Shortest Paths in the Plane with Obstacle Violations

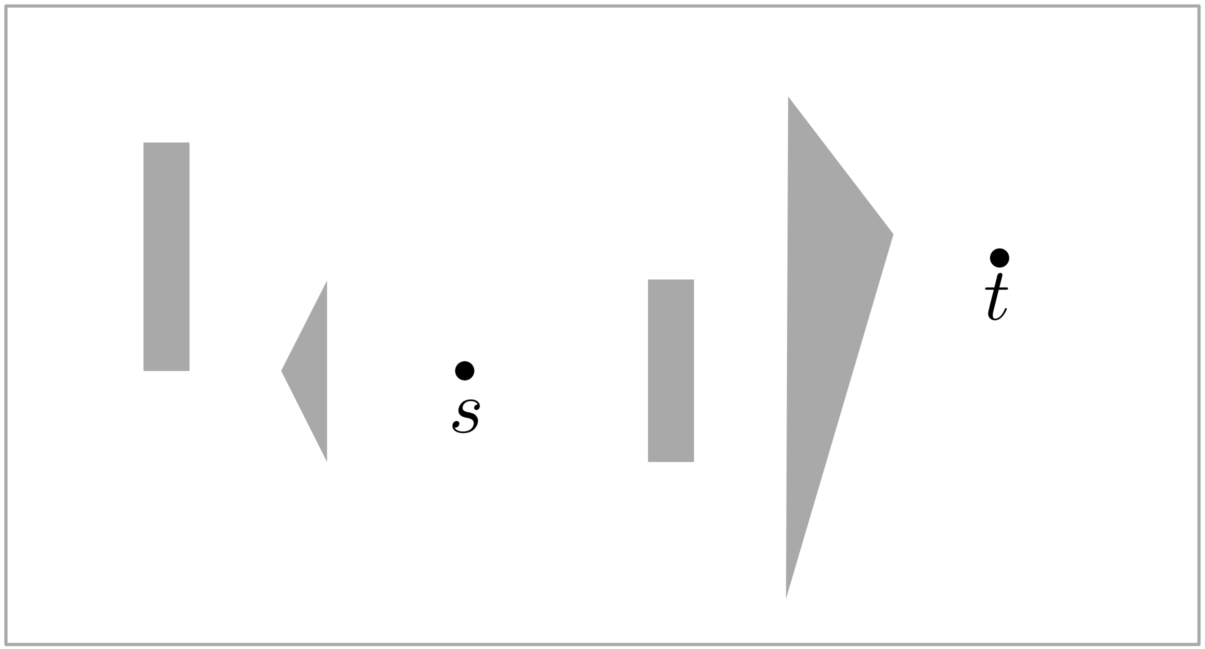
John Hershberger,[*] Neeraj Kumar[†] and Subhash Suri[†]

[*] Mentor Graphics Corporation

[†] University of California, Santa Barbara

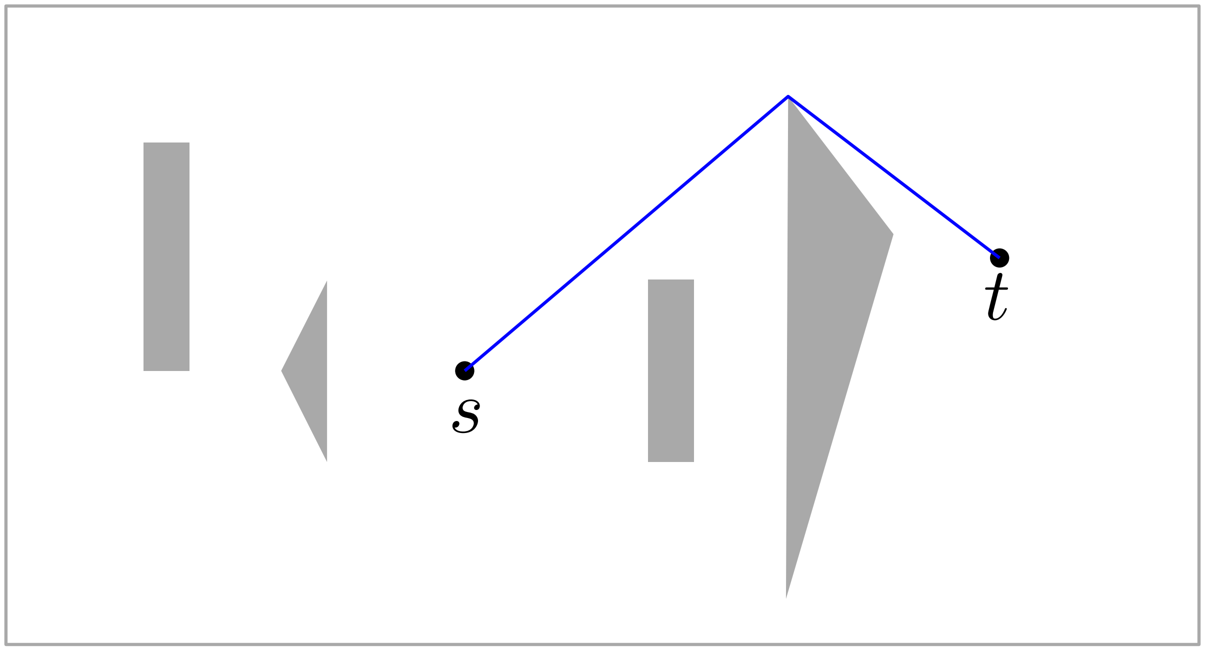
Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t



Problem Description

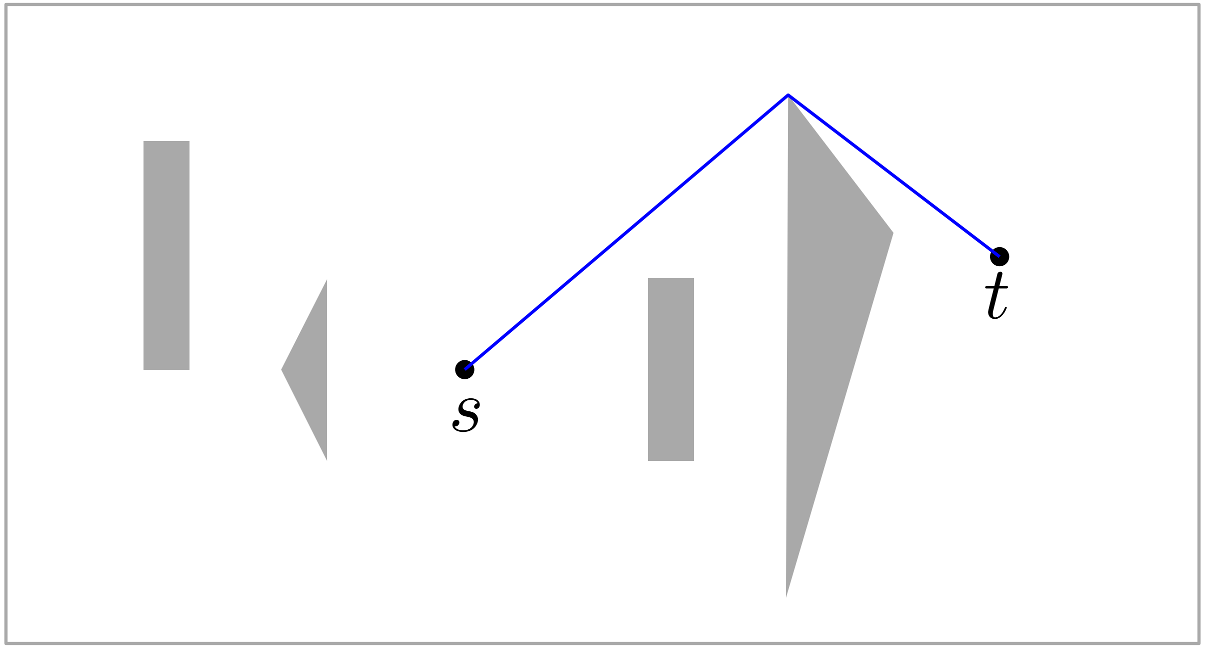
Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t



