Shortest Paths in Directed Planar Graphs with Negative Lengths

Authors: Philip N. Klein, Shay Mozes, Oren Weimann
presented by Sebastian Addicks

19. November 2013

Find an $O(n\ log^2 n)$-time, linear-space algorithm to find shortest paths in planar directed graphs with negative lengths.

requirement: no `negative circles`

Applications

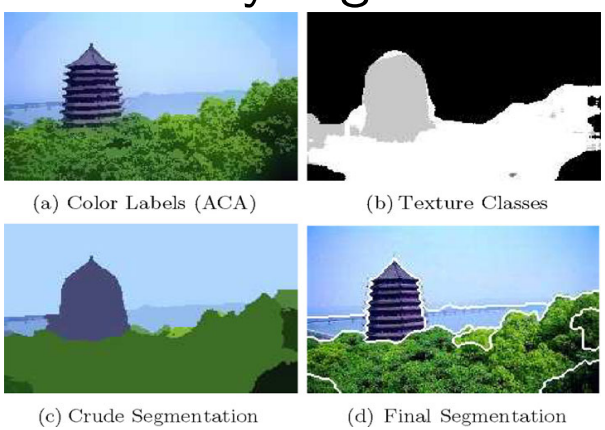
- **perfect matching**
  $\rightarrow$ matching which contains n/2 edges

Applications

- **perfect matching**
  $\rightarrow$ matching which contains n/2 edges
- **stereo matching**
  $\rightarrow$ for a point in image1 find the corresponding one in image2

Applications

- **perfect matching**
  $\rightarrow$ matching which contains n/2 edges
- **stereo matching**
  $\rightarrow$ for a point in image1 find the corresponding one in image2
- **image segmentation**
  $\rightarrow$ identify segments of an image



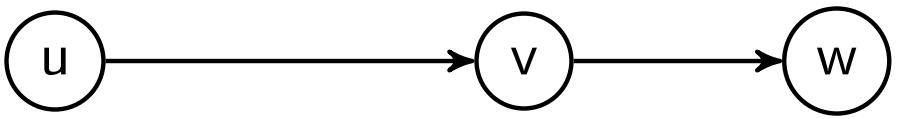
(a) Color Labels (ACA) (b) Texture Classes

(c) Crude Segmentation (d) Final Segmentation

- divide and conquer $\Rightarrow$ separator
- transform the negative lengths $\Rightarrow$ price function

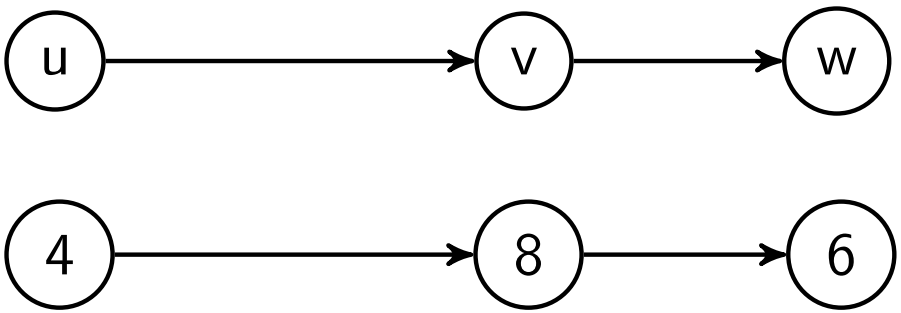
Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



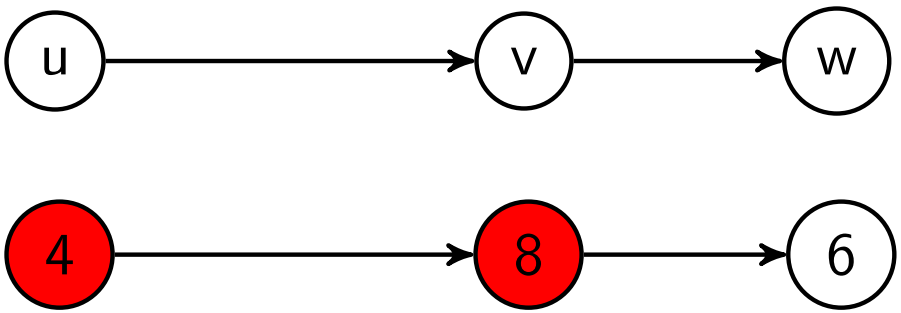
Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



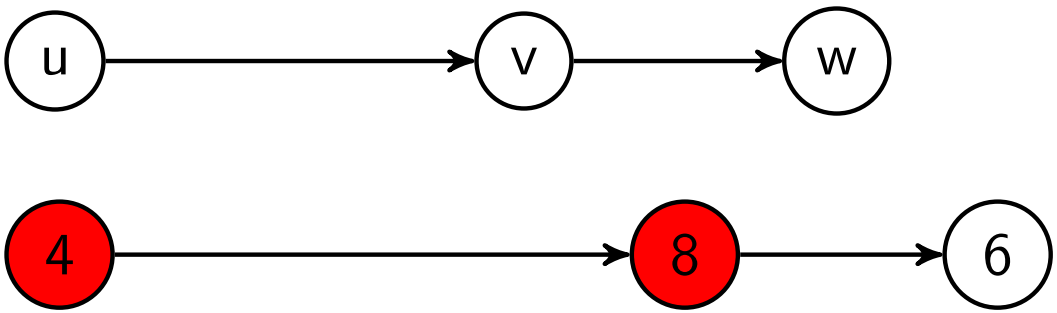
Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



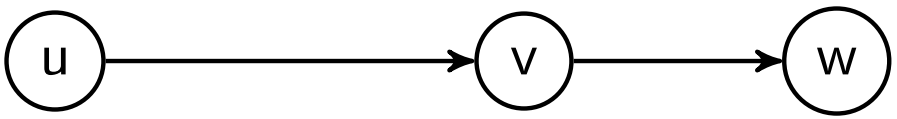
Price Function and Reduced Lengths

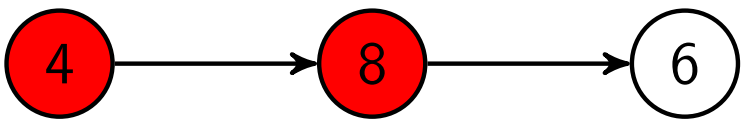
Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



Price Function and Reduced Lengths

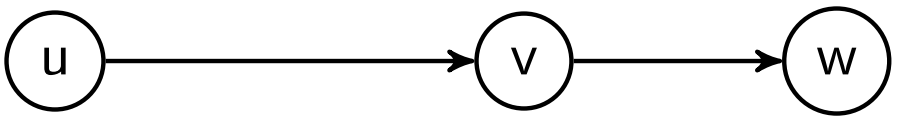
Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

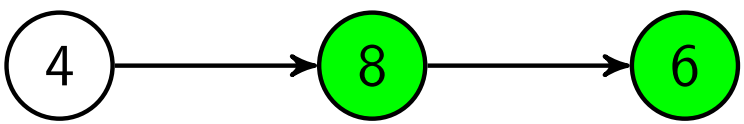




Price Function and Reduced Lengths

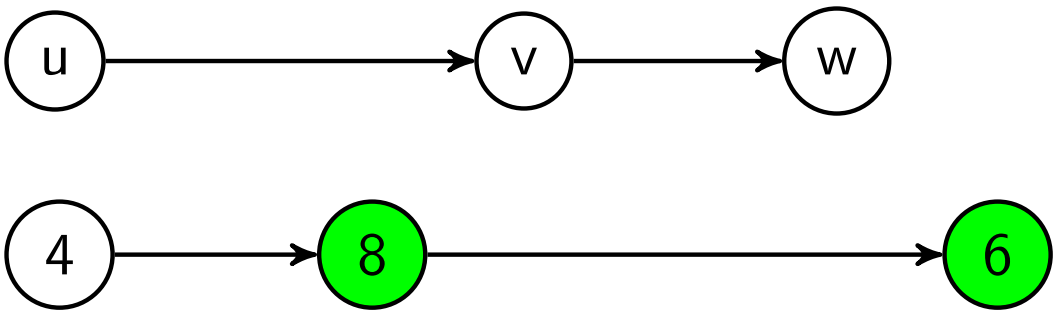
Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$





