Shortest Path in Planar Graphs with real lengths

Thomas Prinz

November 26, 2013

Introduction

The Problem

Given:
Arbitrary planar Graph G=(V,E) with some $s \in V$ and real edge lengths.

The Problem

Given:
Arbitrary planar Graph G=(V,E) with some $s \in V$ and real edge lengths.

Assumptions:

- The Graph does not have negative cycles.

The Problem

Given:
Arbitrary planar Graph G=(V,E) with some $s \in V$ and real edge lengths.

Assumptions:

- The Graph does not have negative cycles.
- The Graph is maximum planar.

The Problem

Given:

Arbitrary planar Graph G=(V,E) with some $s \in V$ and real edge lengths.

Assumptions:

- The Graph does not have negative cycles.
- The Graph is maximum planar.

Goal:

Compute the shortest Path from s to all other vertices.

Applications?

Applications?

Applications?

Applications?

- Route Planing

Applications?

Applications?

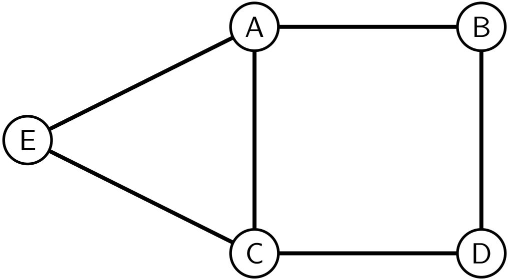
- Route Planing
- Networks/Routing

Applications?

Applications?

- Route Planing
- Networks/Routing
- Computer Vision techniques

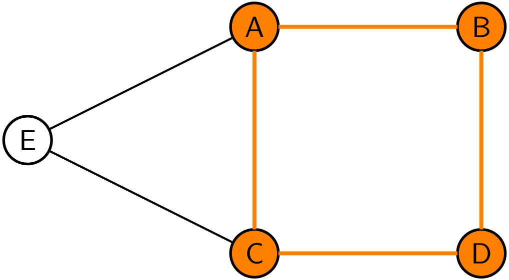
Regions of a Graph



Definition

A Subgraph induced by $S \subset V$ is called a Region of the Graph

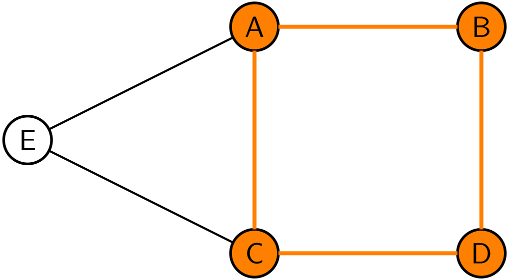
Regions of a Graph



Definition

A Subgraph induced by $S \subset V$ is called a Region of the Graph

Boundary Vertices



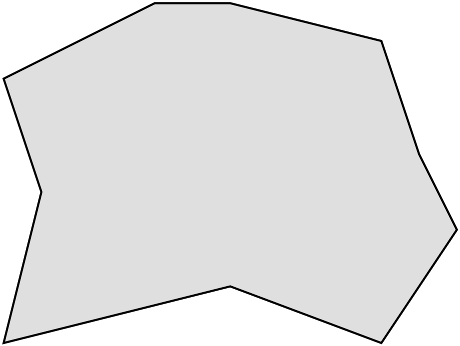
Definition

Vertices of a Region that connect it with the rest of the Graph are called Boundary Vertices (A,C). Other Vertices are called interior Vertices(B,D).

Holes of a Graph

Definition

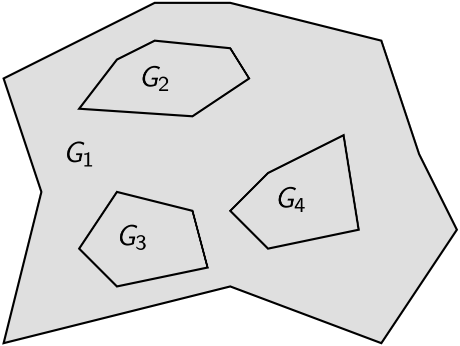
Faces in a Subgraph, that do not exists in the actual Graph are called unnatural faces or holes.