Classical Shortest Path Problem : Obstacles Impenetrable

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t

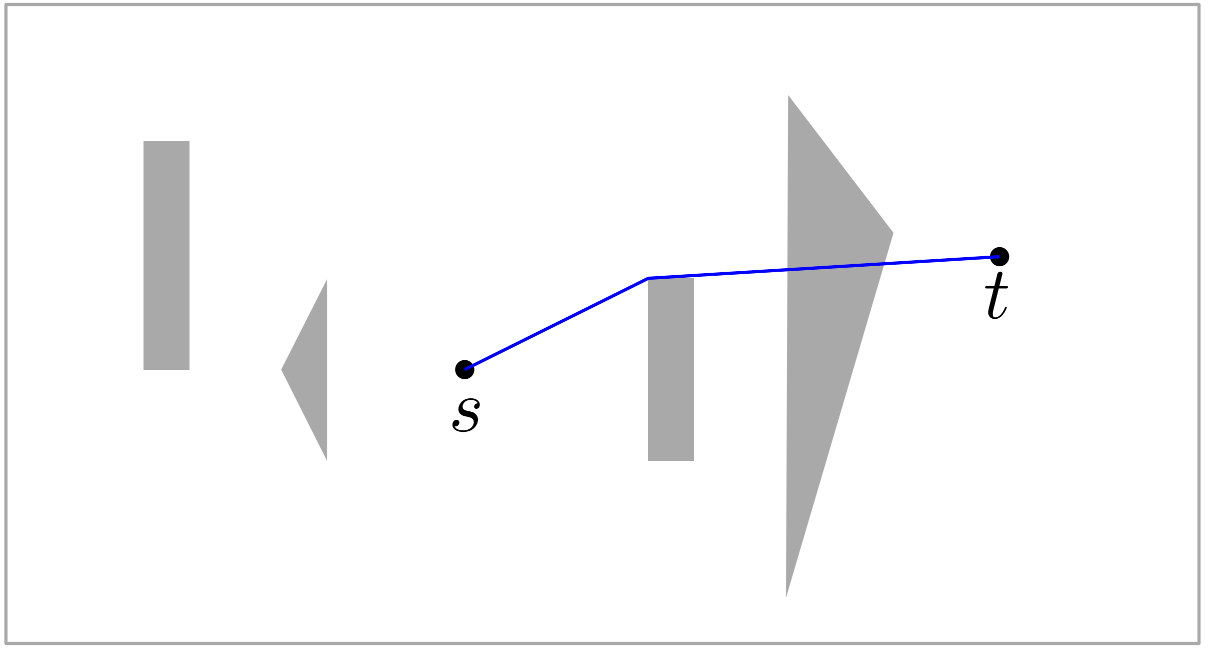


Classical Shortest Path Problem : Obstacles Impenetrable

What if shortest paths can go through obstacles?

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t

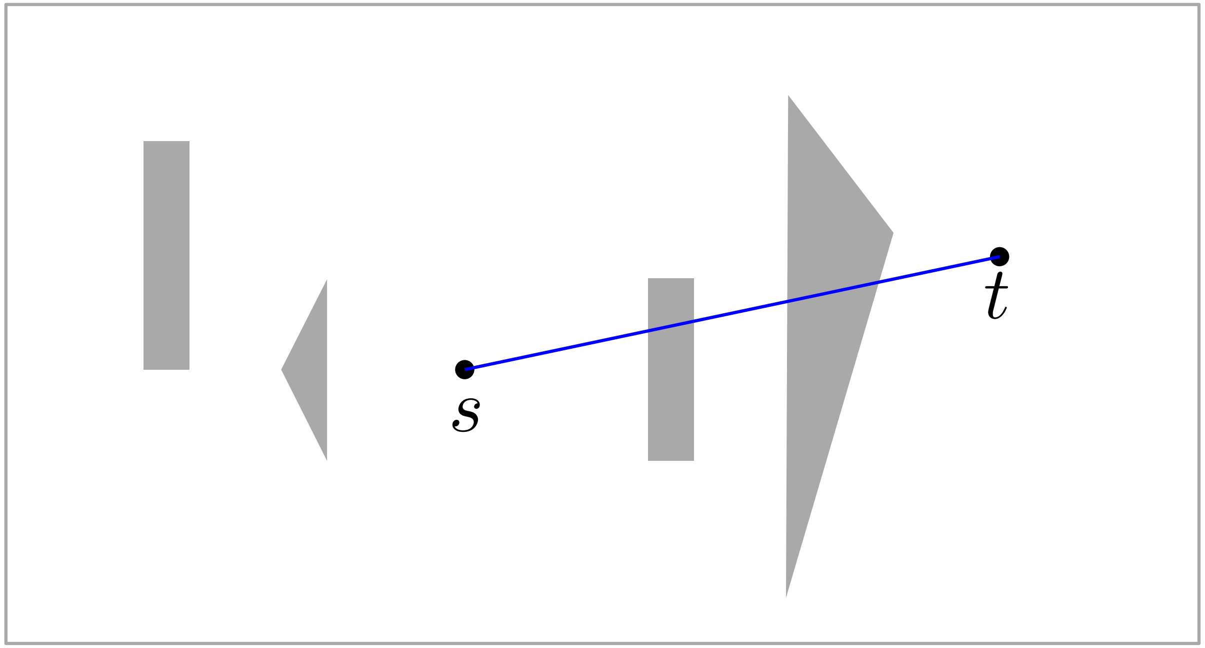


Allowed to cross one obstacle

What if shortest paths can go through obstacles?

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t

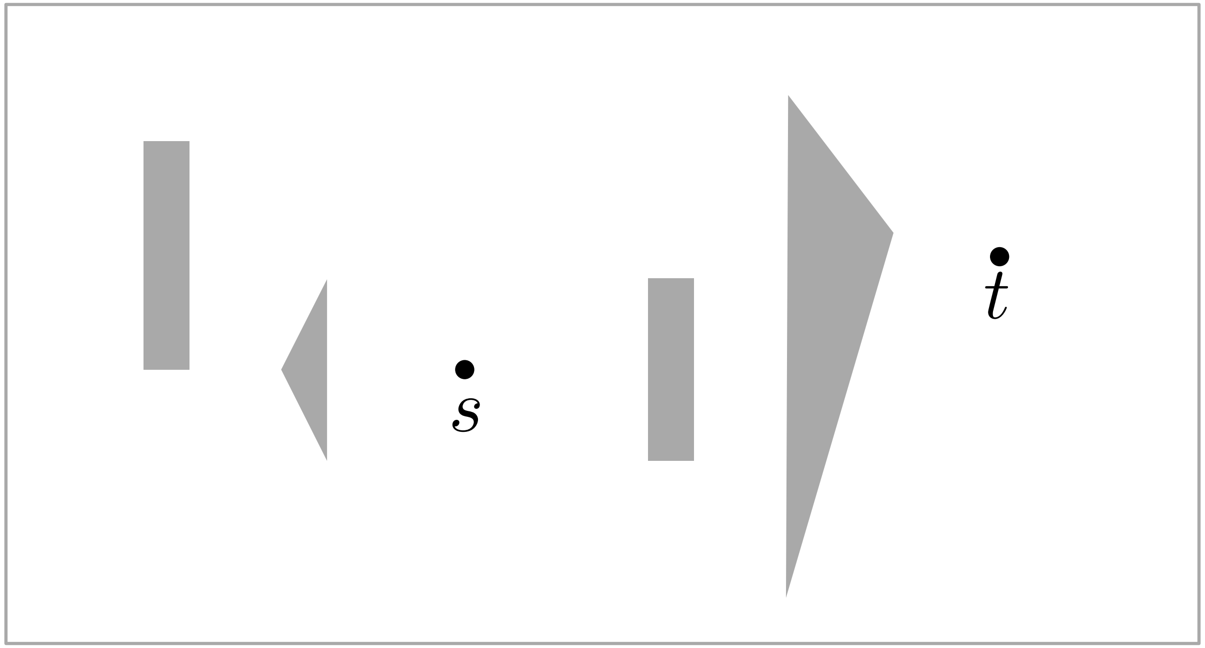


Allowed to cross two obstacles

What if shortest paths can go through obstacles?

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t

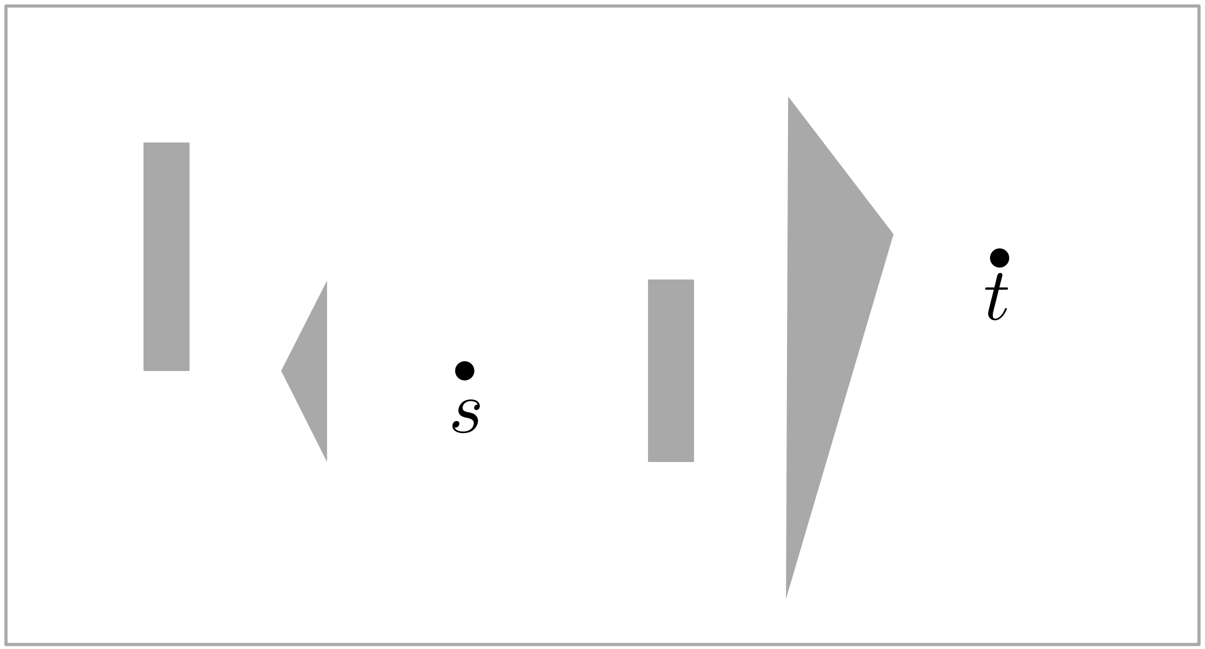


What if shortest paths can go through obstacles?