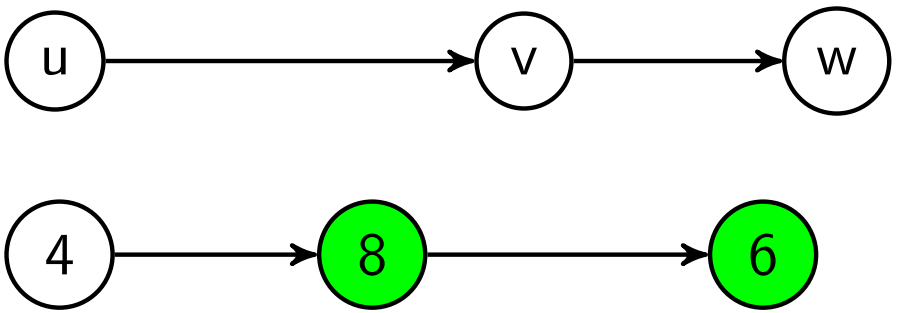
Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



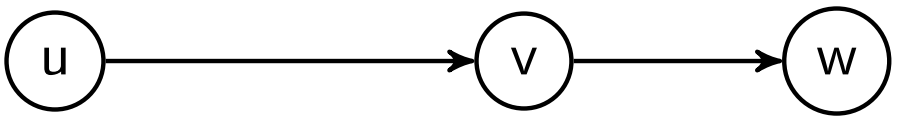
Price Function and Reduced Lengths

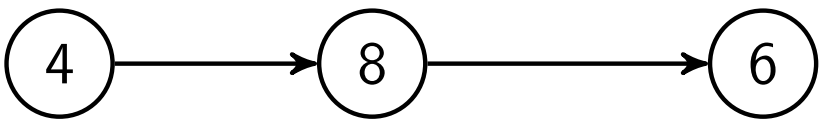
Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$



Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$





Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

- for every s-t-path P $l_\phi(P) = l(P) + \phi(s) - \phi(t)$

Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

- for every s-t-path P $l_\phi(P) = l(P) + \phi(s) - \phi(t)$
- ϕ is `feasible` if l_ϕ is non-negative

Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

- for every s-t-path P $l_\phi(P) = l(P) + \phi(s) - \phi(t)$
- ϕ is `feasible` if l_ϕ is non-negative
- if ϕ is `feasible` it eliminates negative lengths

Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

- for every s-t-path P $l_\phi(P) = l(P) + \phi(s) - \phi(t)$
- ϕ is `feasible` if l_ϕ is non-negative
- if ϕ is `feasible` it eliminates negative lengths

$\Longrightarrow \phi$ reserves shortest paths

Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

Use single source distances from any arbitrary node `w` as `feasible` price function:

Price Function and Reduced Lengths

Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

Use single source distances from any arbitrary node `w` as `feasible` price function:

$$\text{shortest path inequality: } \phi(v) \leq \phi(u) + l(uv)$$

Price Function and Reduced Lengths

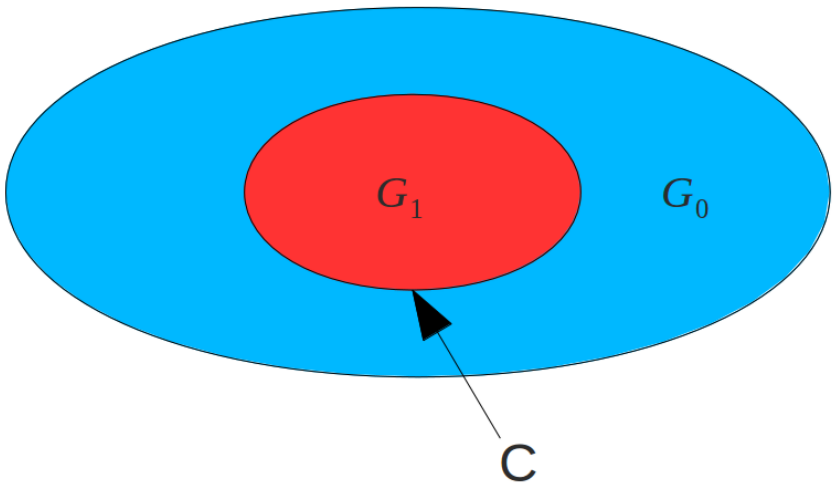
Price function ϕ to eliminate negative lengths, $\phi : N \longrightarrow \mathbb{R}$
$l_\phi(uv) = l(uv) + \phi(u) - \phi(v)$

Use single source distances from any arbitrary node `w` as `feasible` price function:

$$\text{shortest path inequality: } \phi(v) \leq \phi(u) + l(uv)$$

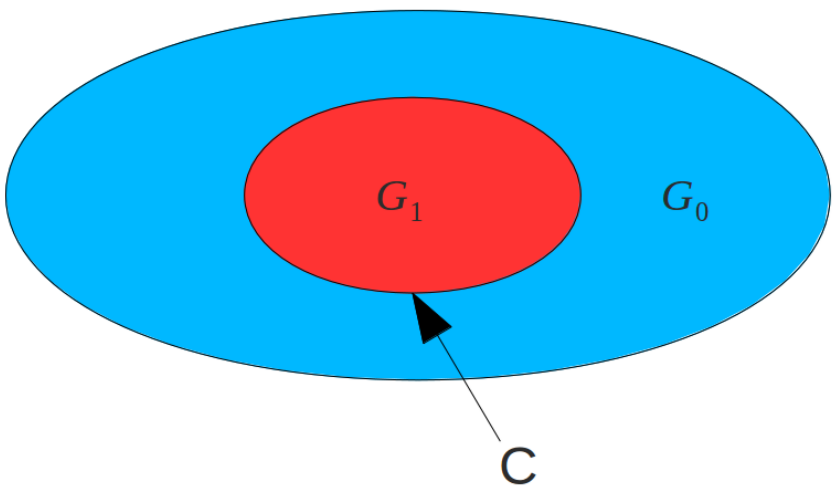
$$\Longrightarrow l_\phi(uv) = l(uv) + \phi(u) - \phi(v) \geq 0$$

Separator in a n-node Graph G:



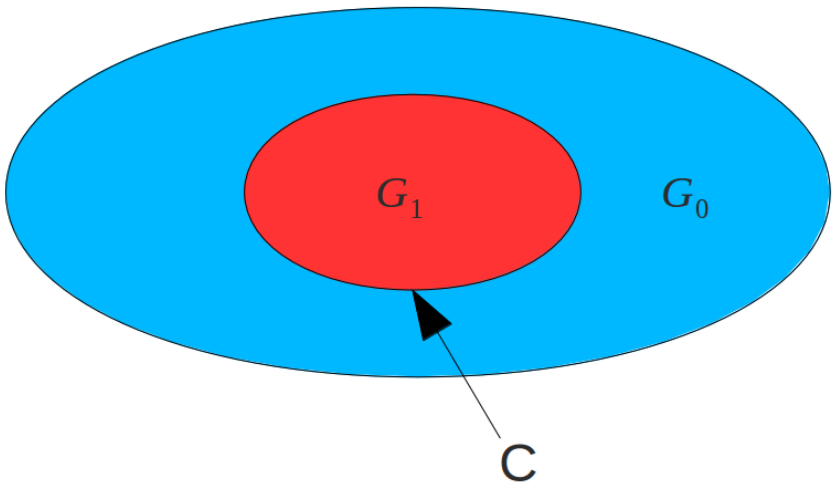
Separator in a n-node Graph G:

- divides G in two parts G_0 and G_1



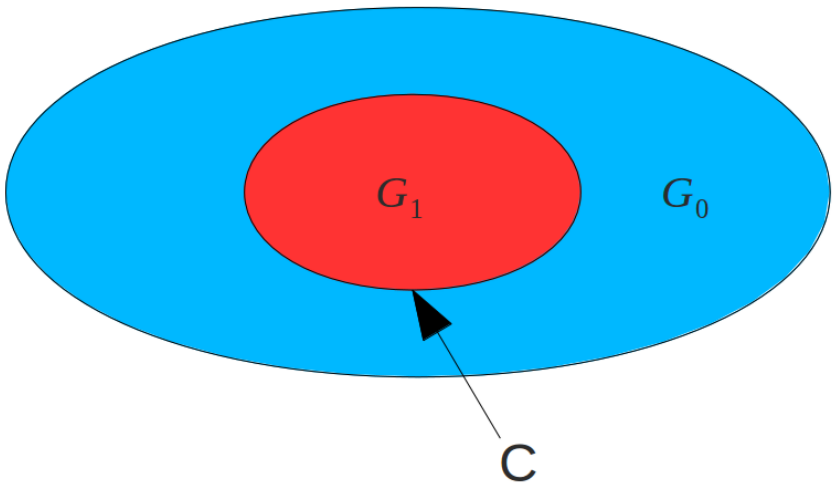
Separator in a n-node Graph G:

- divides G in two parts G_0 and G_1
- G_0 and G_1 have at most 2n/3 nodes



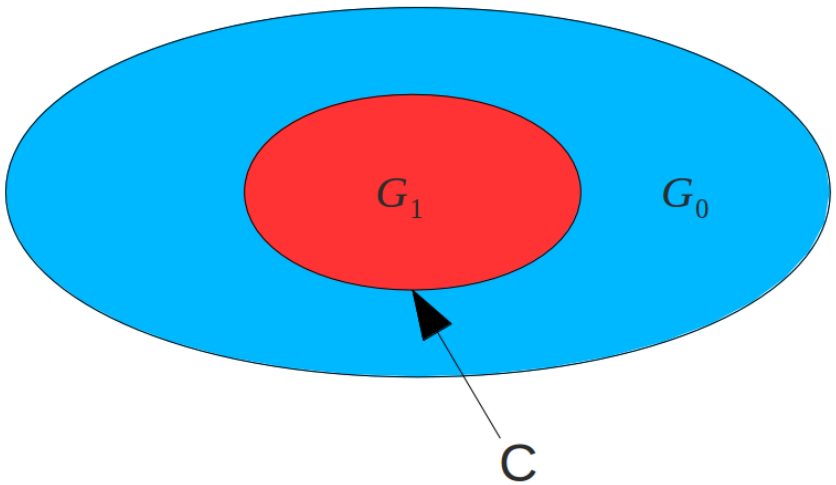
Separator in a n-node Graph G:

- divides G in two parts G_0 and G_1
- G_0 and G_1 have at most 2n/3 nodes
- separator C consists of $\sqrt{n}$ nodes



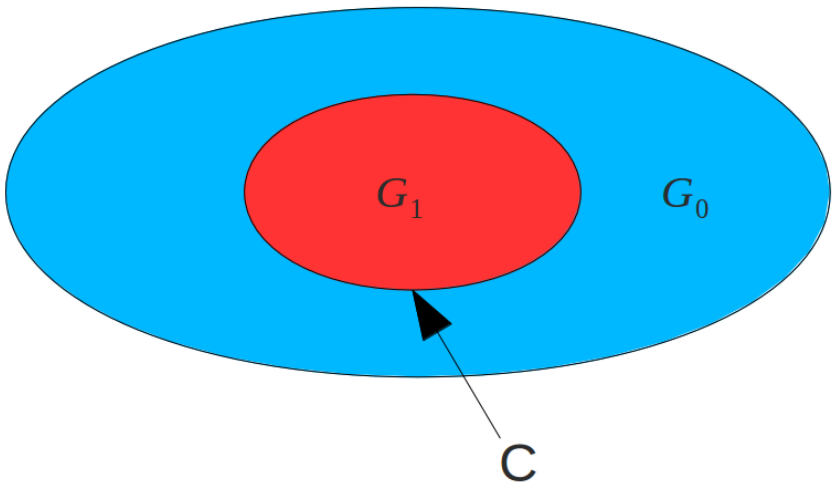
Separator in a n-node Graph G:

- divides G in two parts G_0 and G_1
- G_0 and G_1 have at most 2n/3 nodes
- separator C consists of $\sqrt{n}$ nodes
- there is no edge which starts in $G_0(G_1)$ and ends in $G_1(G_0)$



Separator in a n-node Graph G:

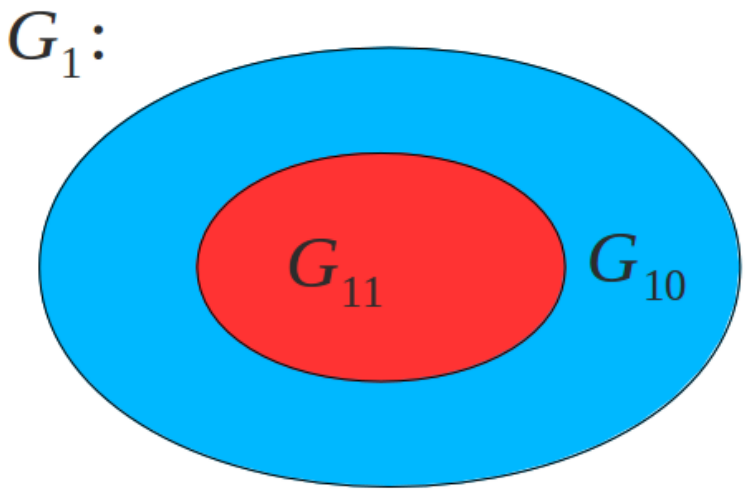
- divides G in two parts G_0 and G_1
- G_0 and G_1 have at most 2n/3 nodes
- separator C consists of $\sqrt{n}$ nodes
- there is no edge which starts in $G_0(G_1)$ and ends in $G_1(G_0)$

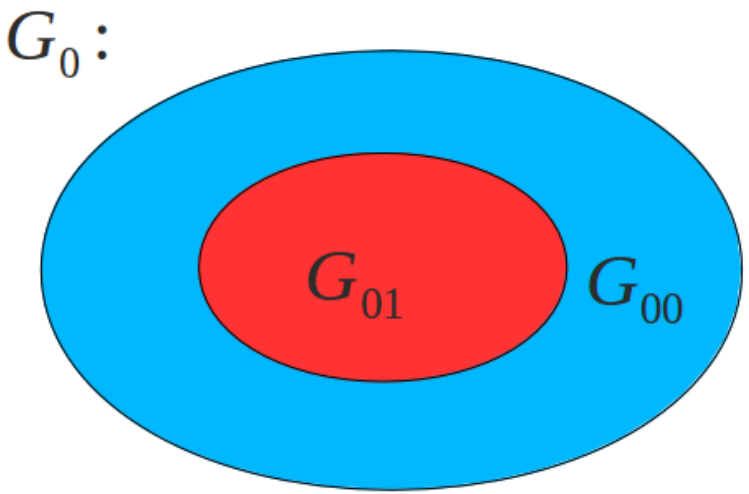


G_1 and G_0 denote the internal and external part of G with respect to C

1. initialization $\longrightarrow$ O(n)
   - find a jordan separator C
   - let G_1 and G_0 be the internal an external part of G with respect to C
   - let r be an arbitrary boundary node

1. initialization $\longrightarrow$ O(n)
   - find a jordan separator C
   - let G_1 and G_0 be the internal an external part of G with respect to C
   - let r be an arbitrary boundary node
2. recursive call
   - for $i = 0, 1$: let d_i shortest-paths(G_i, r)





3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

MSSP:

input:

- directed planar embedded Graph G with non-negative arc lengths
- a face f

3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

MSSP:

input:

- directed planar embedded Graph G with non-negative arc lengths
- a face f
- a table of distances to all nodes from some node on the face f

3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

MSSP:

input:

- directed planar embedded Graph G
- a face f
- a table of distances to all nodes from some node on the face f

3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

MSSP:

input:

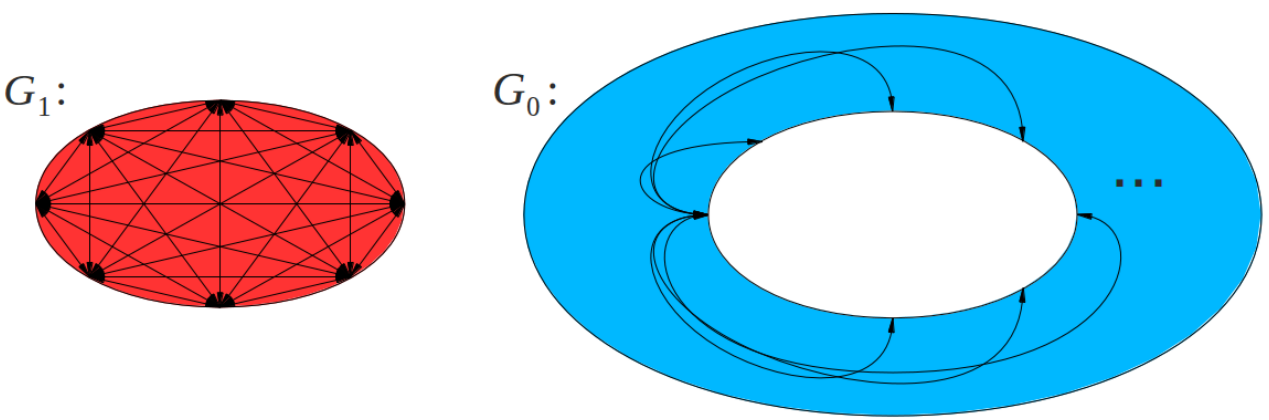
- directed planar embedded Graph G
- a face f
- a table of distances to all nodes from some node on the face f

output:

- the u-to-v distances in G for every pair uv of boundary nodes
- stored in δ_1 and δ_0 such that $\delta_i[uv]$ is the u-to-v distance in G_i

3. intra-part boundary distances $\longrightarrow O(n\ log(n))$
   - multiple-source shortest-path (MSSP) algorithm
   - used on G_1 and G_0

   output:
   - the u-to-v distances in G for every pair uv of boundary nodes
   - stored in δ_1 and δ_0 such that $\delta_i[uv]$ is the u-to-v distance in G_i



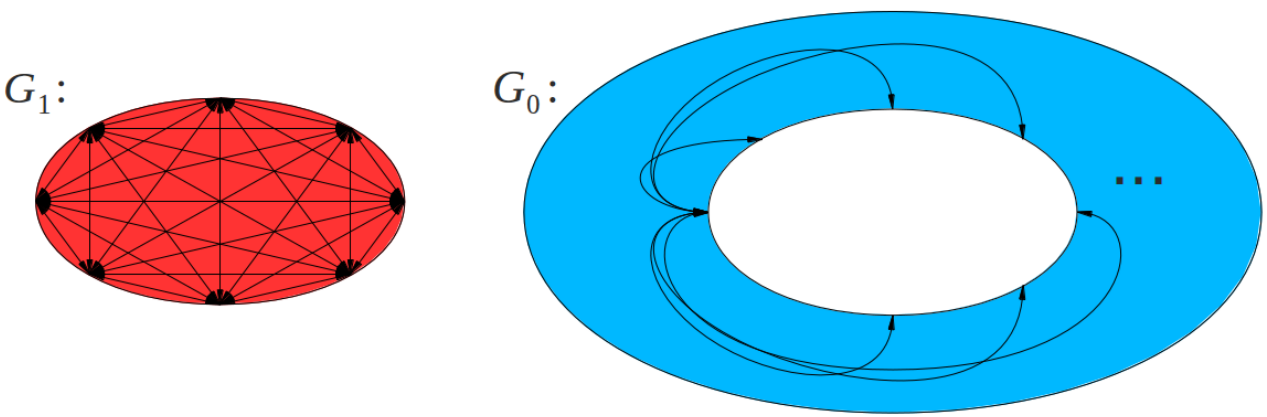
4. single-source inter-part boundary distances
   $\longrightarrow O(n^{3/2})$
   - use δ_1, δ_0 to compute a table B such that $B[v]$ is the r-to-v distance in G for every boundary node

4. single-source inter-part boundary distances
   $\longrightarrow O(n^{3/2})$
   - use δ_1, δ_0 to compute a table B such that $B[v]$ is the r-to-v distance in G for every boundary node

   Bellman Ford
   - using just boundary to boundary distances
   - $\sqrt{n}$ iterations
   - with n edges

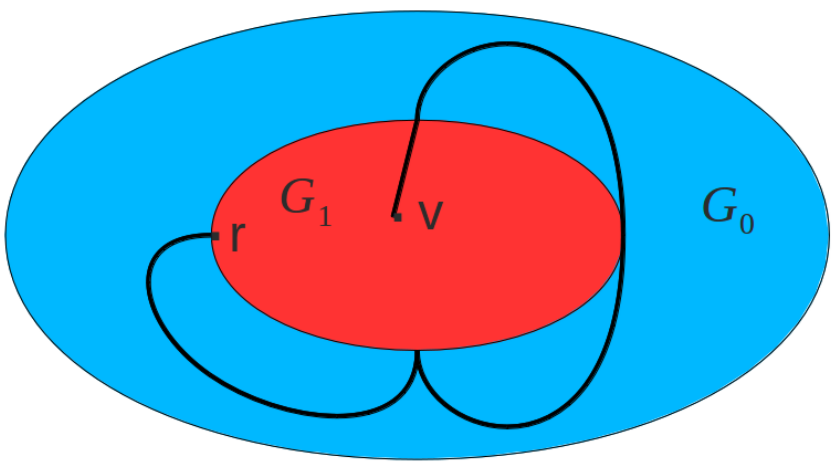
4. single-source inter-part boundary distances
   $\longrightarrow O(n^{3/2})$
   - use δ_1, δ_0 to compute a table B such that $B[v]$ is the r-to-v distance in G for every boundary node

   Bellman Ford
   - using just boundary to boundary distances
   - $\sqrt{n}$ iterations
   - with n edges



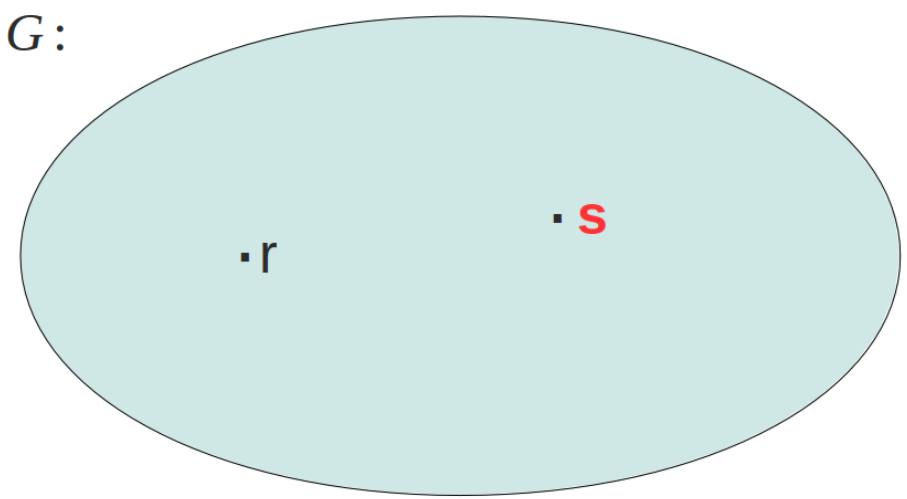
5. single-source inter part distances
   - use d_i and B and Dijkstra's Algorithm to compute d'_i such that $d'_i[v]$ are the r-to-v distances in G for every node v in G_i

5. single-source inter part distances
   - use d_i and B and Dijkstra's Algorithm to compute d'_i such that $d'_i[v]$ are the r-to-v distances in G for every node v in G_i



6. rerooting single-source distances
   - define a price function ϕ with the distances from r
   - use Dijkstra's algorithm with ϕ to compute a table d such that $d[v]$ is the s-to-v distance in G for every v

6. rerooting single-source distances
   - define a price function ϕ with the distances from r
   - use Dijkstra's algorithm with ϕ to compute a table d such that $d[v]$ is the s-to-v distance in G for every v