Holes of a Graph

Definition

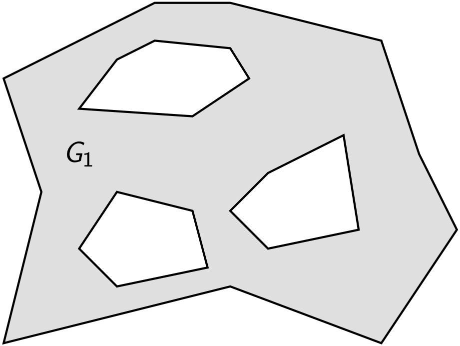
Faces in a Subgraph, that do not exists in the actual Graph are called unnatural faces or holes.



Holes of a Graph

Definition

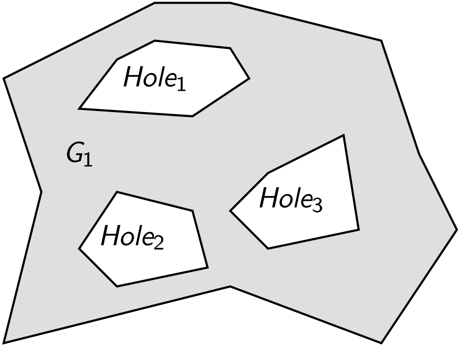
Faces in a Subgraph, that do not exists in the actual Graph are called unnatural faces or holes.



Holes of a Graph

Definition

Faces in a Subgraph, that do not exists in the actual Graph are called unnatural faces or holes.



Price functions

Definition

A function $P : V \rightarrow R$ is called a price function.

Price functions

Definition

A function $P : V \to R$ is called a price function.

Definition

A Price function induces a reduced cost function $w : E \to R$ with w(u,v) = p(u) + l(u,v) - p(v) for every edge e(u,v) where l(u,v) is the length of e(u,v).

Price functions

Definition

A function $P : V \rightarrow R$ is called a price function.

Definition

A Price function induces a reduced cost function $w : E \rightarrow R$ with w(u,v) = p(u) + l(u,v) - p(v) for every edge e(u,v) where l(u,v) is the length of e(u,v).

Definition

If the reduced cost function assigns every edge a non-negative value, then we say P is a feasible price function.

Price functions

Lemma

A Single Source Shortest Path distance from one vertex r (root) to all others induces a feasible price function on the Graph G with:
$\phi(v)$ = *(shortest) r-to-v distance*

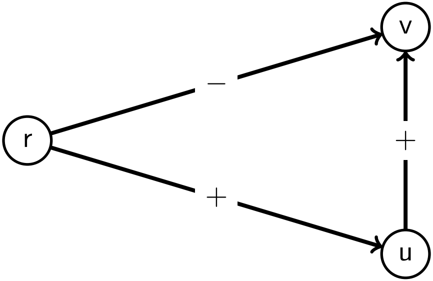
$$l_\phi(u, v) = l(u, v) + \phi(u) - \phi(v)$$

Price functions

Lemma

A Single Source Shortest Path distance from one vertex r (root) to all others induces a feasible price function on the Graph G with:
$\phi(v) =$ *(shortest) r-to-v distance*

$$l_\phi(u, v) = l(u, v) + \phi(u) - \phi(v)$$



The improved Algorithm of Mozes/Wulff-Nilsen

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.
4. Compute the distances from r to every boundary node in G.

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.
4. Compute the distances from r to every boundary node in G.
5. Use Dijkstra with feasible price function to compute distances from r to all nodes (in G_i)

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.
4. Compute the distances from r to every boundary node in G.
5. Use Dijkstra with feasible price function to compute distances from r to all nodes (in G_i)
6. Use SSSP distances from 5. as price funtion and use Dijstra.

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$
5. Price function/Dijkstra $\Rightarrow$ $O(n \cdot log(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$
5. Price function/Dijkstra $\Rightarrow$ $O(n \cdot log(n))$
6. Price function/Dijkstra $\Rightarrow$ $O(n \cdot log(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$
5. Price function/Dijkstra $\Rightarrow$ $O(n \cdot log(n))$
6. Price function/Dijkstra $\Rightarrow$ $O(n \cdot log(n))$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$
5. Price function/Henzinger $\Rightarrow$ $O(n)$
6. Price function/Henzinger $\Rightarrow$ $O(n)$

The Algorithm of Klein (Reminder)

1. Dividing $\Rightarrow$ $O(n)$
2. Recursive Call $\Rightarrow$ $O(log(n))$
3. MSSP $\Rightarrow$ $O(n \cdot log(n))$
4. Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$
5. Price function/Henzinger $\Rightarrow$ $O(n)$
6. Price function/Henzinger $\Rightarrow$ $O(n)$

The Algorithm of Klein (Reminder)

1. Divide the Graph into two Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.
4. Compute the distances from r to every boundary node in G.
5. Use Dijkstra with feasible price function to compute distances from r to all nodes (in G_i)
6. Use SSSP distances from 5. as price funtion and use Dijstra.