Compute shortest s–t path that can go through at most k obstacles

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t



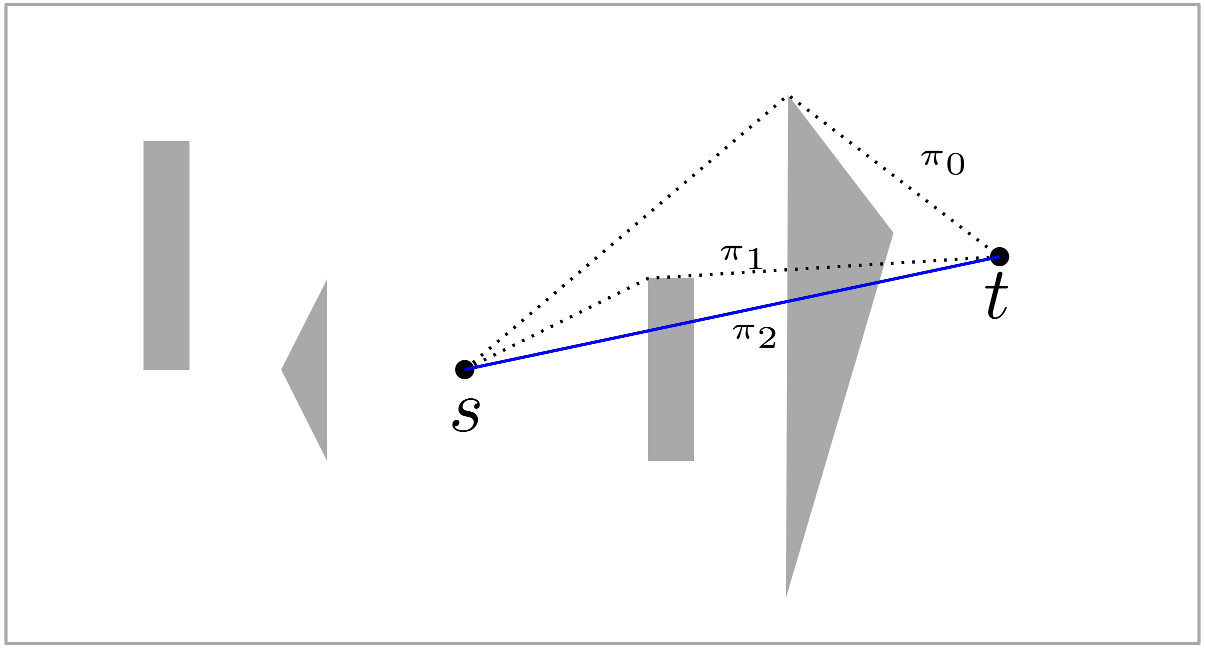
What if shortest paths can go through obstacles?

Compute shortest s–t $\underbrace{\text{path that can go through at most } k \text{ obstacles}}$

k-path π_k

Problem Description

Input: A polygonal domain P with h convex obstacles, n vertices, source s and target t



Shortest k-paths

Compute shortest s–t path that can go through at most k obstacles

k-path $\quad \pi_k$

Shortest k-Paths : Motivation

A ‘less restricted’ version of classical shortest path problem

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

Lots of Related work

Visibility Graph [GM '91, KM'88. . .]

Continuous Dijkstra [HS '97, RS'94, M'87 . . .]

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

What if feasible shortest path is too long?

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

What if feasible shortest path is too long?

Optimization Problem with constraints that can be violated

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

What if feasible shortest path is too long?

Optimization Problem with constraints that can be violated

$\Rightarrow$ Robot Motion Planning

For example , obstacles are doors that can be opened

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

What if feasible shortest path is too long?

Optimization Problem with constraints that can be violated

$\Rightarrow$ Robot Motion Planning

For example , obstacles are doors that can be opened

$\Rightarrow$ Path Planning

Paying for a toll bridge vs a longer route?

Shortest k-Paths : Motivation

A 'less restricted' version of classical shortest path problem

What if feasible shortest path is too long?

Optimization Problem with constraints that can be violated

$\Rightarrow$ Robot Motion Planning

For example , obstacles are doors that can be opened

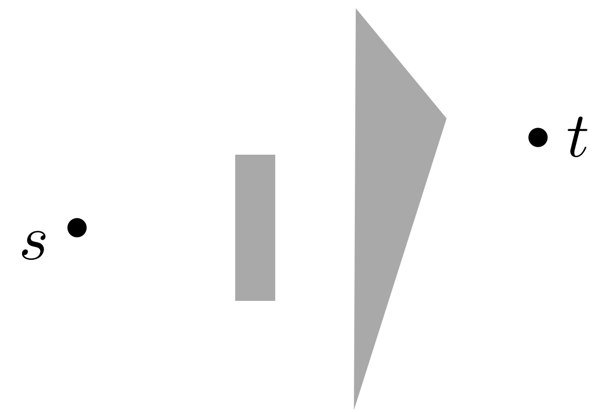
$\Rightarrow$ Path Planning

Paying for a toll bridge vs a longer route?

$\Rightarrow$ Geometric Network Augmentation

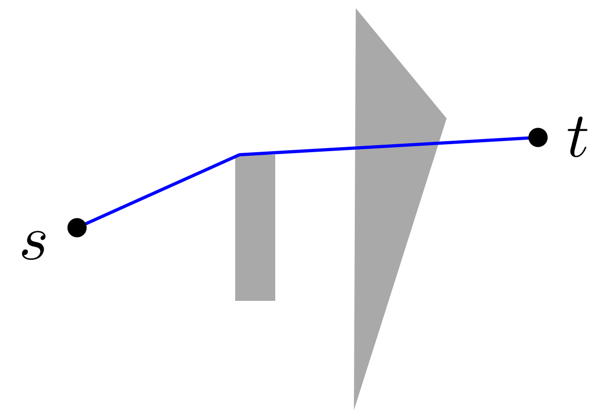
Removing Obstacles $\equiv$ Adding Edges to Visibility Graph

A Simple $O(kn^2)$ Algorithm



A Simple $O(kn^2)$ Algorithm

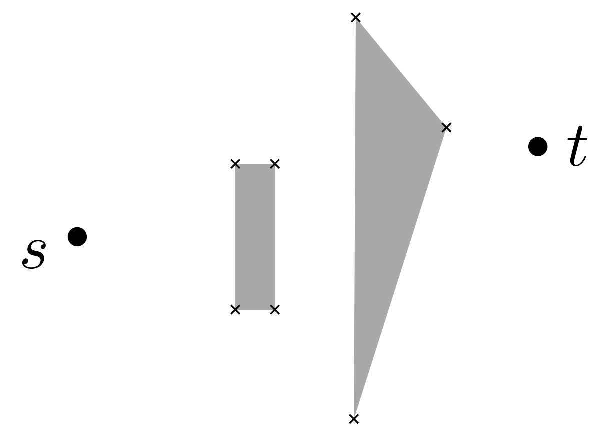
Key Idea: Shortest k-path turns only at obstacle vertices



A Simple $O(kn^2)$ Algorithm

Key Idea: Shortest k-path turns only at obstacle vertices

Construct k-visibility graph G_k

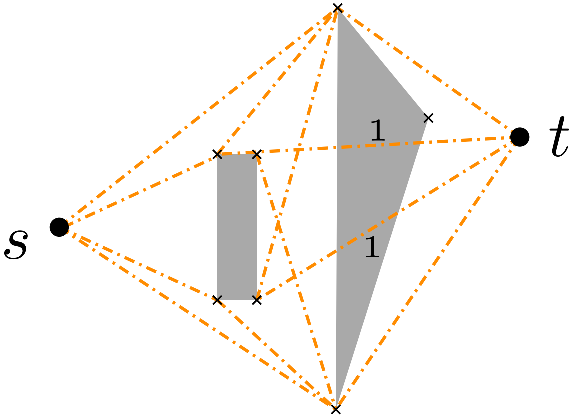


A Simple $O(kn^2)$ Algorithm

Key Idea: Shortest k-path turns only at obstacle vertices

Construct k-visibility graph G_k

- $\overline{pq}$ is an edge if it crosses $\leq k$ obstacles
- Label edges with crossing number
- Weight of an edge is its Euclidean length