1. initialization $\longrightarrow O(n)$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n^{3/2})$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n^{3/2})$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n^{3/2})$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$
6. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

---

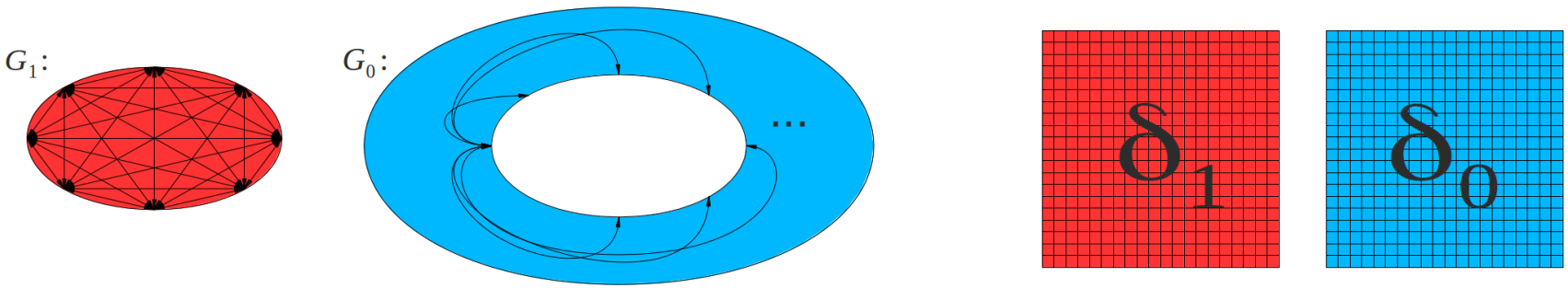
1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n^{3/2})$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$
6. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

---

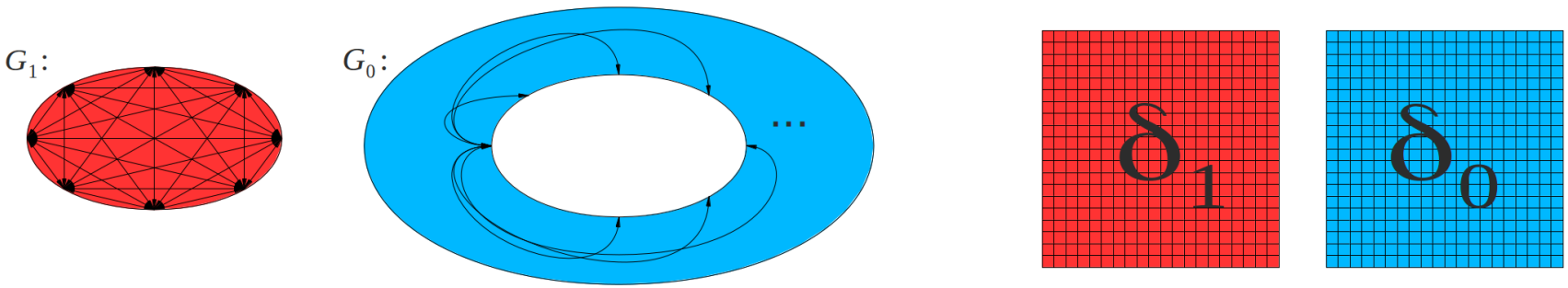
step 5 and 6 can be done in $O(n)$

to get $O(n\ log^2 n)$ as overall time improve step 4

4. single-source inter-part boundary distances

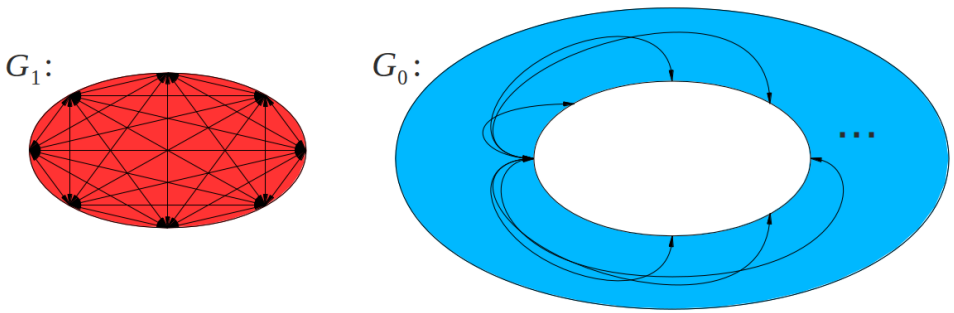


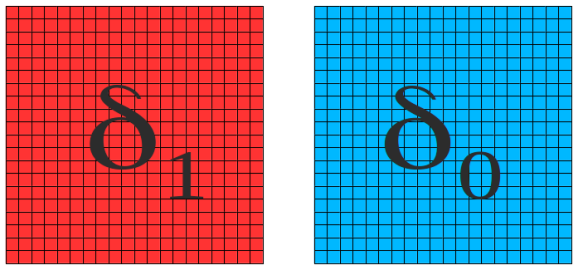
4. single-source inter-part boundary distances



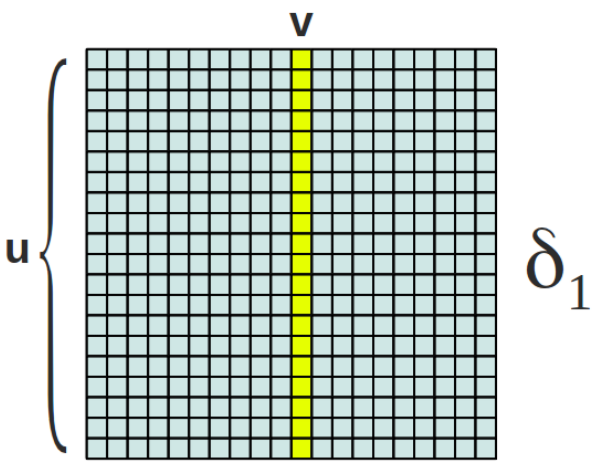
$$\forall v \in C \; d(v) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$

