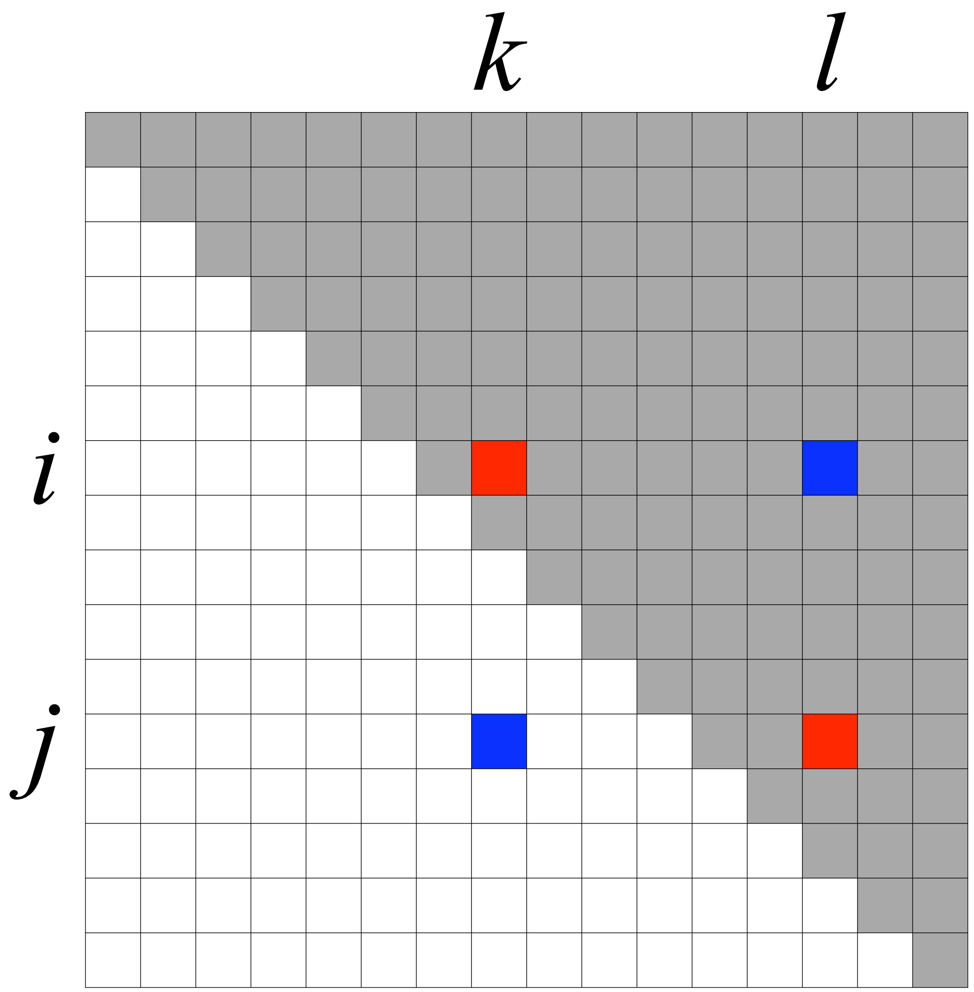
$$\delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k)$$