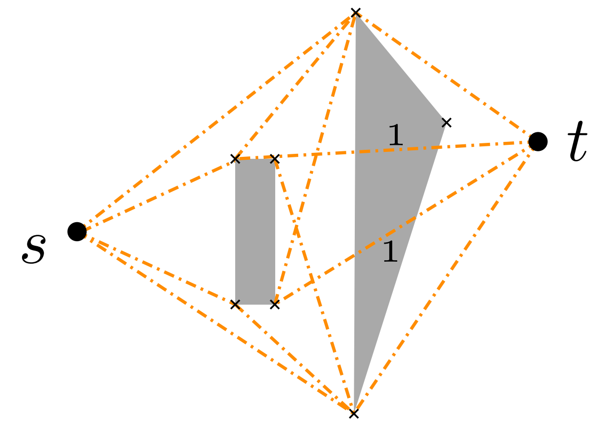
$O(n)$ vertices; $O(n + h^2)$ edges

A Simple $O(kn^2)$ Algorithm

Key Idea: Shortest k-path turns only at obstacle vertices

Construct k-visibility graph G_k

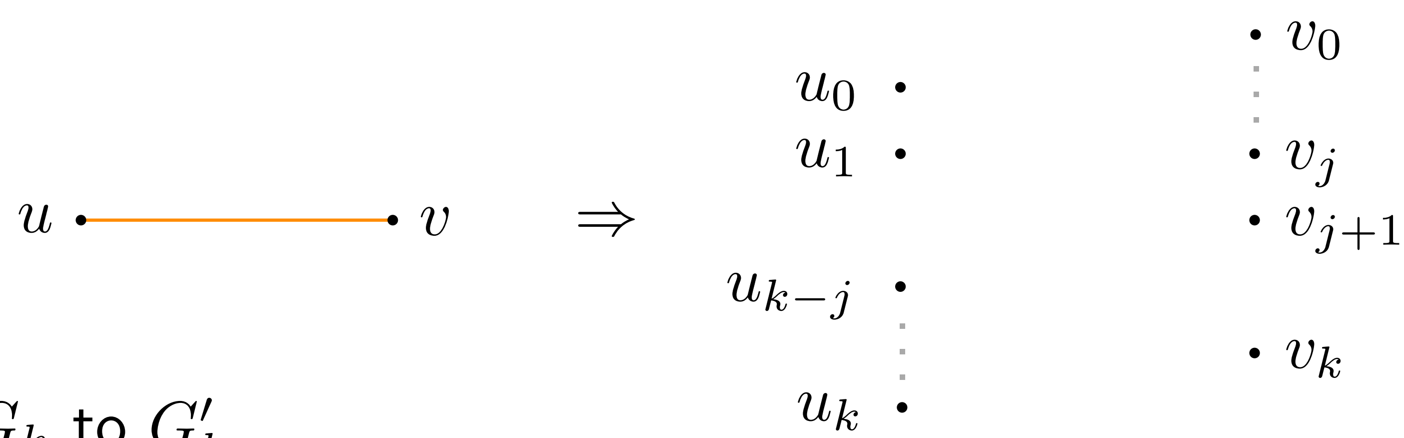
- $\overline{pq}$ is an edge if it crosses $\leq k$ obstacles
- Label edges with crossing number
- Weight of an edge is its Euclidean length



$O(n)$ vertices; $O(n + h^2)$ edges

Find shortest s–t path such that sum of labels on its edges is $\leq k$

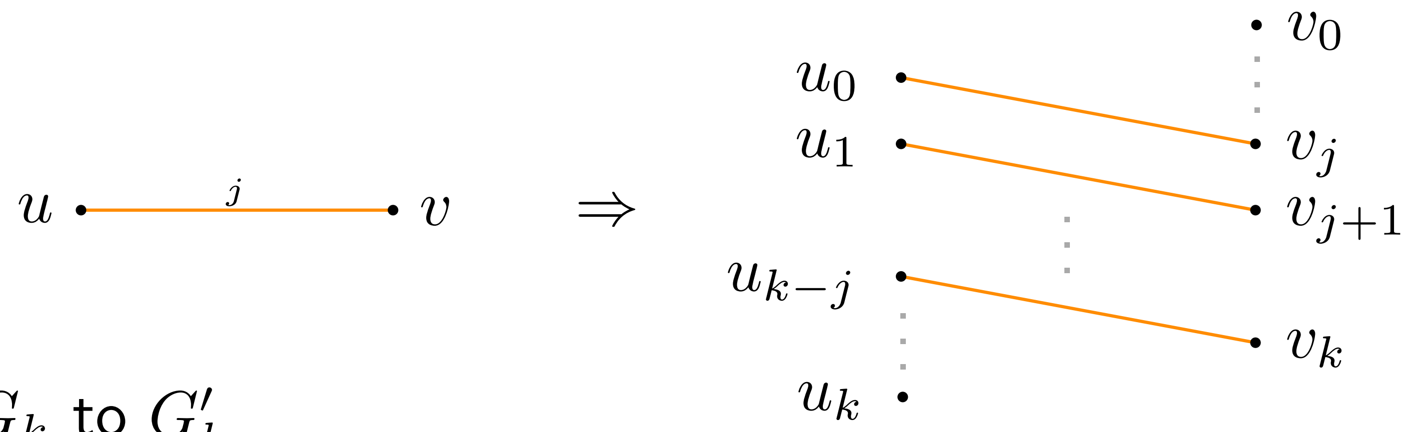
A Simple $O(kn^2)$ Algorithm



Transform G_k to G'_k

– k copies of each vertex

A Simple $O(kn^2)$ Algorithm

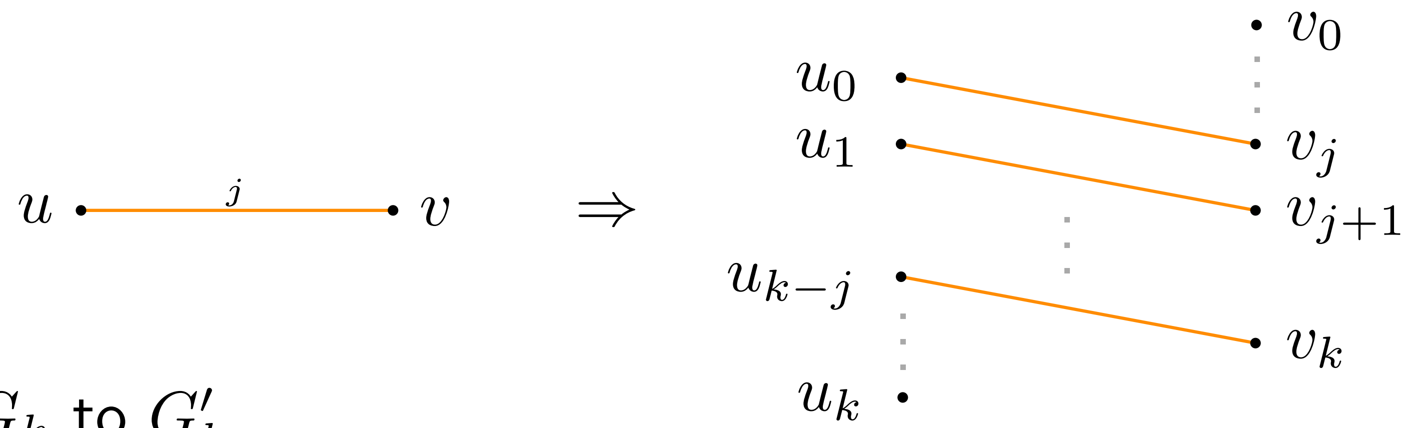


Transform G_k to G'_k

- k copies of each vertex
- For edge (u, v) with j crossings

  Add edges $(u_0, v_j), (u_1, v_{j+1})$ $\ldots$ (u_{k-j}, v_k)
- Connect s, t to all respective copies in G'_k