4. single-source inter-part boundary distances

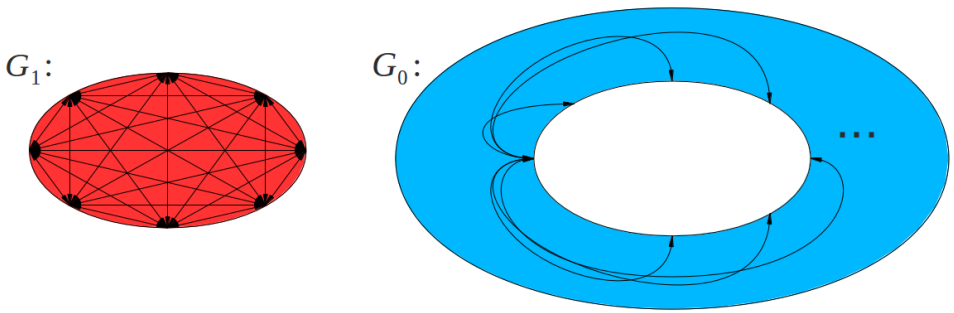


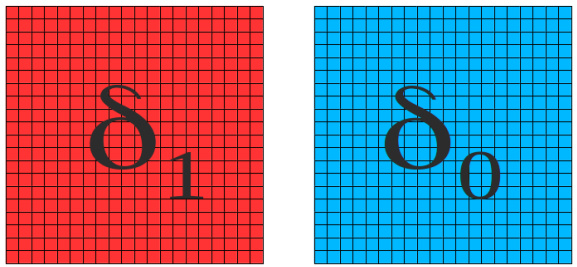


$$\forall v \in C \; d(v) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$

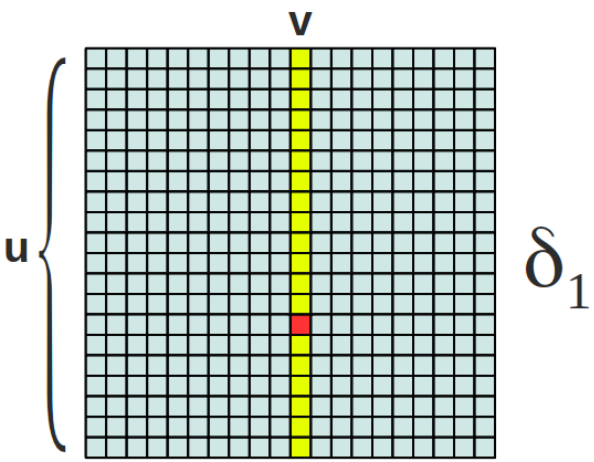


4. single-source inter-part boundary distances

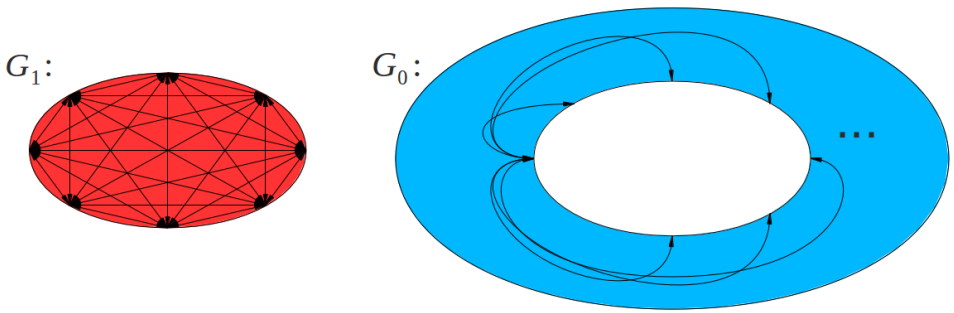


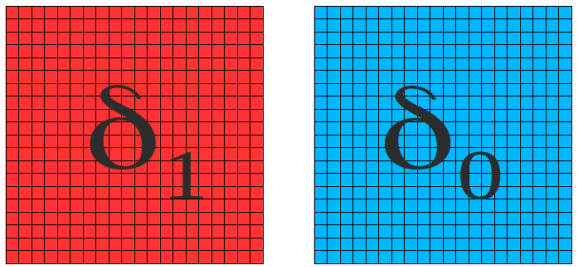


$$\forall v \in C \; d(v) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$

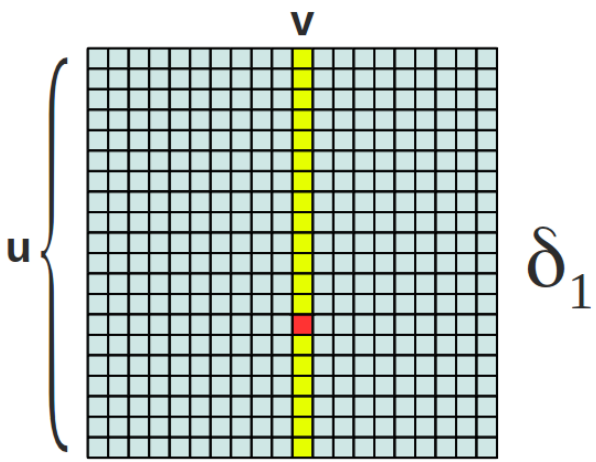


4. single-source inter-part boundary distances





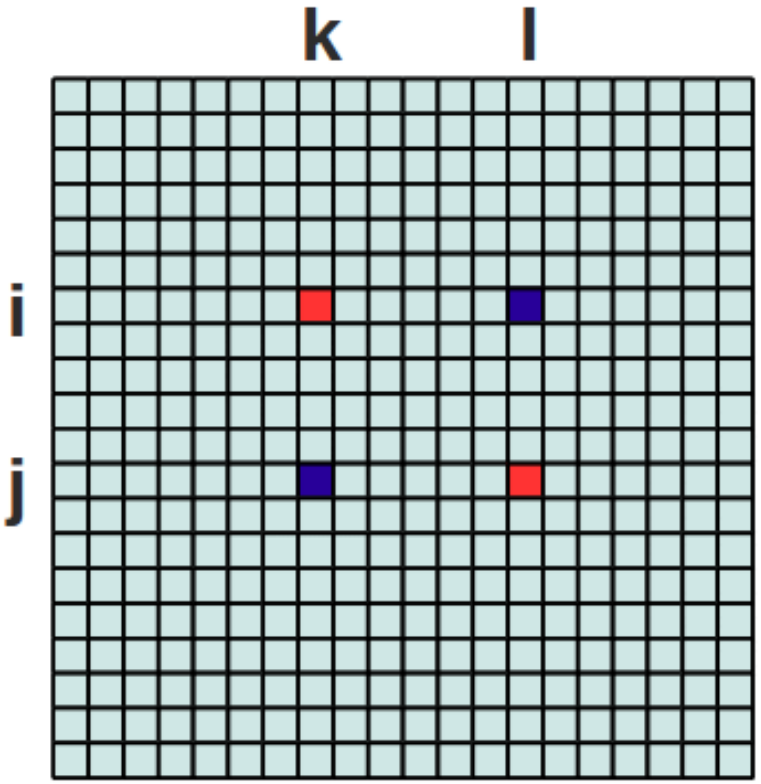
$$\forall v \in C \; d(v) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$



finding column minima!

Monge Matrices

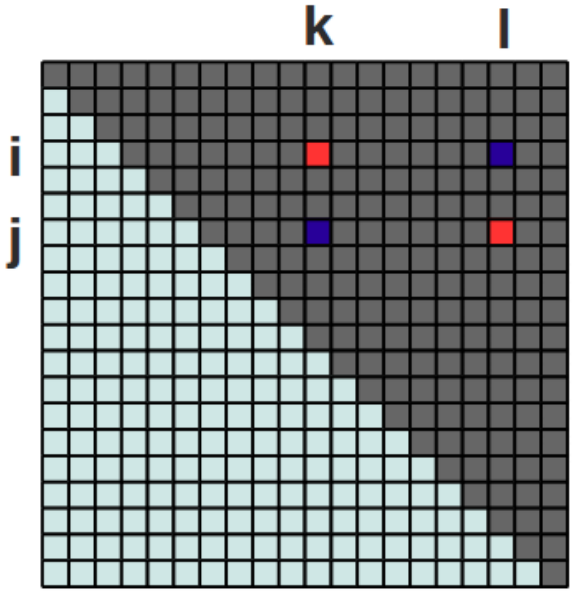
A:



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

Monge Property

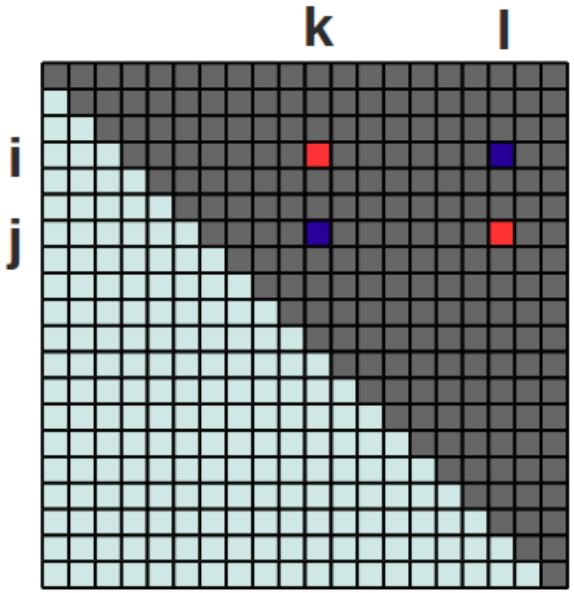
δ :