The Algorithm of Klein (Reminder)

1. Divide the Graph into p Subgraphs using planar separators.
2. Recursivly compute SSSP for both subgraphs.
3. Compute MSSP of all pairs of boundary nodes in Subgraphs.
4. Compute the distances from r to every boundary node in G.
5. Use Dijkstra with feasible price function to compute distances from r to all nodes (in G_i)
6. Use SSSP distances from 5. as price funtion and use Dijstra.

Modifying the algorithm of Klein

We modify the algorithm of Klein, so that it works with a division into p subgraphs instead of 2.

Modifying the algorithm of Klein

We modify the algorithm of Klein, so that it works with a division into p subgraphs instead of 2.

Assumption:

- No Subgraph has holes.
  (Therefore all boundary nodes lie on a single face)

Millers Cycle Seperator Therorem

Definition

Given a triangulated planar Graph G with n vertices. One can find a cycle seperator (of length $O(\sqrt{n})$) that divides the Graph into 2 Pieces, with each having 2n/3 vertices at most in linear time

An r-Division (Dividing Step)

Definition

By recursivly using a cycle Separator one can divide a Graph into $O(n/r)$ Subgraphs. Such a Division of the Graph is called r-Division.

An r-Division (Dividing Step)

Definition

By recursivly using a cycle Separator one can divide a Graph into O(n/r) Subgraphs. Such a Division of the Graph is called r-Division.

1. No two Subgraphs share Interior vertices.

An r-Division (Dividing Step)

Definition

By recursivly using a cycle Separator one can divide a Graph into O(n/r) Subgraphs. Such a Division of the Graph is called r-Division.

1. No two Subgraphs share Interior vertices.
2. Each Subgraphs has at most r vertices and $O(\sqrt{r})$ boundary vertices.

An r-Division (Dividing Step)

Definition

By recursivly using a cycle Separator one can divide a Graph into O(n/r) Subgraphs. Such a Division of the Graph is called r-Division.

1. No two Subgraphs share Interior vertices.
2. Each Subgraphs has at most r vertices and $O(\sqrt{r})$ boundary vertices.
3. Each Subgraphs has a boundary contained in $O(1)$ Faces (Holes), defined by simple cycles.

An r-Division (Dividing Step)

Definition

By recursivly using a cycle Separator one can divide a Graph into O(n/r) Subgraphs. Such a Division of the Graph is called r-Division.

1. No two Subgraphs share Interior vertices.

2. Each Subgraphs has at most r vertices and $O(\sqrt{r})$ boundary vertices.

3. Each Subgraphs has a boundary contained in $O(1)$ Faces (Holes), defined by simple cycles.

Cost creating r-Division: $O(n \cdot log(n))$

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions.

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions.

Cost: For each Region $O(|V_R| \cdot log(|V_R|))$ (for $|V_R|$ being the number of vertices in that Region.)

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions.

Cost: For each Region $O(|V_R| \cdot log(|V_R|))$ (for $|V_R|$ being the number of vertices in that Region.)

Overall cost: $O(n \cdot log(n))$ ✓

2. SSSP Inter-region Boundary Distances (Bellman Ford)

Let r be the root, B be the Boundary and b=$|B|$.

1. $e_j[v] := \infty$ for all $v \in B$ and j = 0,...,b
2. $e_0[r] := 0$
3. for j = 1,2,...,b
4. let C be the cycle defining the boundary.
5. $e_j[v] := min_{w \in V_C}\{e_{j-1}[w] + \delta_i[w, v]\}$
6. $D[v] := e_b[v]$ for all $v \in B$

2. SSSP Inter-region Boundary Distances (Bellman Ford)

Let r be the root, B be the Boundary and b=$|B|$.