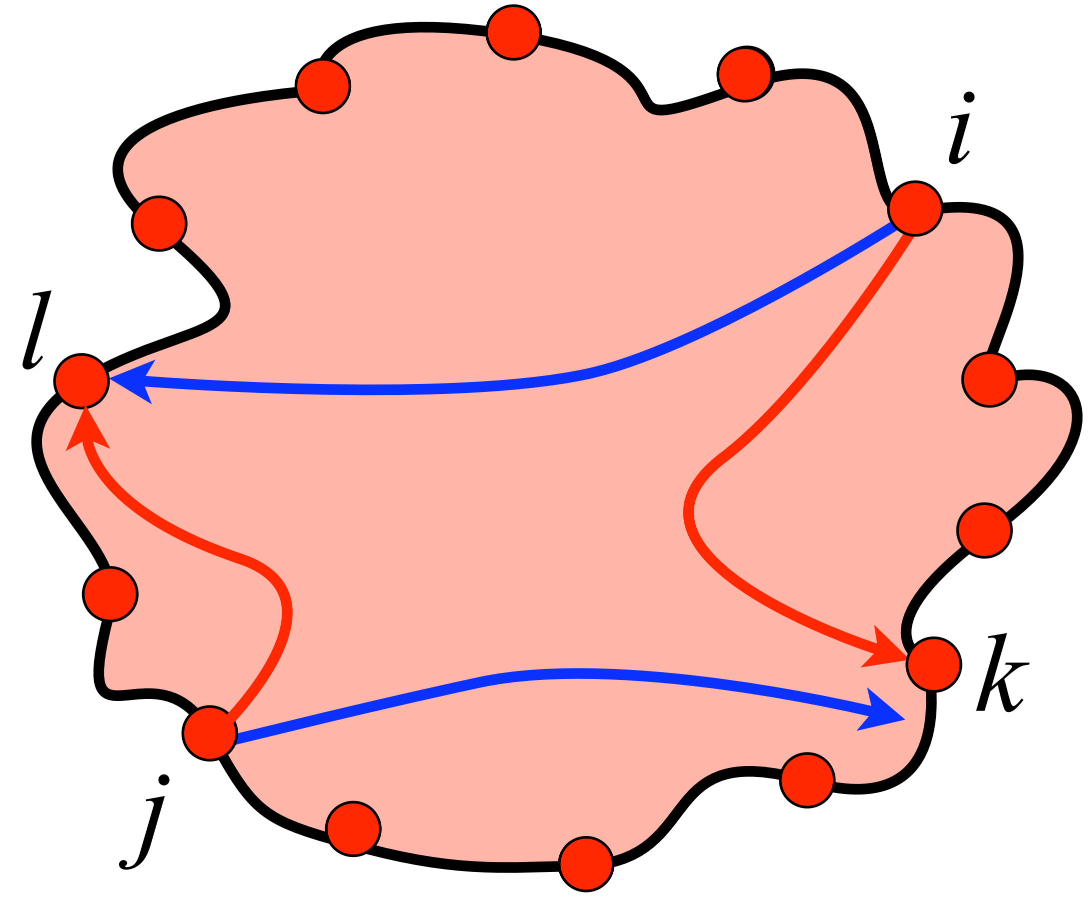
Crossings and the Monge Property



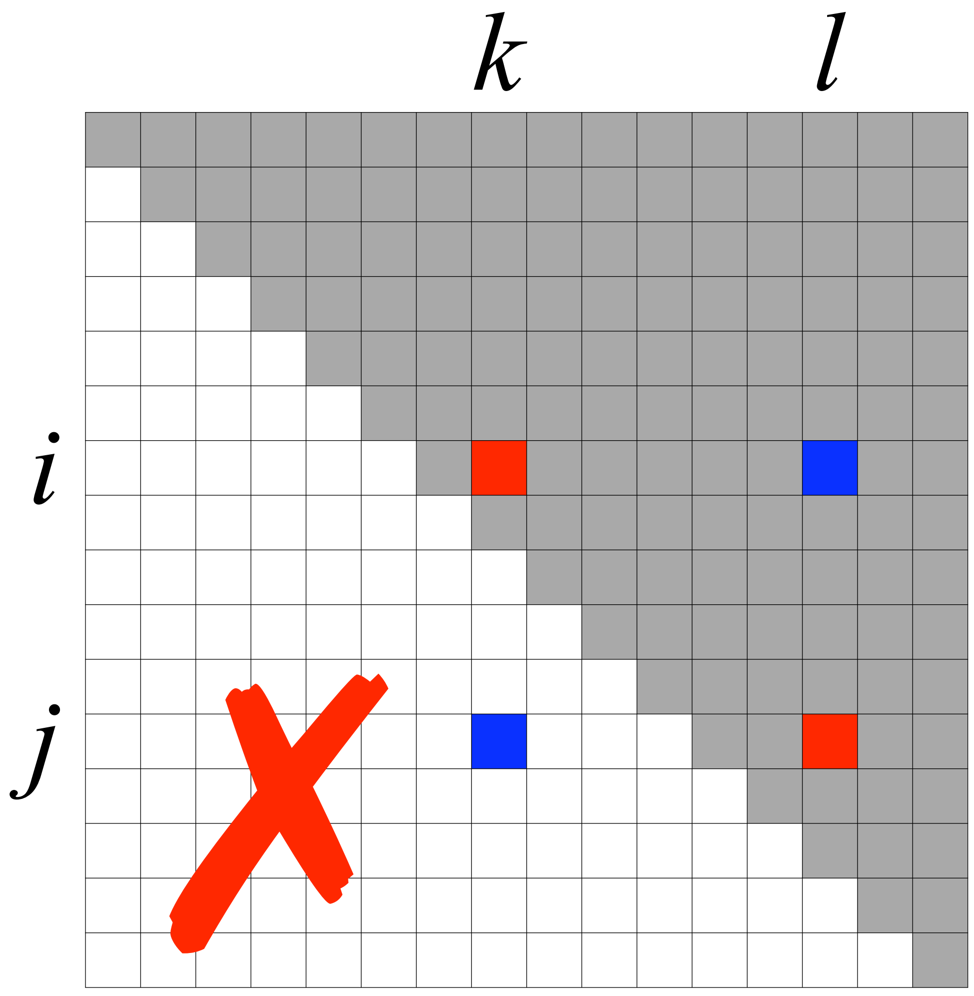


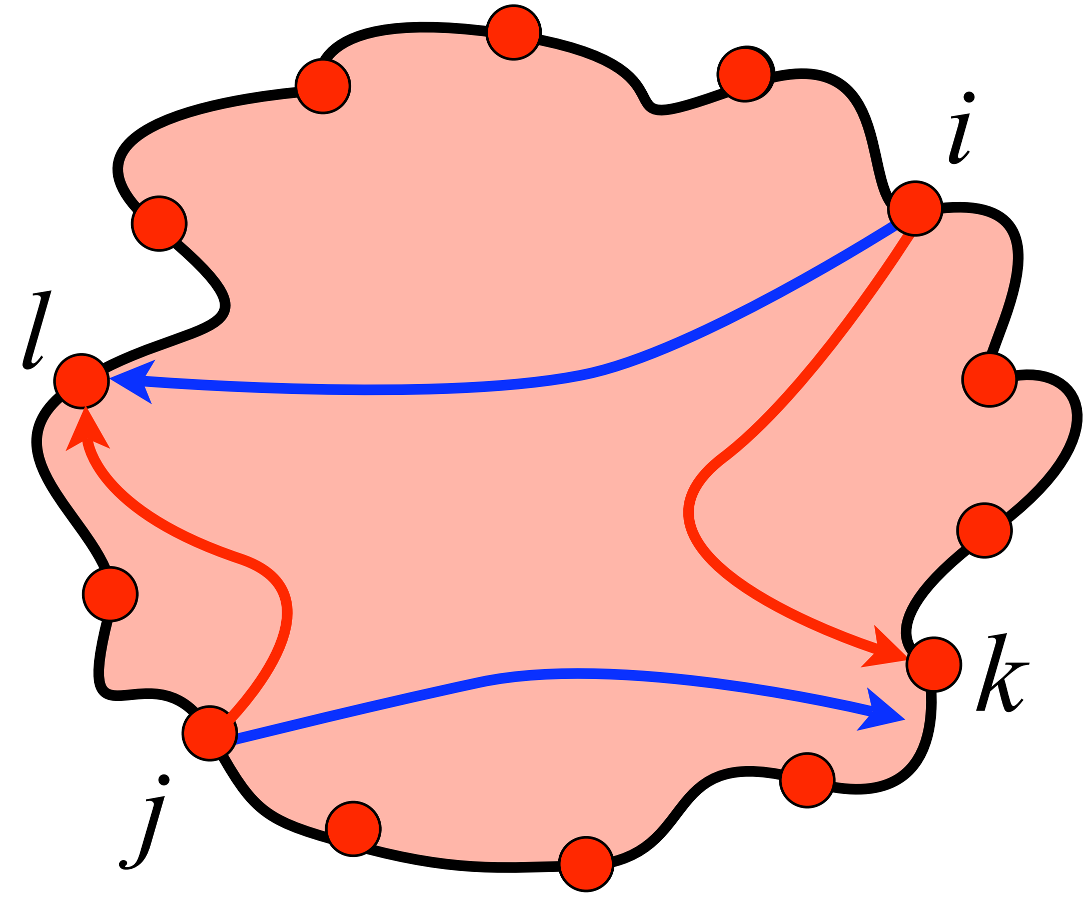
Crossings and the Monge Property