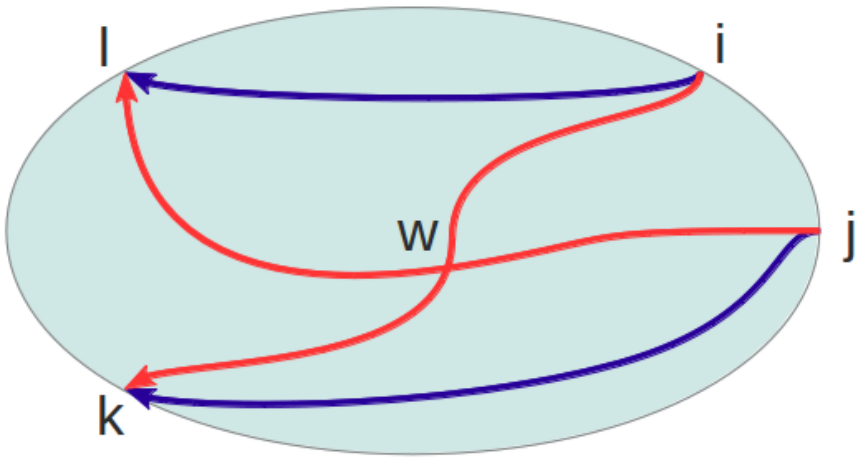


$$A(i,k) + A(j,l) \geq A(i,l) + A(j,k)$$

Monge Property

δ :

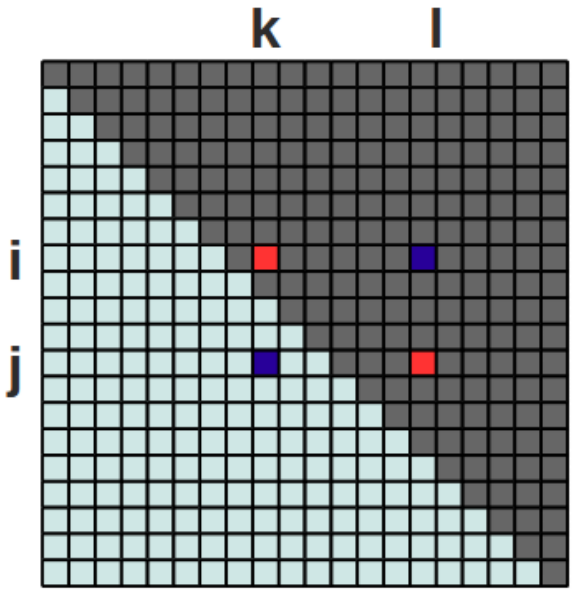


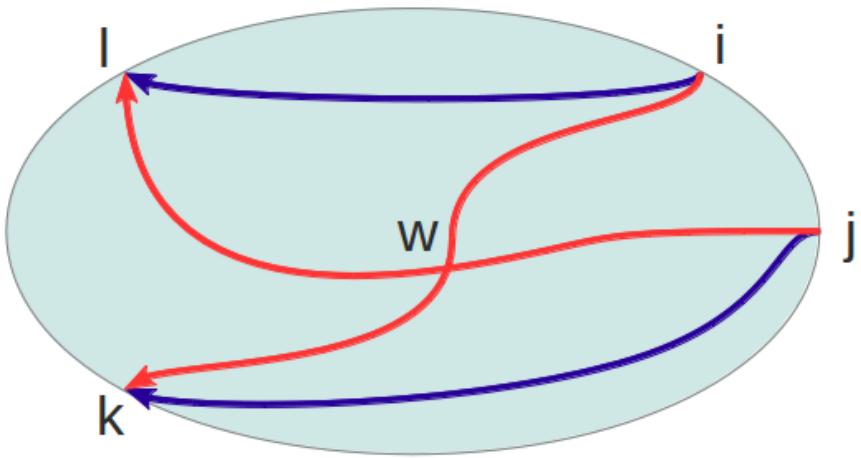


$$A(i,k) + A(j,l) \geq A(i,l) + A(j,k)$$

Monge Property

δ :

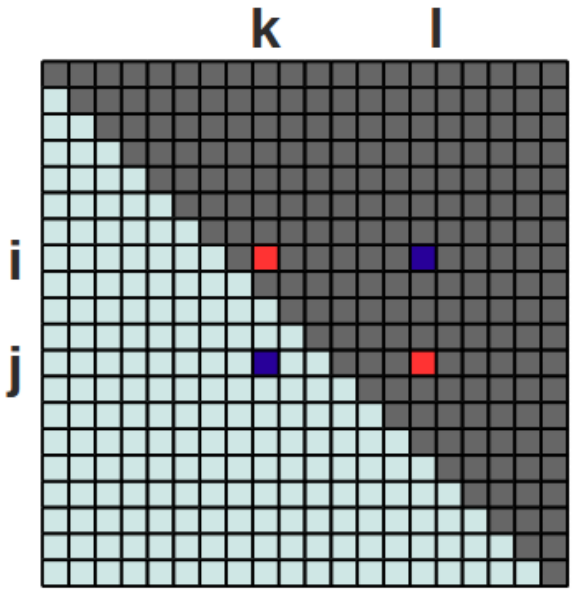


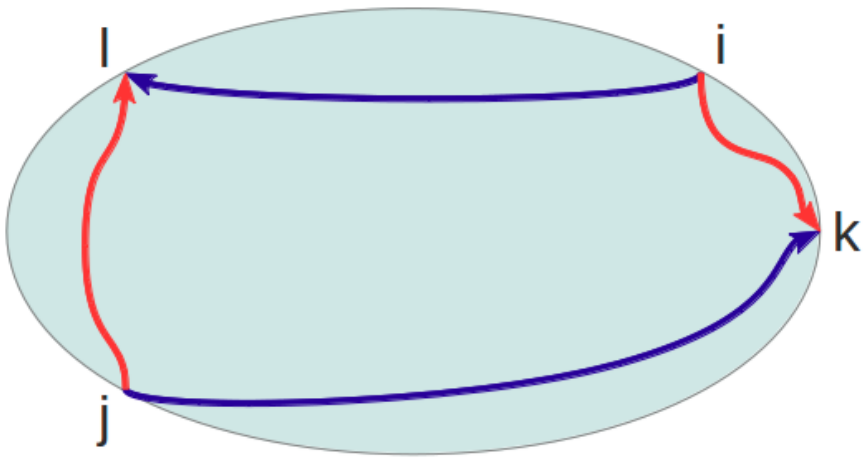


$$A(i,k) + A(j,l) \geq A(i,l) + A(j,k)$$

Monge Property

δ :

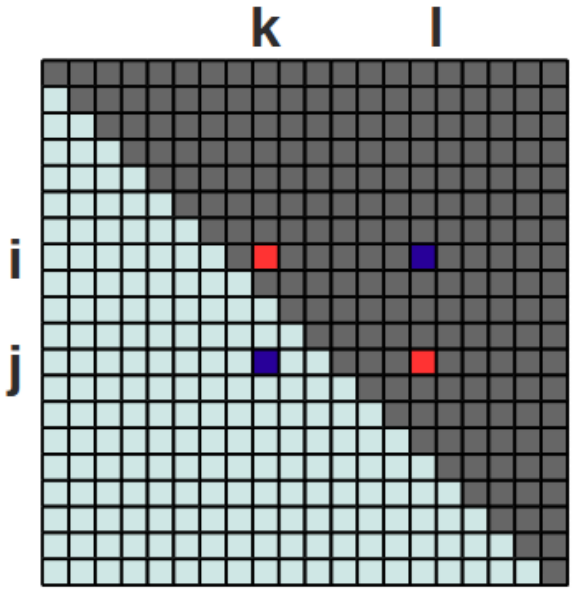


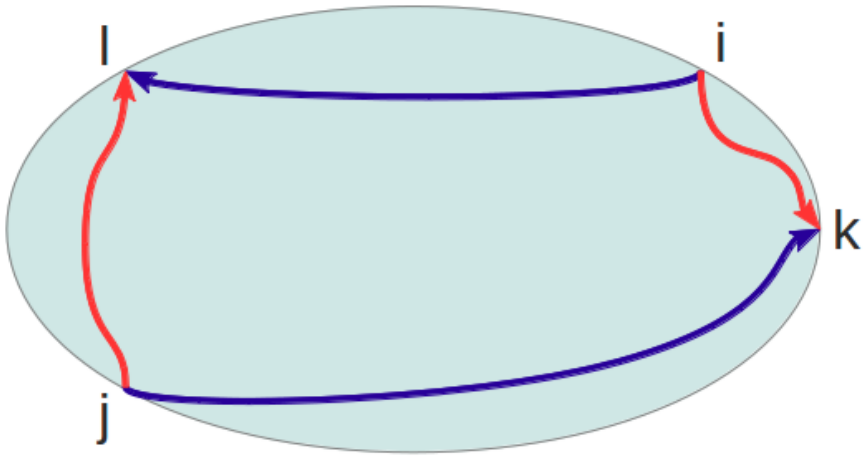


$$A(i,k) + A(j,l) \geq A(i,l) + A(j,k)$$

Monge Property

δ :