1. $e_j[v] := \infty$ for all $v \in B$ and j = 0,...,b
2. $e_0[r] := 0$
3. for j = 1,2,...,b
4. for each region $R \in R_{all}$
5. let C be the cycle defining the boundary of R.
6. $e_j[v] := min\{e_j[v], min\{w \in V_C\ e_{j-1}[w] + d_R(w,v)\}\}\ f.\ all\ v \in V_C$
7. $D[v] := e_b[v]$ for all $v \in B$

Bellman Ford time complexity

Optimized Bellman Ford $\Rightarrow$ $O(n \cdot \alpha(n))$

Bellman Ford time complexity

Optimized Bellman Ford	$\Rightarrow$	$O(n \cdot \alpha(n))$
Optimized Bellman Ford (p Regions)	$\Rightarrow$	$O(n \cdot p \cdot \alpha(n))$

Bellman Ford time complexity

$$\begin{array}{lcl} \text{Optimized Bellman Ford} & \Rightarrow & O(n \cdot \alpha(n)) \\ \text{Optimized Bellman Ford (p Regions)} & \Rightarrow & O(n \cdot p \cdot \alpha(n)) \end{array}$$

$$\text{Bellman Ford runs in } O(n \cdot log(n)) \Leftrightarrow p \leq \frac{log(n)}{\alpha(n)}$$

Bellman Ford time complexity

Optimized Bellman Ford	$\Rightarrow$	$O(n \cdot \alpha(n))$
Optimized Bellman Ford (p Regions)	$\Rightarrow$	$O(n \cdot p \cdot \alpha(n))$

Bellman Ford runs in $O(n \cdot log(n)) \Leftrightarrow p \leq \frac{log(n)}{\alpha(n)}$

Optimal $p = \frac{log(n)}{\alpha(n)}$

3. Single-source Inter-region Distances (for all vertices)

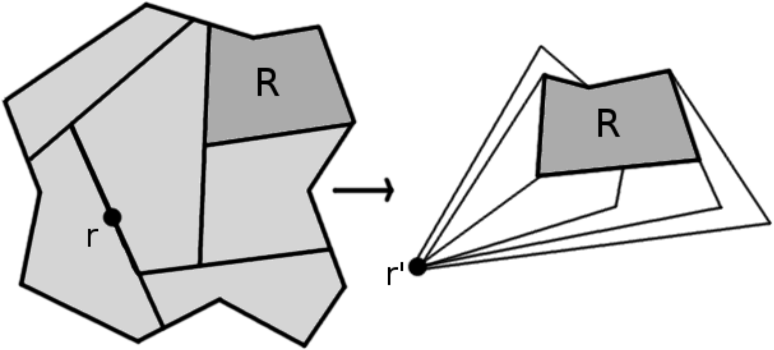
For each Region R we have the SSSP from a boundary vertex r_R already recursively computed.

3. Single-source Inter-region Distances (for all vertices)

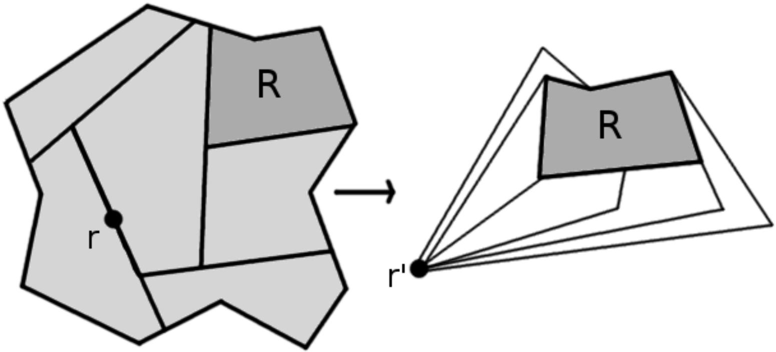
For each Region R we have the SSSP from a boundary vertex r_R already recursively computed.
We have the distances from some boundary vertex r to all other boundary vertices

3. Single-source Inter-region Distances (for all vertices)

For each Region R we have the SSSP from a boundary vertex r_R already recursively computed.
We have the distances from some boundary vertex r to all other boundary vertices