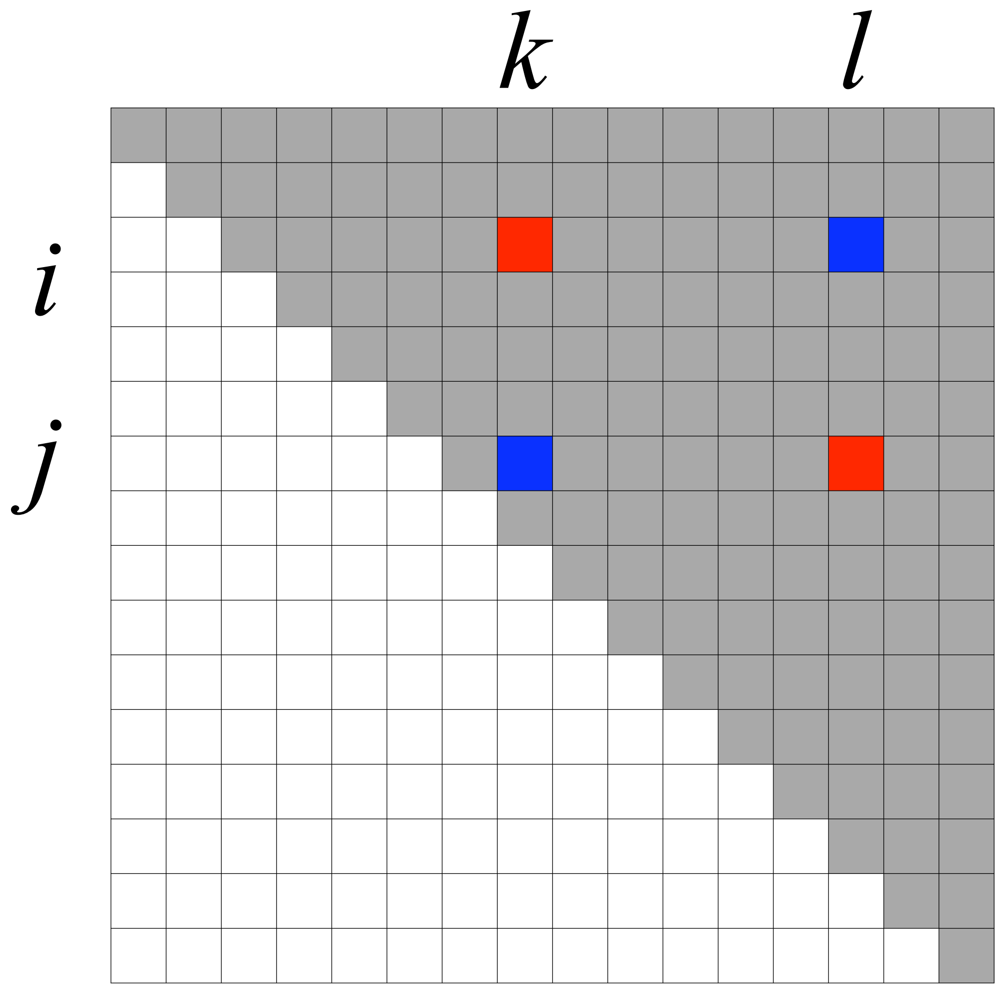
Crossings and the Monge Property





Partial Monge Matrices

- Column Minima of a triangular Monge matrix can be found in $O(n\alpha(n))$ time [Klawe-Kleitman 1990]