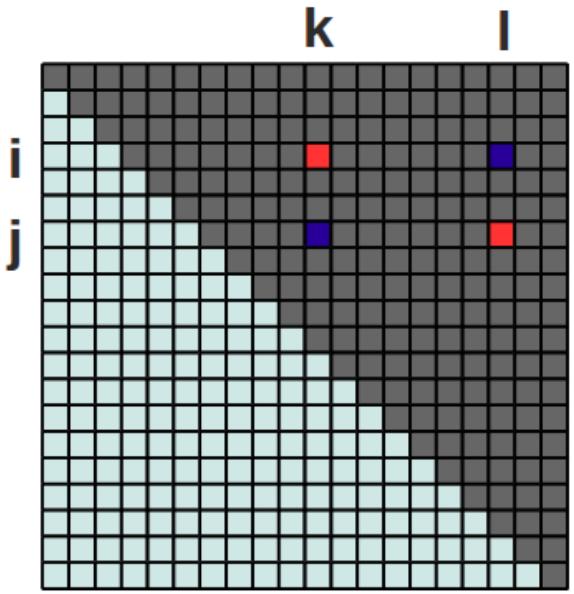




$$A(i,k) + A(j,l) \geq A(i,l) + A(j,k)$$

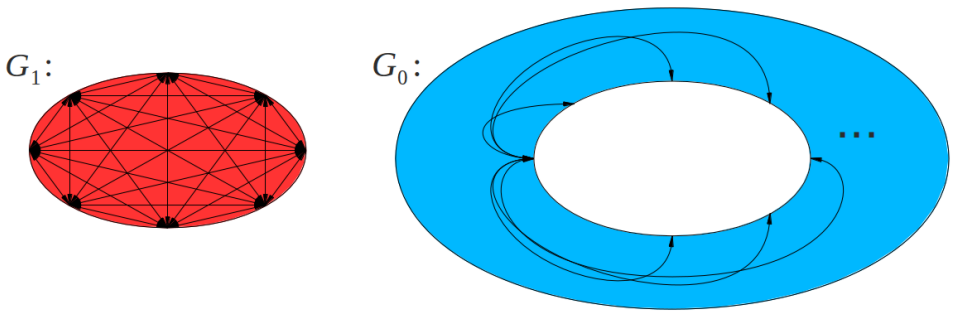
Monge Property

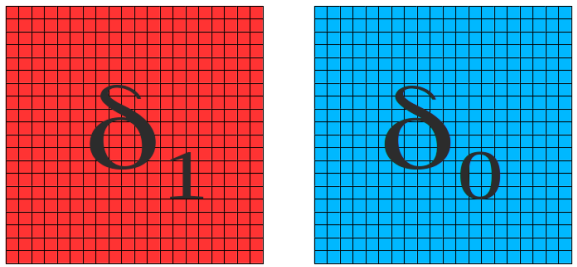
δ :



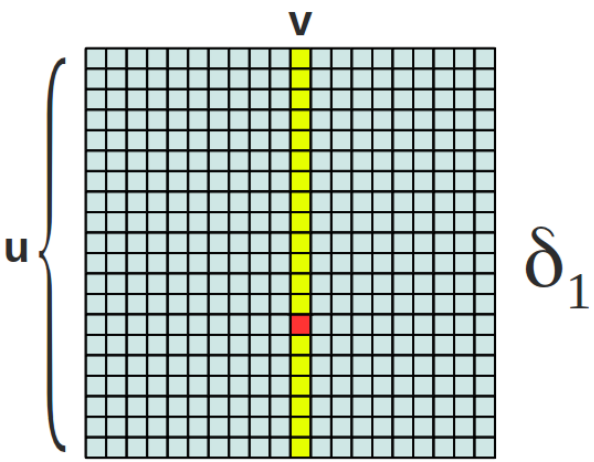
- it is possible to find Column Minima in a triangular Monge matrix in $O(n\alpha(n))$ [Klawe-Kleitman 1990]

4. single-source inter-part boundary distances

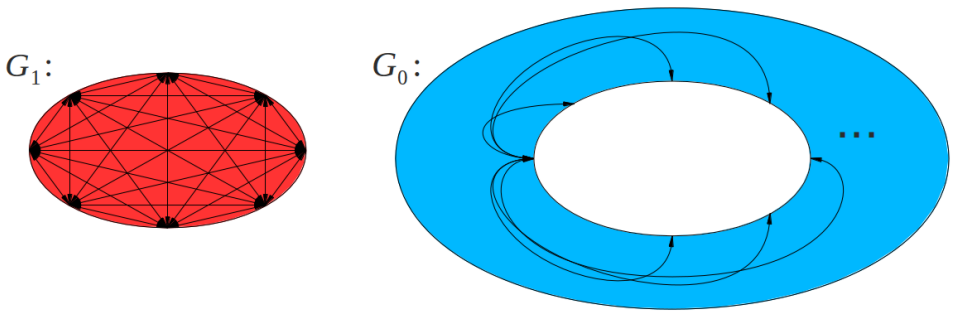


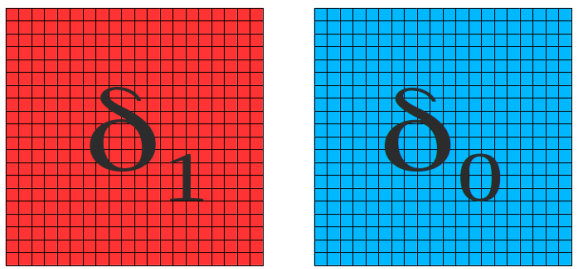


$$\forall v \in C \; d(u) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$

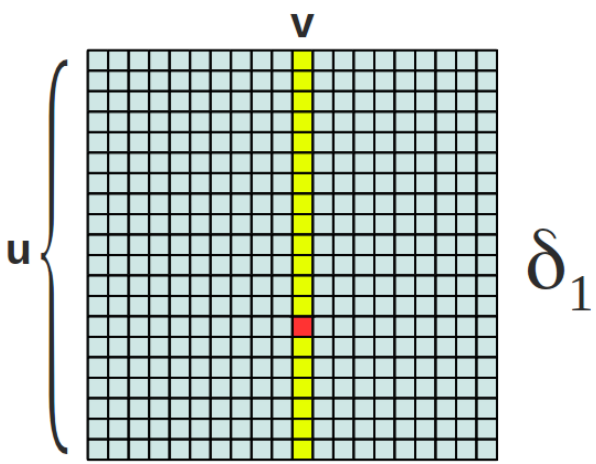


4. single-source inter-part boundary distances





$$\forall v \in C \; d(u) := min_{u \in C} \left\{ \begin{array}{l} d(u) + \delta_1(u, v) \\ d(u) + \delta_0(u, v) \end{array} \right\}$$



- each iteration takes $O(\sqrt{n}\alpha(\sqrt{n}))$
- $\sqrt{n}$ iterations

  overall: $O(n\alpha(n))$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n^{3/2})$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$
6. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

---

step 5 and 6 can be done in $O(n)$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n\alpha(n))$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$
6. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

---

step 5 and 6 can be done in $O(n)$

1. initialization $\longrightarrow O(n)$
2. recursive call $\longrightarrow log(n)$ levels
3. MSSP $\longrightarrow O(n\ log(n))$
4. Bellman Ford $\longrightarrow O(n\alpha(n))$
5. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$
6. feasible price function, Dijkstra $\longrightarrow O(n\ log(n))$

---

step 5 and 6 can be done in $O(n)$

overall: $O(n\ log^2 n)$

Sources:

[Image 1: image segmentation]
http://spie.org/x8899.xml?pf=true&ArticleID=x8899

http:
//www.leda-tutorial.org/de/offiziell/ch05s02s06.html

Thank You