A Simple $O(kn^2)$ Algorithm



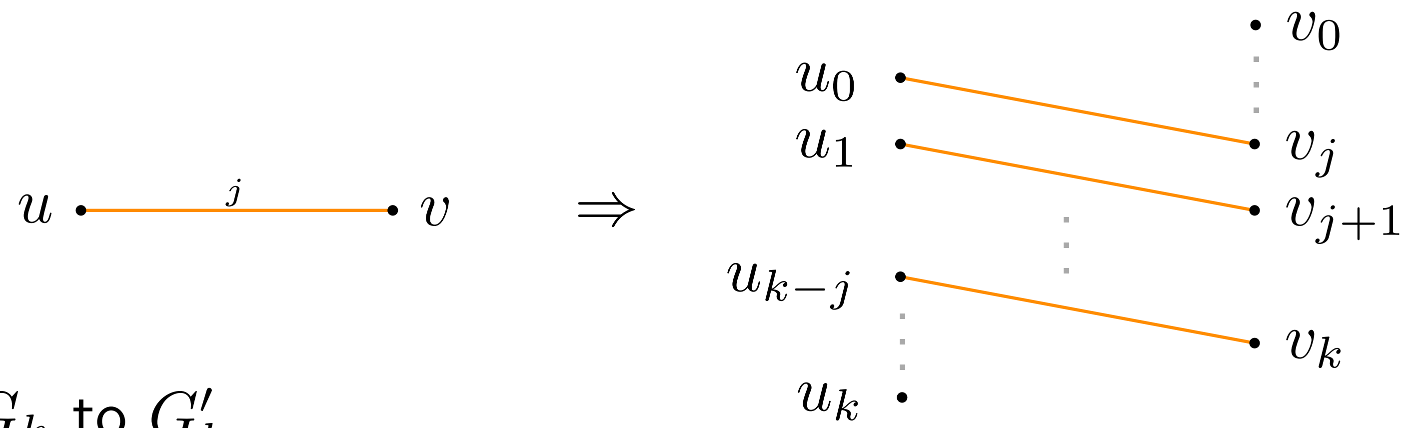
Transform G_k to G'_k

- k copies of each vertex
- For edge (u, v) with j crossings

  Add edges $(u_0, v_j), (u_1, v_{j+1}) \ \ldots \ (u_{k-j}, v_k)$
- Connect s, t to all respective copies in G'_k

Find shortest path from s to t in this transformed graph

A Simple $O(kn^2)$ Algorithm



Transform G_k to G'_k

- k copies of each vertex
- For edge (u, v) with j crossings

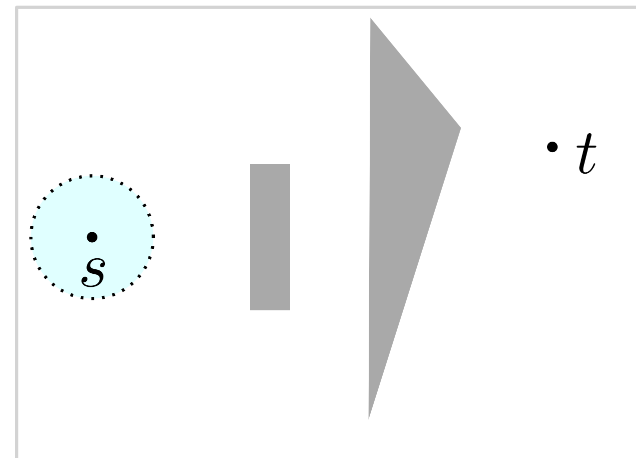
  Add edges $(u_0, v_j), (u_1, v_{j+1}) \;\ldots\; (u_{k-j}, v_k)$
- Connect s, t to all respective copies in G'_k

Find shortest path from s to t in this transformed graph

$O(kn^2)$ worst case using Dijkstra's Algorithm

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s

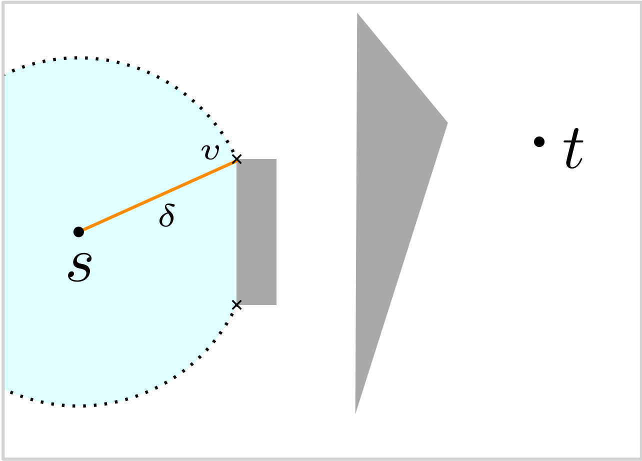


Wavefront at time $T \equiv$ all points at distance T from s

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s

– Wavefront turns at obstacle vertices



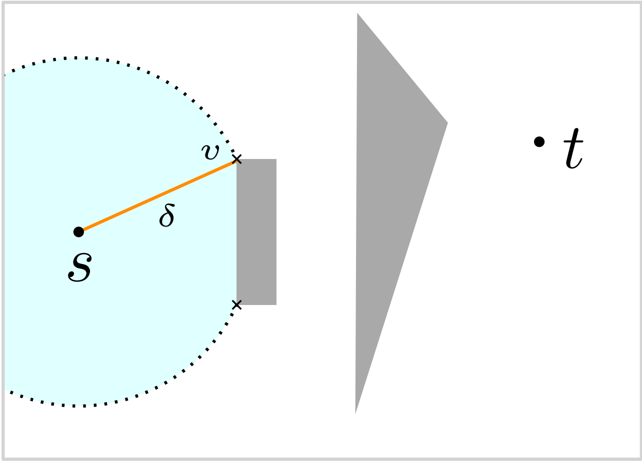
Adds new wavelet identified by its source $(v\,,\,\delta)$

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s

– Wavefront turns at obstacle vertices

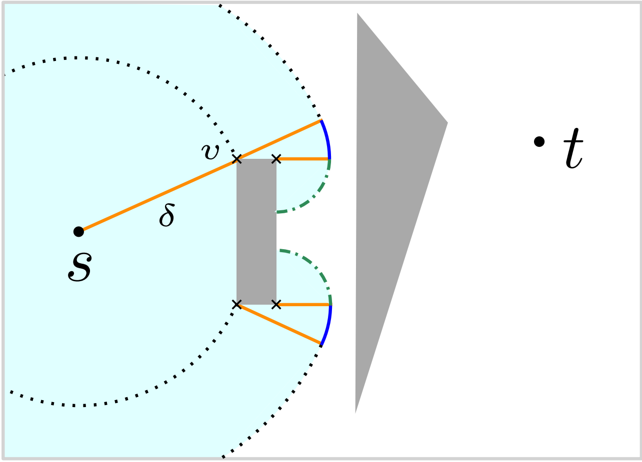
– Reaches t by a shortest path



Adds new wavelet identified by its source $(v\,,\,\delta)$

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s



– Wavefront turns at obstacle vertices

– Reaches t by a shortest path

– Challenge is to keep track of events

Wavelet-Obstacle collisions

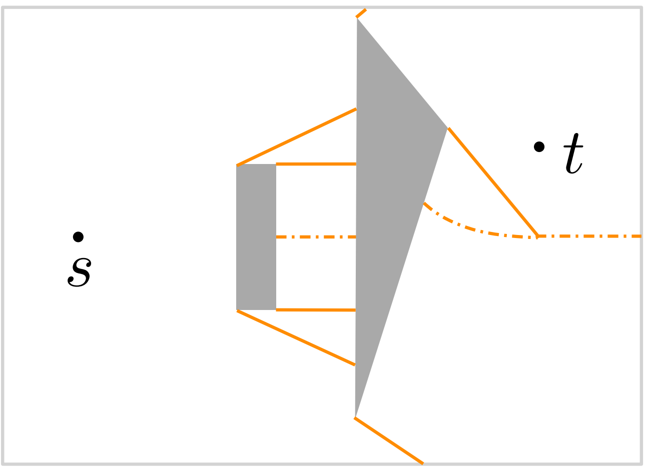
Wavelet-Wavelet collisions

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s

- Wavefront turns at obstacle vertices
- Reaches t by a shortest path
- Challenge is to keep track of events

  Wavelet-Obstacle collisions
  Wavelet-Wavelet collisions



Shortest Path Map (SPM)

- Computes a planar subdivision of P : Shortest Path Map

$O(n \log n)$ algorithm known [Hershberger-Suri, '97]

Towards a faster Algorithm : Continuous Dijkstra

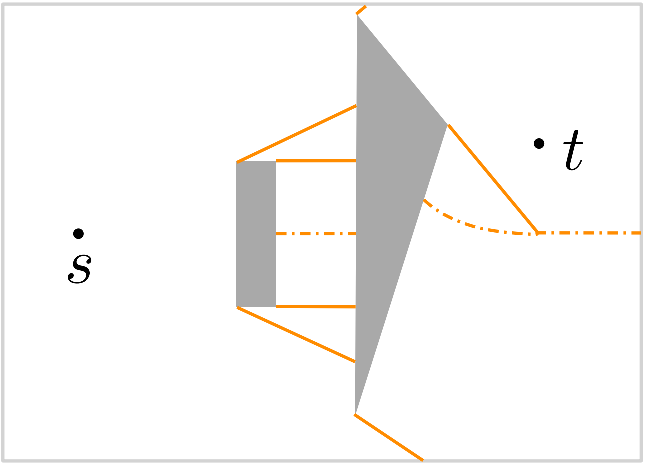
Simulates propagation of unit-speed wavefront starting at s

– Wavefront turns at obstacle vertices

– Reaches t by a shortest path

– Challenge is to keep track of events

Wavelet-Obstacle collisions
Wavelet-Wavelet collisions



Shortest Path Map (SPM)