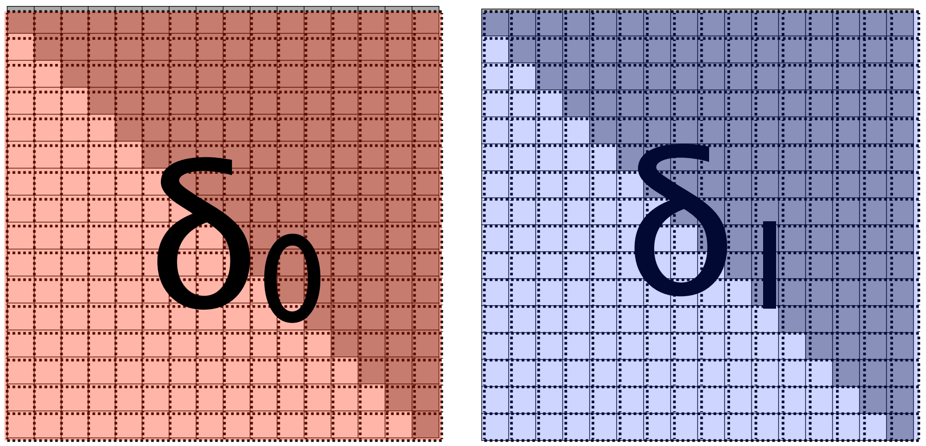


III. r-to-boundary Distances in G

$$\forall v \;\; e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w,v]\}$$

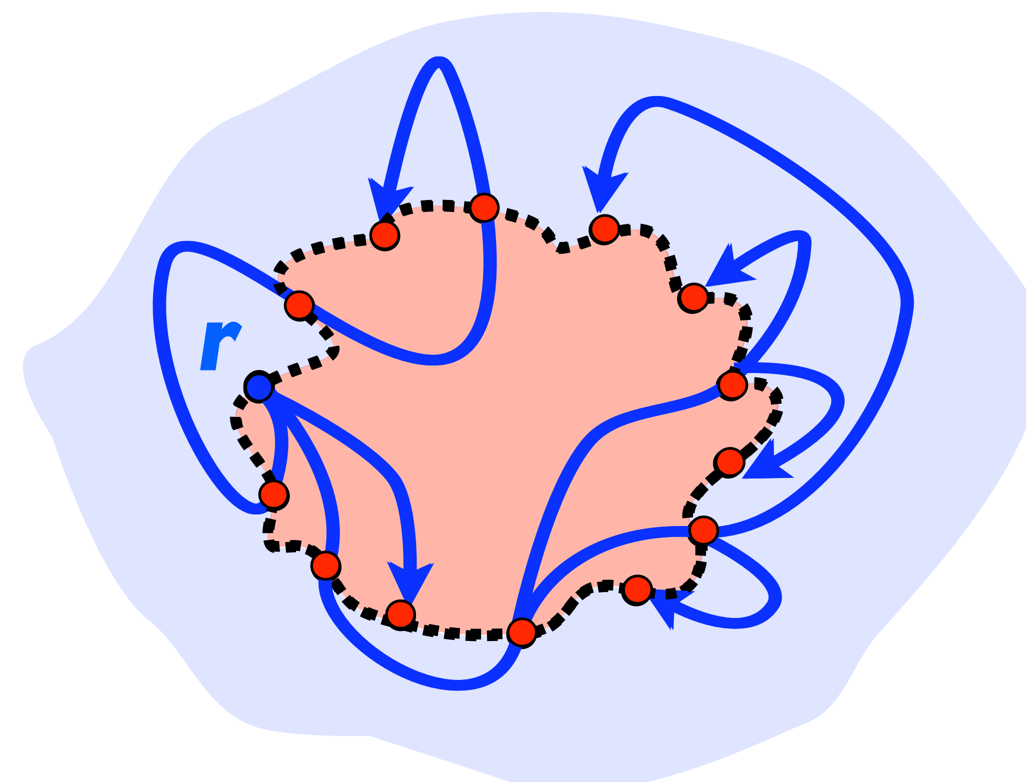
- δ_i is partially Monge even when adding $e_{j\text{-}1}[w]$ to row w
- Each iteration takes $O(\sqrt{n}\alpha(\sqrt{n}))$
- $O\left(\sqrt{n}\right)$ iterations