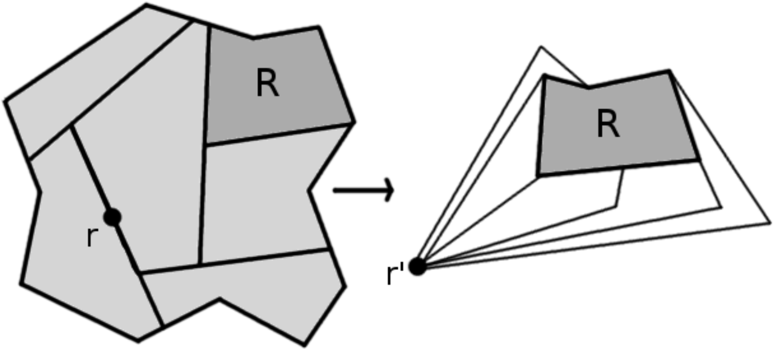


3. Single-source Inter-region Distances (for all vertices)



$\phi(v) = d_R(r_R, v)$ *if* $v \neq r'$
$\phi(v) = max\{d_R(r_R, b) - d_G(r, b)\}$ *if* $v = r'$ $(b \in B_R)$
Claim: ϕ is a feasible price function.

3. Single-source Inter-region Distances (for all vertices)



Cost: $O(|V_R| \cdot log(|V_R|))$ (for $|V_R|$ Vertices per Region.)
Overall Cost: $O(n \cdot log(n))$ ✓

4. Compute feasible price function for entire G

The Last step is identical to original algorithm:

4. Compute feasible price function for entire G

The Last step is identical to original algorithm:

1. Take SSSP Distances from previous Step to compute a feasible price function (O(n))

4. Compute feasible price function for entire G

The Last step is identical to original algorithm:

1. Take SSSP Distances from previous Step to compute a feasible price function (O(n))

2. Use Dijstras algorithm ($O(n \cdot log(n))$) or Henzingers algorithm (O(n)) to compute SSSP for arbitrary source nodes.

4. Compute feasible price function for entire G

The Last step is identical to original algorithm:

1. Take SSSP Distances from previous Step to compute a feasible price function (O(n))

2. Use Dijstras algorithm ($O(n \cdot log(n))$) or Henzingers algorithm (O(n)) to compute SSSP for arbitrary source nodes.

Overall Cost: $O(n)/O(n \cdot log(n))$

Holes?

What if we have Holes?

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions once for every face/hole.

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions once for every face/hole.

Cost: For each Region $O(|V_R| \cdot log(|V_R|) + h \cdot |V_R| \cdot log(|V_R|))$ (for $|V_R|$ being the number of vertices in that Region.)

1. Intra-region Boundary Distances (MSSP)

Apply the MSSP algorithm (Klein) to all p Regions once for every face/hole.

Cost: For each Region $O(|V_R| \cdot log(|V_R|) + h \cdot |V_R| \cdot log(|V_R|))$ (for $|V_R|$ being the number of vertices in that Region.)

Overall cost: $O(n \cdot log(n))$ ✓

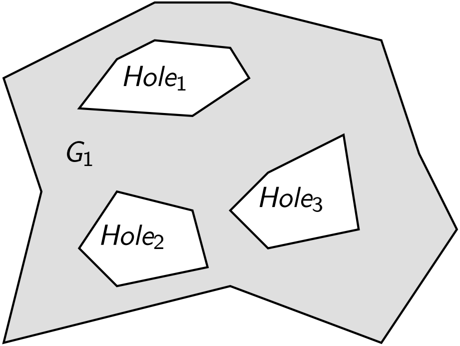
2. Inter-region Boundary Distances (Bellman Ford)

Optimized Bellman Ford Pseudocode:

Let r be the root, B be the Boundary and b=$|B|$.

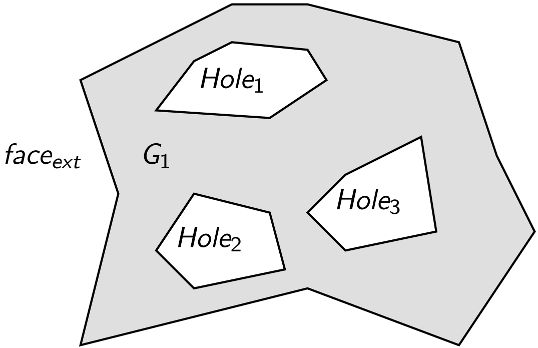
1. initialize vector $e_j[v]$ for j = 0,...,b and $v \in B$
2. $e_j[v] := \infty$ for all $v \in B$ and j = 0,...,b
3. $e_0[r] := 0$
4. for j = 1,...,b
5. for each region $R \in R_{all}$
6. let C be the cycle defining the boundary of R
7. $e_j[v] := min\{e_j[v], min\{w \in V_C\ e_{j-1}[w] + d_R(w, v)\}\}\ f.\ all\ v \in V_C$
8. $D[v] := e_b[v]$ for all $v \in B$