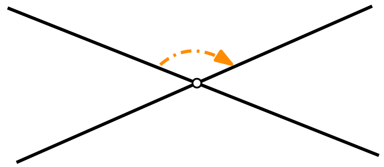
– Computes a planar subdivision of P : Shortest Path Map

$O(n \log n)$ algorithm known [Hershberger-Suri, '97]

Works because of one key property of shortest 0-paths

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s



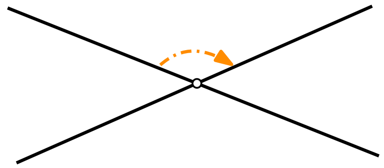
If they did, can locally reconnect to obtain a shorter path

Shortest 0-paths do not cross each other

Works because of one key property of shortest 0-paths

Towards a faster Algorithm : Continuous Dijkstra

Simulates propagation of unit-speed wavefront starting at s



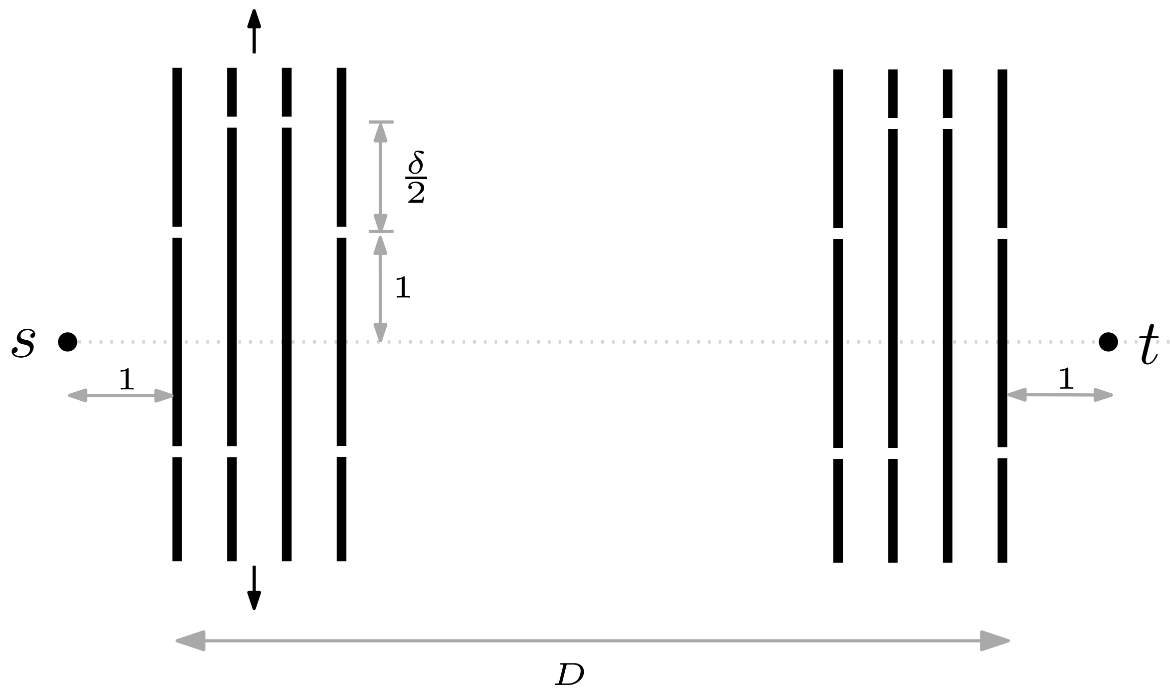
If they did, can locally reconnect to obtain a shorter path

Shortest 0-paths do not cross each other

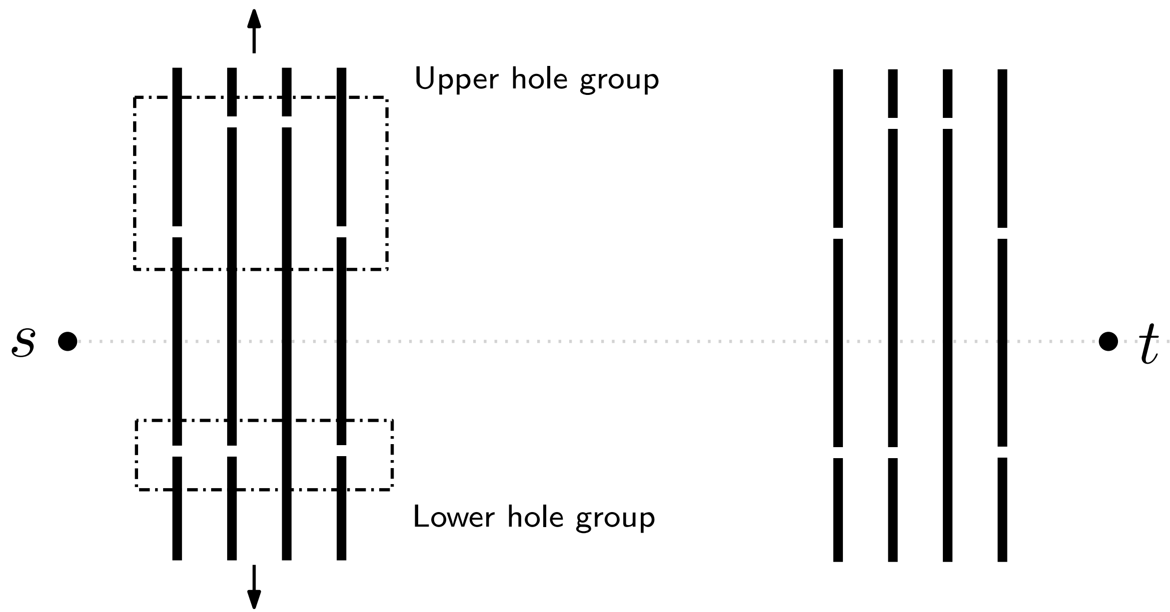
NOT true for shortest k-paths for $k > 0$

Works because of one key property of shortest 0-paths

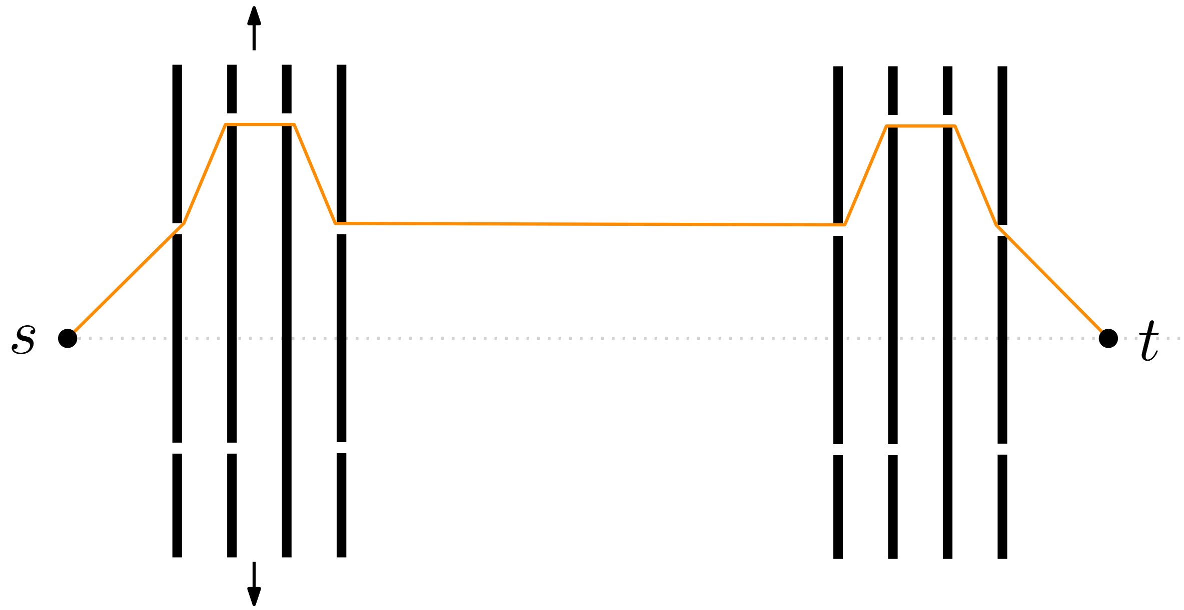
Shortest k-Paths may Cross!



Shortest k-Paths may Cross!