- All iterations in $O(n\alpha(n))$ time

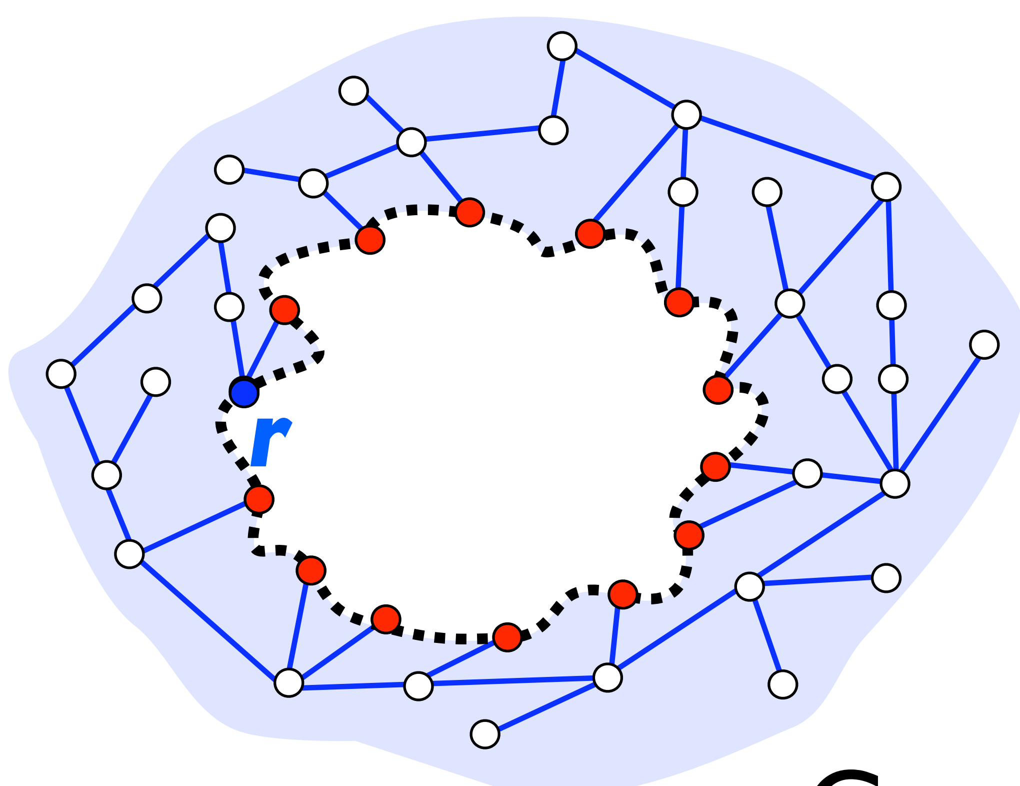


So Far We Have:

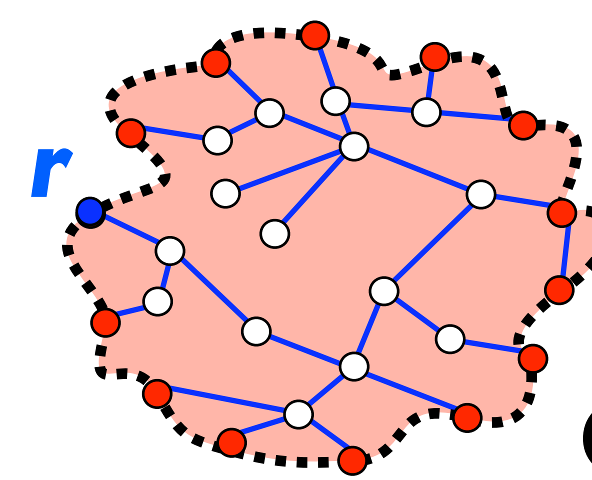
- r-to-boundary distances in G



- r-to-all distances in G_i

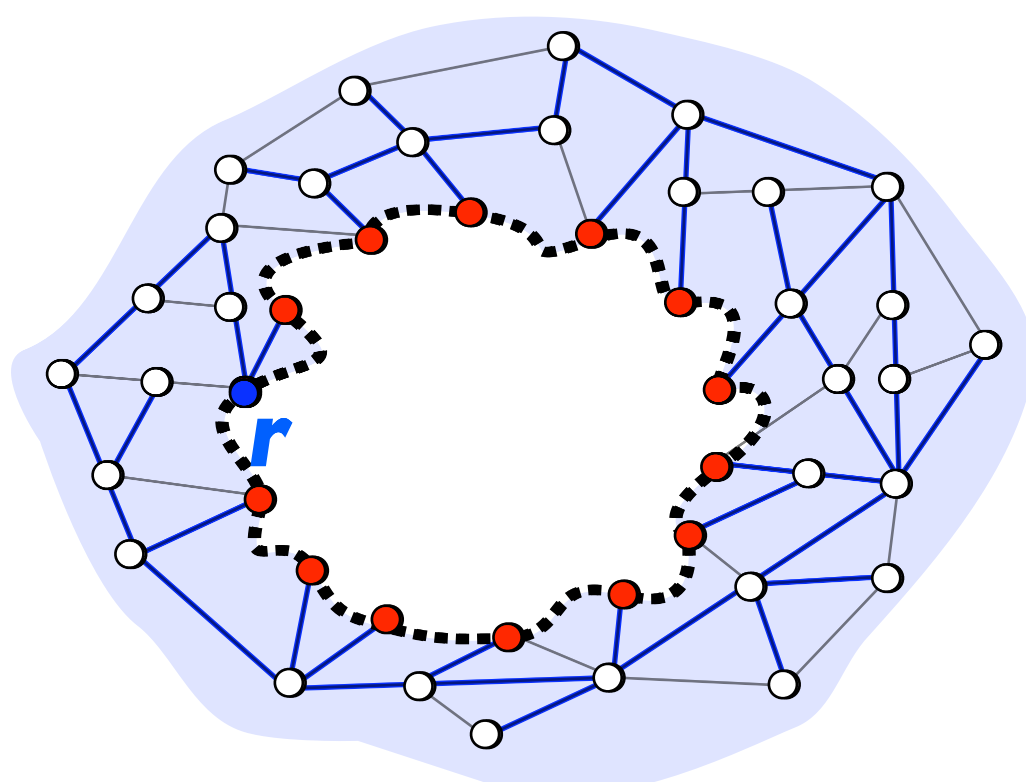


G_1



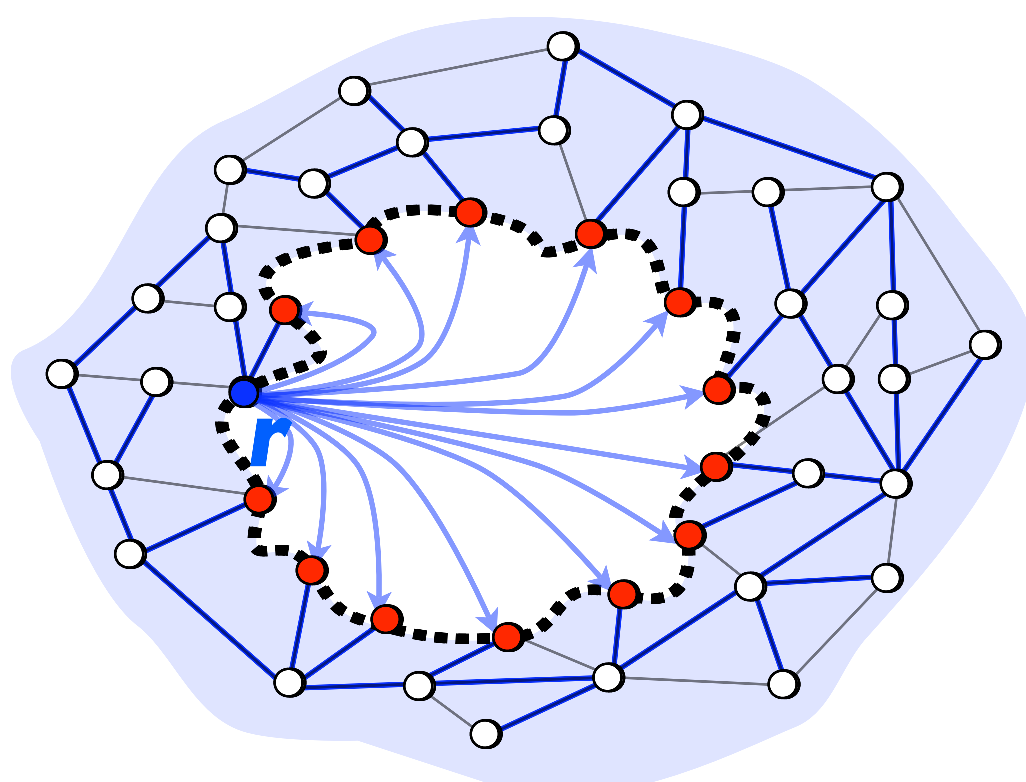
G_0

IV. From-r Distances in G



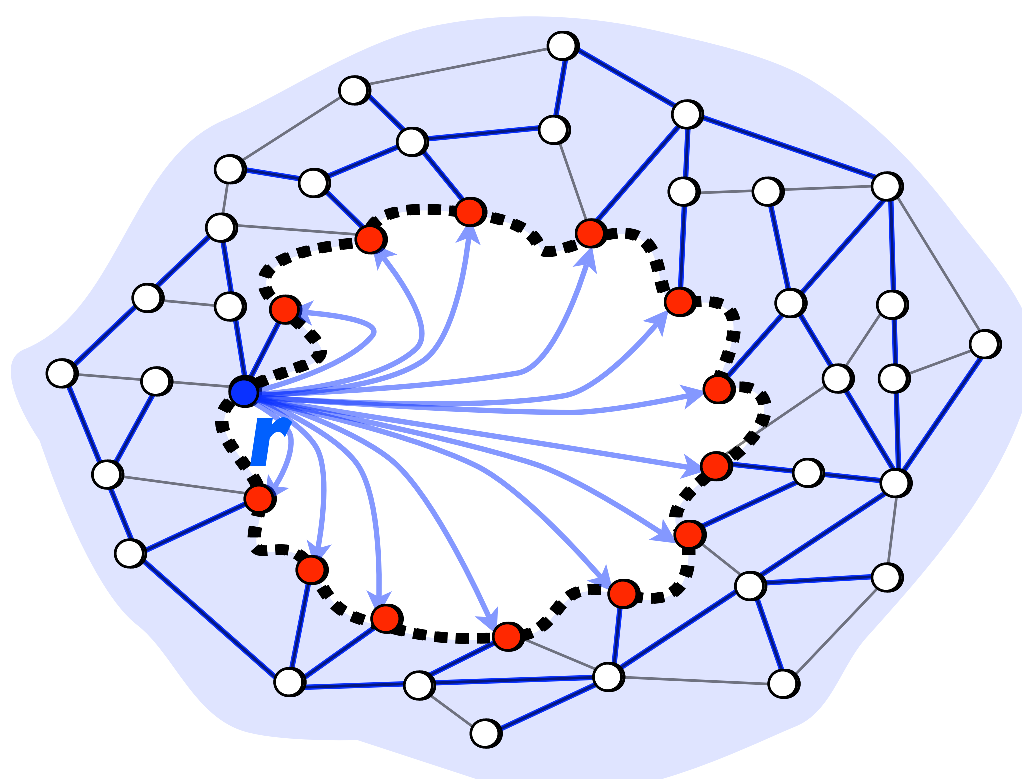
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths
- Distances from r in this graph are equal to distances in G