2. Inter-region Boundary Distances (Bellman Ford)



Problem: We have no Cylce C that defines the boundary. There are holes, each defining some of the boundary.

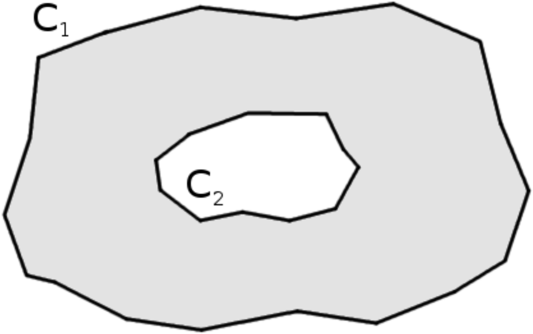
2. Inter-region Boundary Distances (Bellman Ford)



Consider: (H1,H2) (H1,H3) (H1,face) (H2,H1) (H2,H3) (H2,face) ... $(O(h^2) = O(1))$

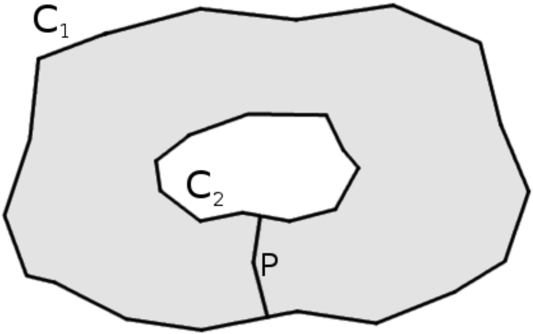
2. Inter-region Boundary Distances (Bellman Ford)

1. Let (C_1, C_2) be a pair of holes/external face.



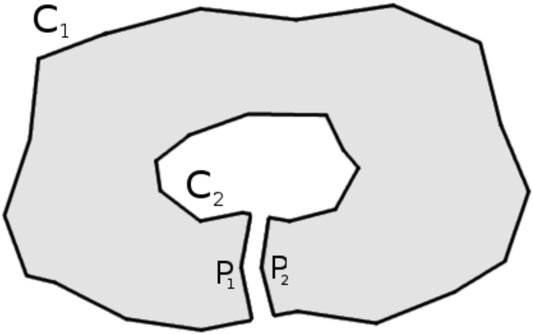
2. Inter-region Boundary Distances (Bellman Ford)

2. Let P be some Path starting in C_1 and ending in C_2.



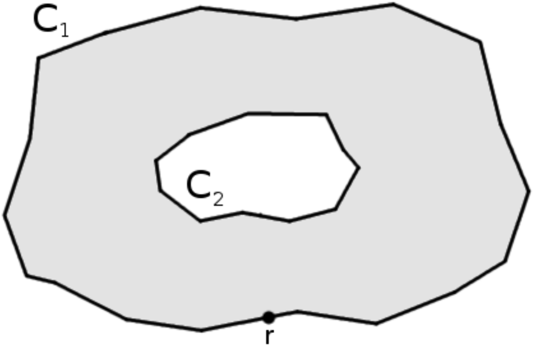
2. Inter-region Boundary Distances (Bellman Ford)

3. Cut the graph along P. Now all node lie on the same face again.



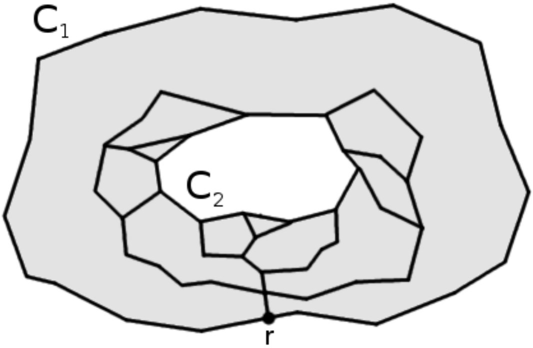
2. Inter-region Boundary Distances (Bellman Ford)