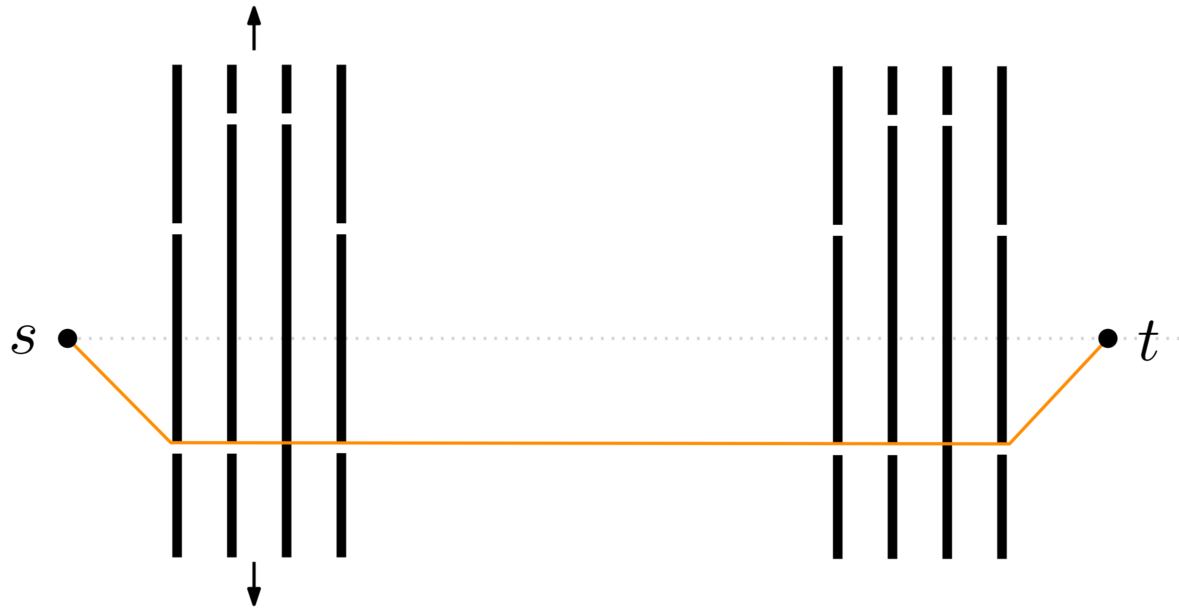


Shortest k-Paths may Cross!



- Shortest 0-path: $D + 2\sqrt{2} + 2\delta$

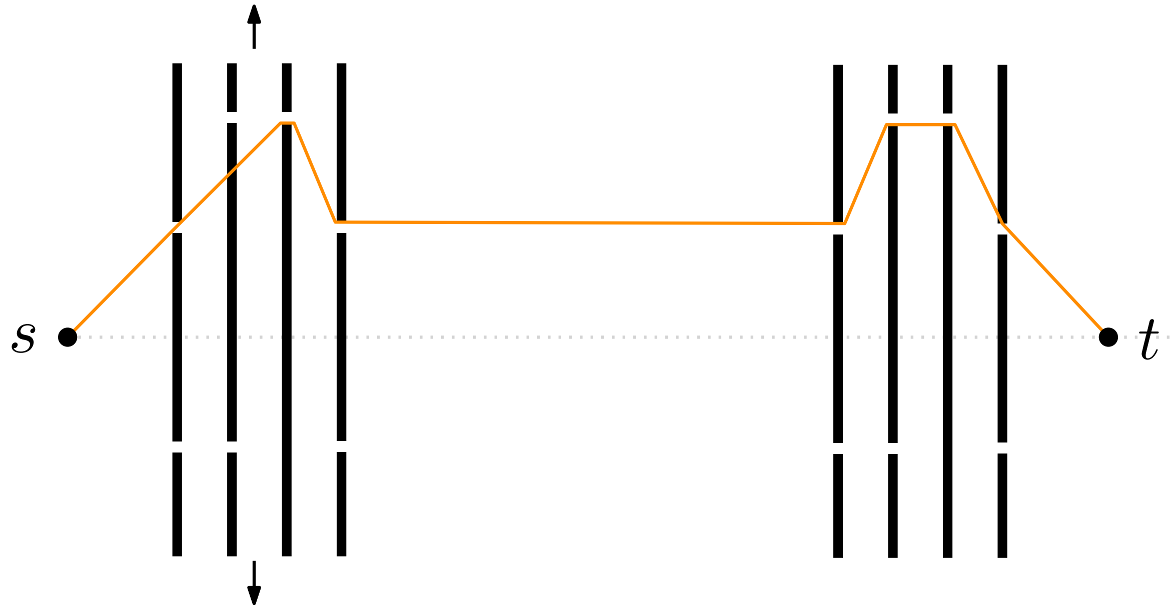
Shortest k-Paths may Cross!



Shortest 0-path: $D + 2\sqrt{2} + 2\delta$

➨ Shortest 2-path: $D + 2\sqrt{2}$

Shortest k-Paths may Cross!

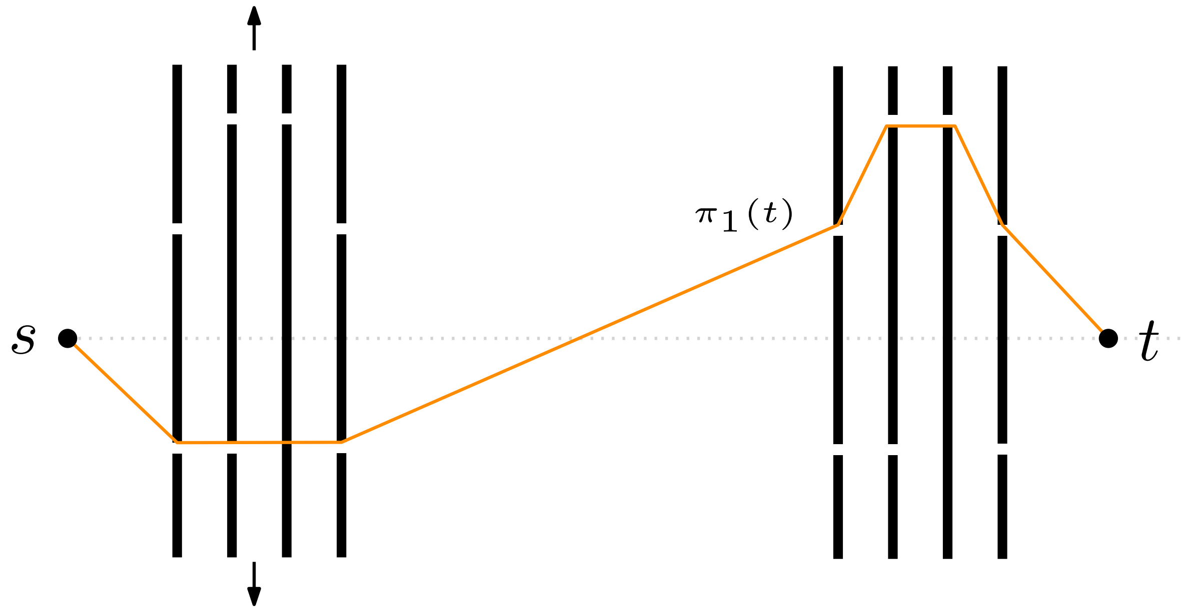


Shortest 0-path: $D + 2\sqrt{2} + 2\delta$

Shortest 2-path: $D + 2\sqrt{2}$

➡ 1-path crossing obstacle in upper group $> D + 2\sqrt{2} + 3\delta/2$

Shortest k-Paths may Cross!



Setting $D = 10$, $\delta = 0.4$ makes this path shortest 1-path

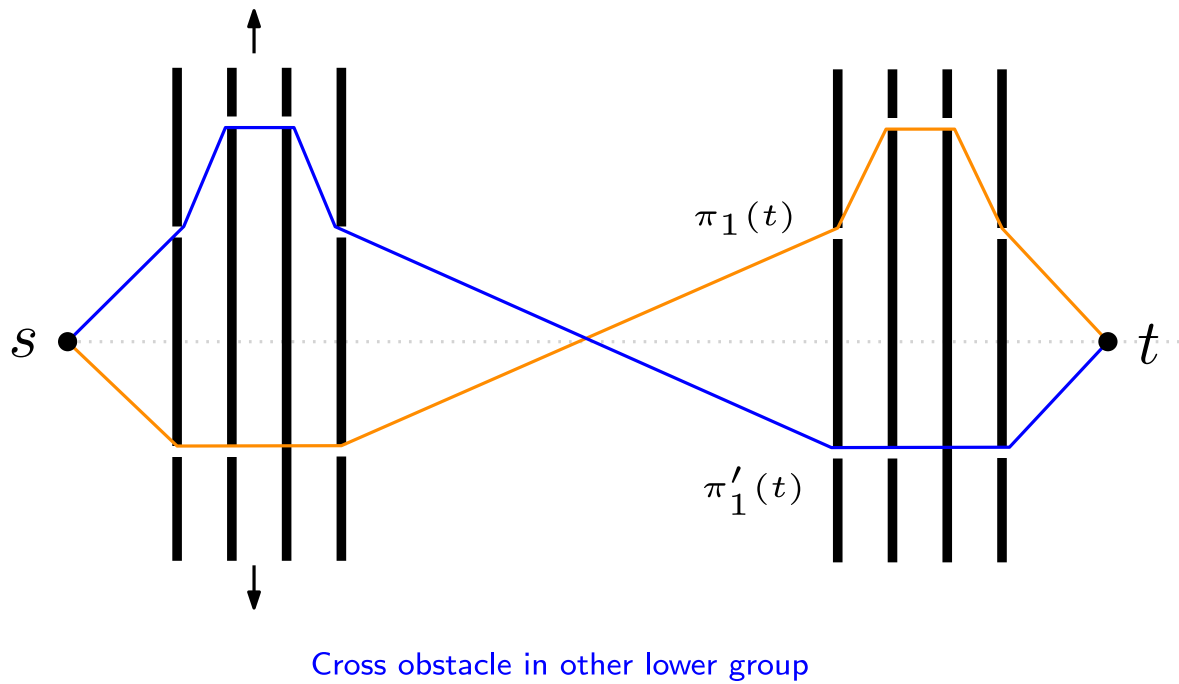
Shortest 0-path: $D + 2\sqrt{2} + 2\delta$

Shortest 2-path: $D + 2\sqrt{2}$

1-path crossing obstacle in upper group $> D + 2\sqrt{2} + 3\delta/2$

➨ Shortest 1-path crossing obstacle in lower group : $2\sqrt{2} + \sqrt{D^2 + 4} + \delta$

Shortest k-Paths may Cross!