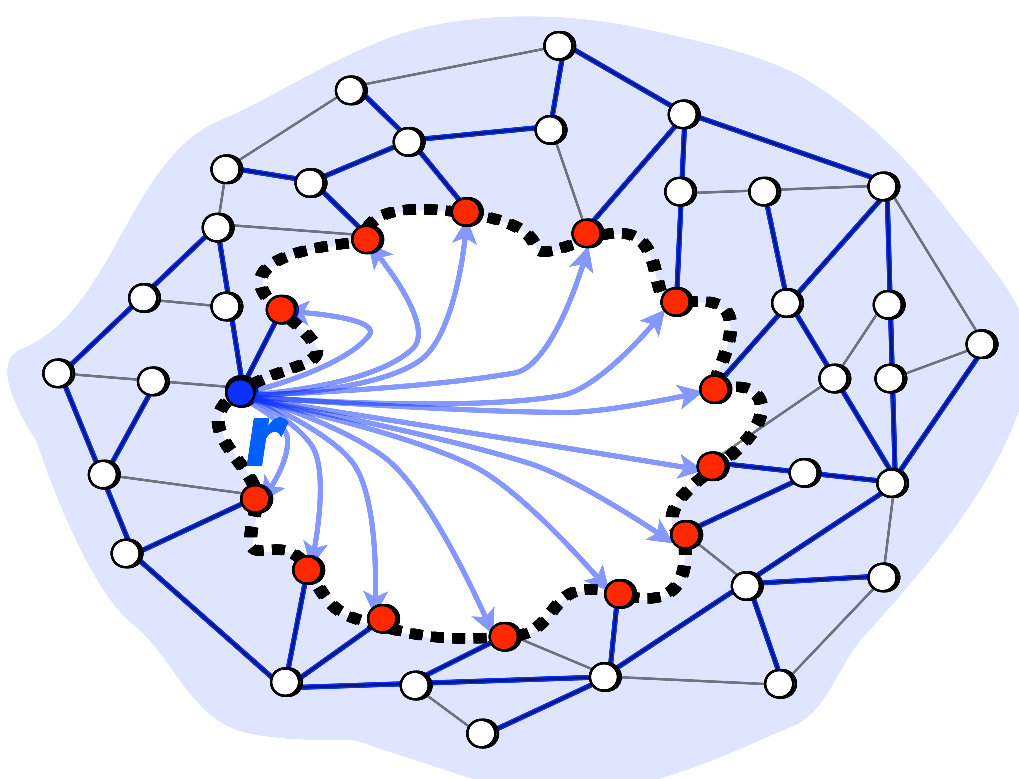
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths
- Distances from r in this graph are equal to distances in G
- Distances from r in G_1 are almost feasible price function



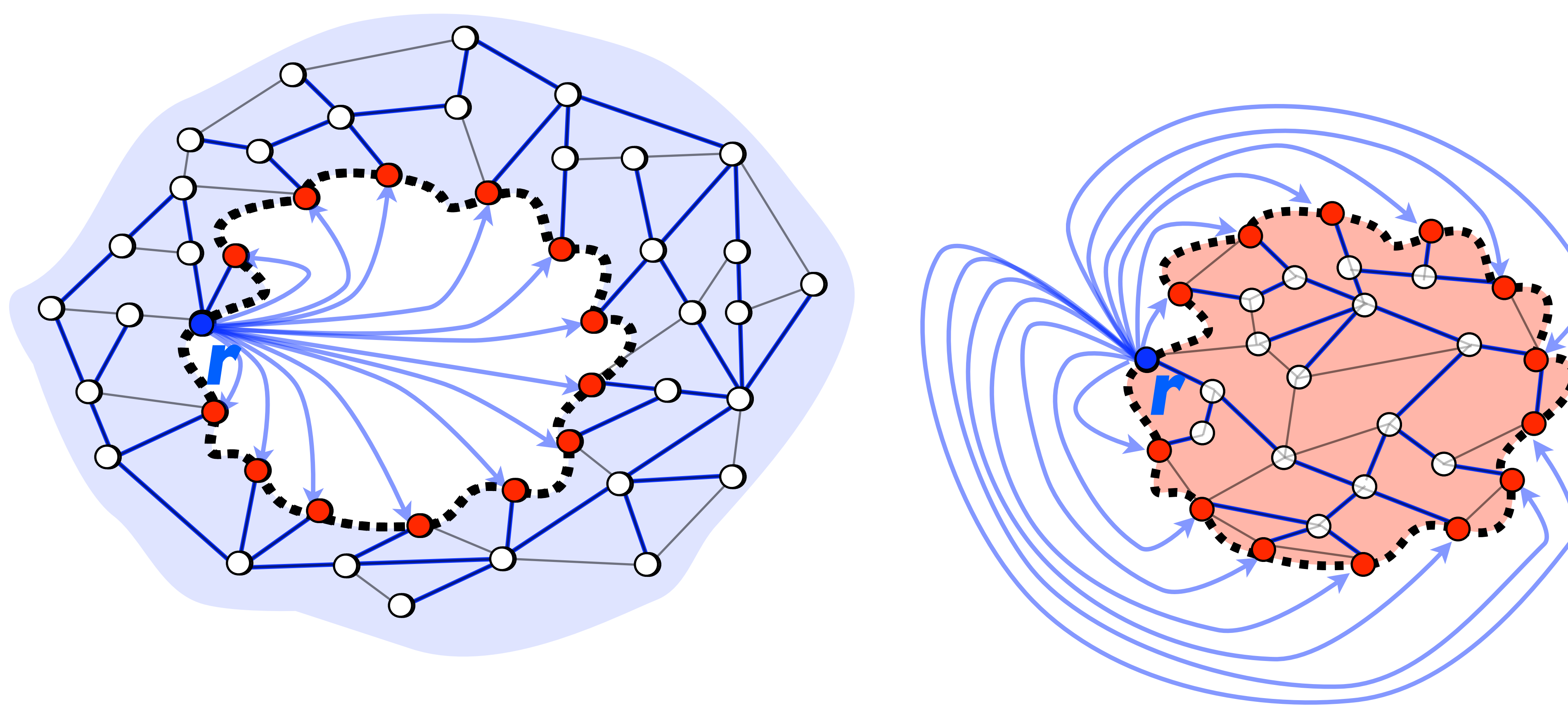
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths
- Distances from r in this graph are equal to distances in G
- Distances from r in G_I are almost feasible price function
- Setting $\varphi(r)$ to a sufficiently large value makes it feasible



IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths
- Distances from r in this graph are equal to distances in G
- Distances from r in G_I are almost feasible price function
- Setting $\varphi(r)$ to a sufficiently large value makes it feasible



Analysis

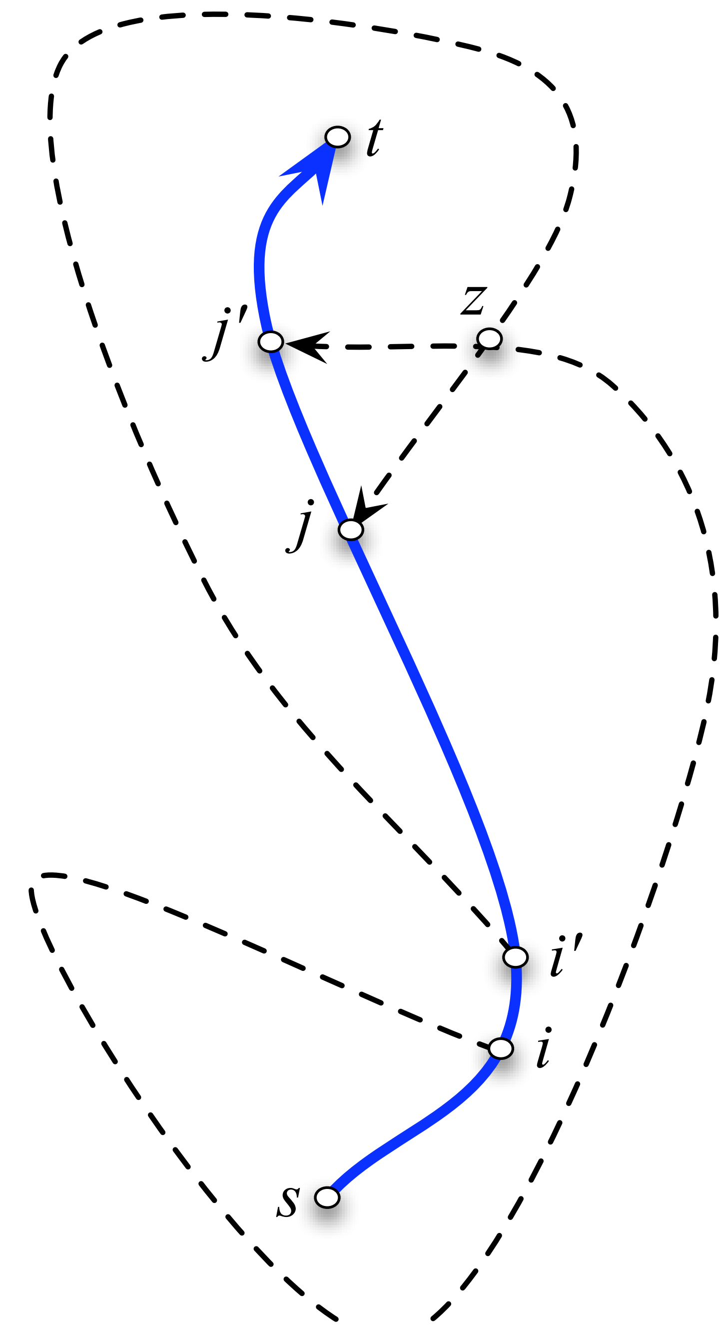
	step	techniques	time
I	recursion	planar separator	
II	boundary to boundary distances in G_i	multiple-source planar shortest paths [Klein 2005]	$\lvert G\rvert \log(\lvert G\rvert)$
III	r-to-boundary distances in G	"Bellman-Ford", partial Monge searching [Klawe-Kleitman 1990]	$\lvert G\rvert \alpha(\lvert G\rvert)$
IV	distances from r in G	augmented graph, feasible price function, Dijkstra	$\lvert G\rvert \log(\lvert G\rvert)$ (can be done in $O(\lvert G\rvert)$)
V	rerooting - distances from s in G	feasible price function, Dijkstra	$\lvert G\rvert \log(\lvert G\rvert)$ (can be done in $O(\lvert G\rvert)$)

$O(\log n)$ levels $\Rightarrow O(n \log^2 n)$ time

$O(n)$ space

Monge in Other Planar Problems

- Use of efficient Monge searching may be applicable in other planar graphs problem
- Example:
  improvement on the running time of an algorithm for the replacement path problem [Emek, Peleg, Roditty SODA08]



Thank You!