3. Choose some node r on C_1.



2. Inter-region Boundary Distances (Bellman Ford)

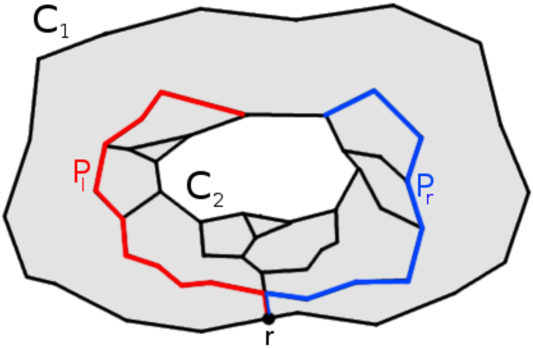
4. Compute the shortest Path Tree from r to all nodes on C_2.



2. Inter-region Boundary Distances (Bellman Ford)

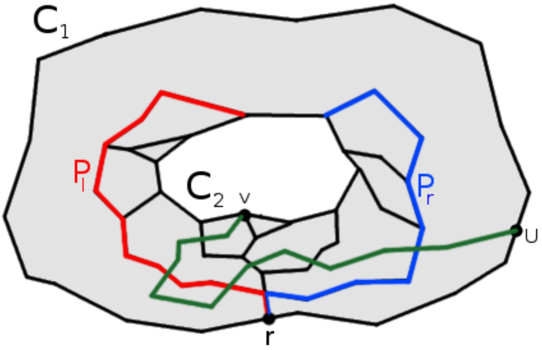
Lemma

There is always a shortest Path from $v \in C_1$ to $u \in C_2$ which does not cross both, P_l and P_r.



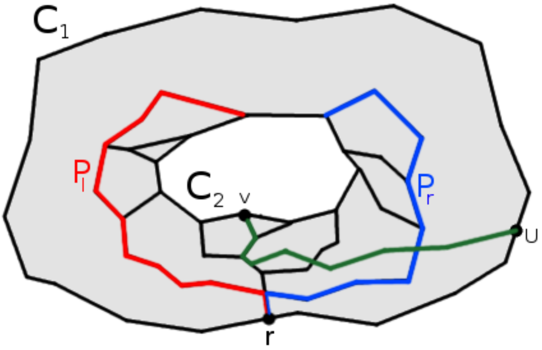
2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 1:



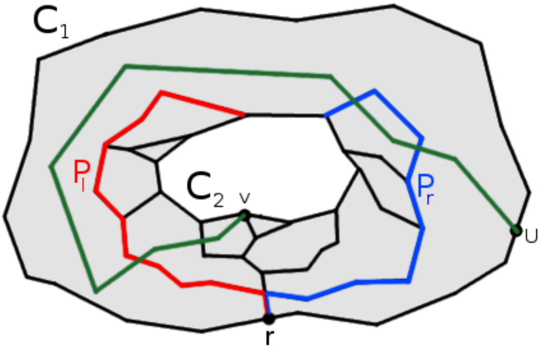
2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 1:



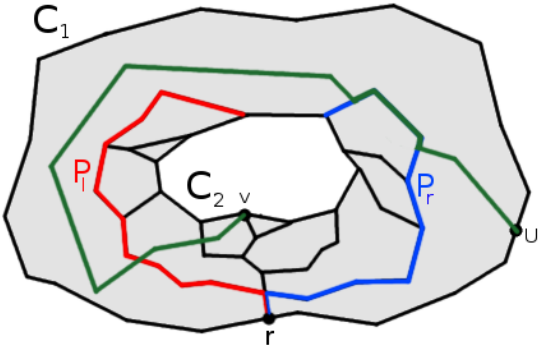
2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 2:



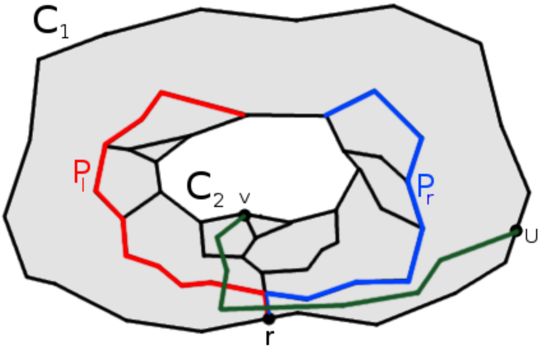
2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 2:



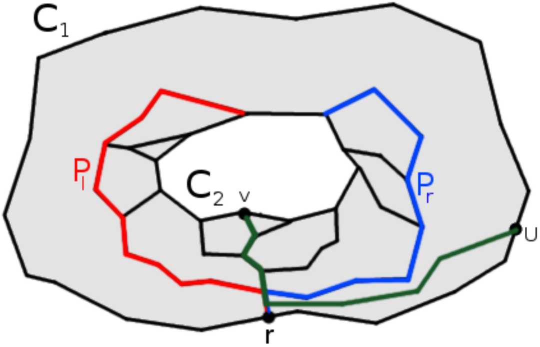
2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 3:



2. Inter-region Boundary Distances (Bellman Ford)

Assume there is a shortest Path, say from u to v, that crosses both:
Case 3:



New modified Bellman Ford

Let r be the root, B be the Boundary and b=$|B|$.

1. initialize vector $e_j[v]$ for j = 0,...,b and $v \in B$
2. $e_j[v] := \infty$ for all $v \in B$ and j = 0,...,b
3. $e_0[r] := 0$
4. for j = 1,...,b
5. for each region $R \in R_{all}$
6. for each Pair (C_1, C_2)
7. IF $(C_1 = C_2)$ THEN continue as before
7. ELSE (assume C_1 is external and d_{P_i} have been precomputed)

$e_j[v] := min\{e_j[v], min_{w \in C_1}\{e_{j-1}[w] + d_{P_r}(w', v')\}\}$ $f.\ all\ v \in C_2$
$e_j[v] := min\{e_j[v], min_{w \in C_1}\{e_{j-1}[w] + d_{P_l}(w', v')\}\}$ $f.\ all\ v \in C_2$

8. $D[v] := e_b[v]$ for all $v \in B$

New modified Bellman Ford time complexity

We have:

- p Regions ($p = \frac{log(n)}{\alpha(n)}$)

New modified Bellman Ford time complexity

We have:

- p Regions ($p = \frac{log(n)}{\alpha(n)}$)
- h^2 Pairs of holes (=O(1))

New modified Bellman Ford time complexity

We have:

- p Regions ($p = \frac{log(n)}{\alpha(n)}$)
- h^2 Pairs of holes (=O(1))
- 2 Iterations per hole pair (=O(1))

Overall Cost: $O(2 \cdot h^2 \cdot p \cdot n \cdot \alpha(n)) = O(n \cdot log(n))$

Overall Cost of Improved algorithm

1. Dividing into p pieces	$\Rightarrow$	$O(n \cdot log(n))$
2. Recursive Call	$\Rightarrow$	$O(\frac{log(n)}{log(log(n))})$
3. MSSP	$\Rightarrow$	$O(n \cdot log(n))$
4. Optimized Bellman Ford	$\Rightarrow$	$O(n \cdot log(n))$
5. Price function/Dijkstra	$\Rightarrow$	$O(n \cdot log(n))$
6. Price function/Dijkstra	$\Rightarrow$	$O(n \cdot log(n))$
Overall Cost:	$\Rightarrow$	$O(\frac{n \cdot log(n) \cdot log(n)}{log(log(n))})$

Conclusion

Conclusion

- SSSP with real weight for planar graphs can be computed in $O(n \cdot logn^2/log(log(n)))$

Conclusion

- SSSP with real weight for planar graphs can be computed in $O(n \cdot logn^2/log(log(n)))$
- Most of the time is used to find a feasible price function, we then simply use djistra.

Conclusion

- SSSP with real weight for planar graphs can be computed in $O(n \cdot logn^2/log(log(n)))$
- Most of the time is used to find a feasible price function, we then simply use djistra.
- Is it possible to find a feasible price function in $O(n \cdot log(n))$ time?

Conclusion

- SSSP with real weight for planar graphs can be computed in $O(n \cdot logn^2/log(log(n)))$
- Most of the time is used to find a feasible price function, we then simply use djistra.
- Is it possible to find a feasible price function in $O(n \cdot log(n))$ time?
- What about Constanst hidden in $O - Notation$?

Thank You!