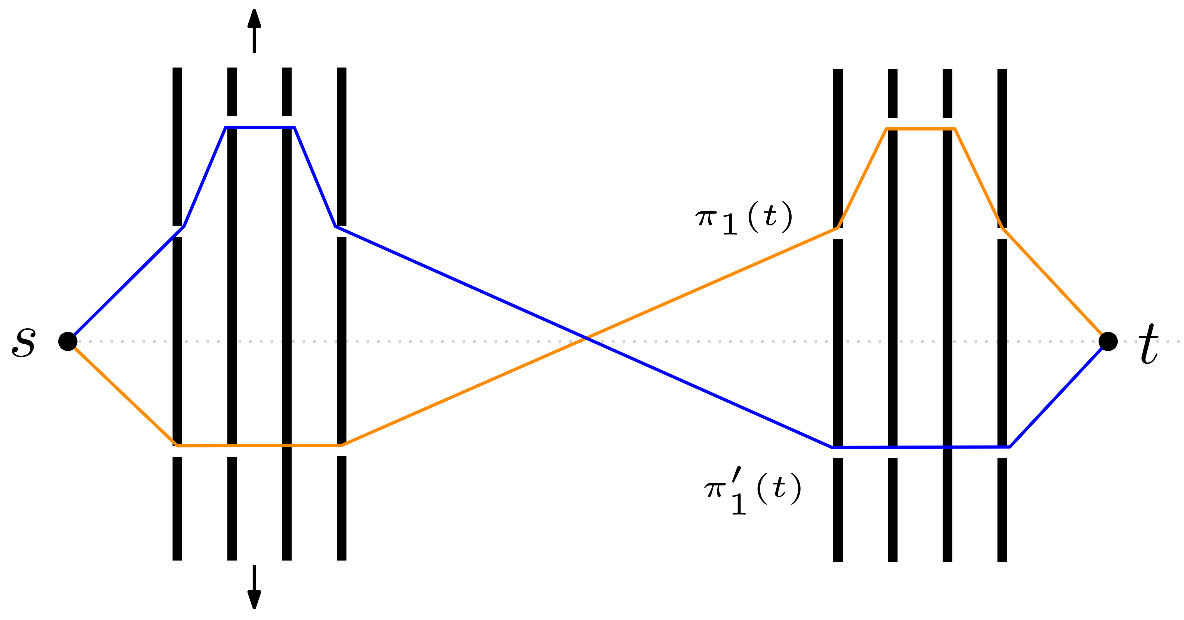
Shortest 0-path: $D + 2\sqrt{2} + 2\delta$

Shortest 2-path: $D + 2\sqrt{2}$

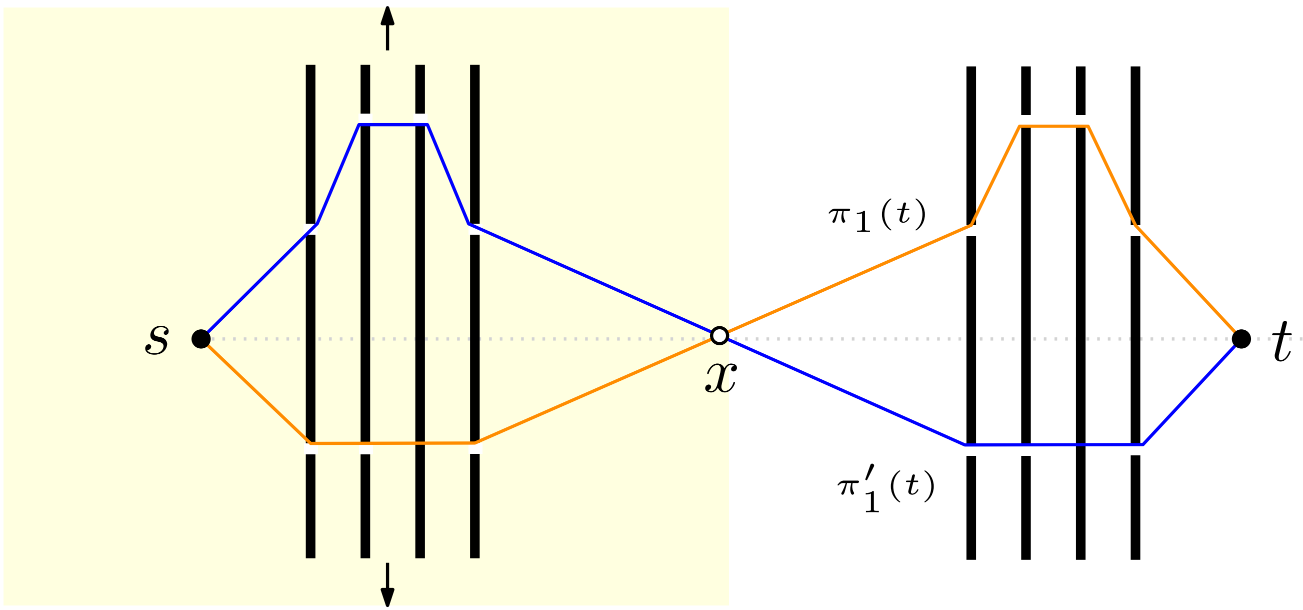
1-path crossing obstacle in upper group $> D + 2\sqrt{2} + 3\delta/2$

➡ Shortest 1-path crossing obstacle in lower group : $2\sqrt{2} + \sqrt{D^2 + 4} + \delta$

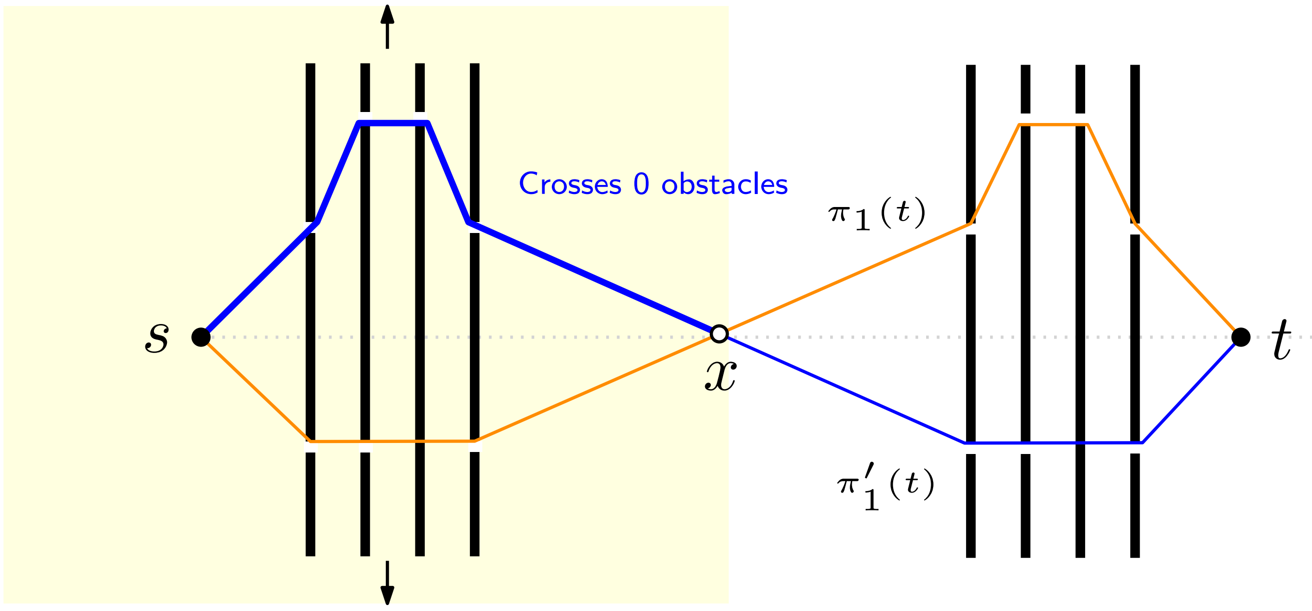
Shortest k-Paths may Cross!



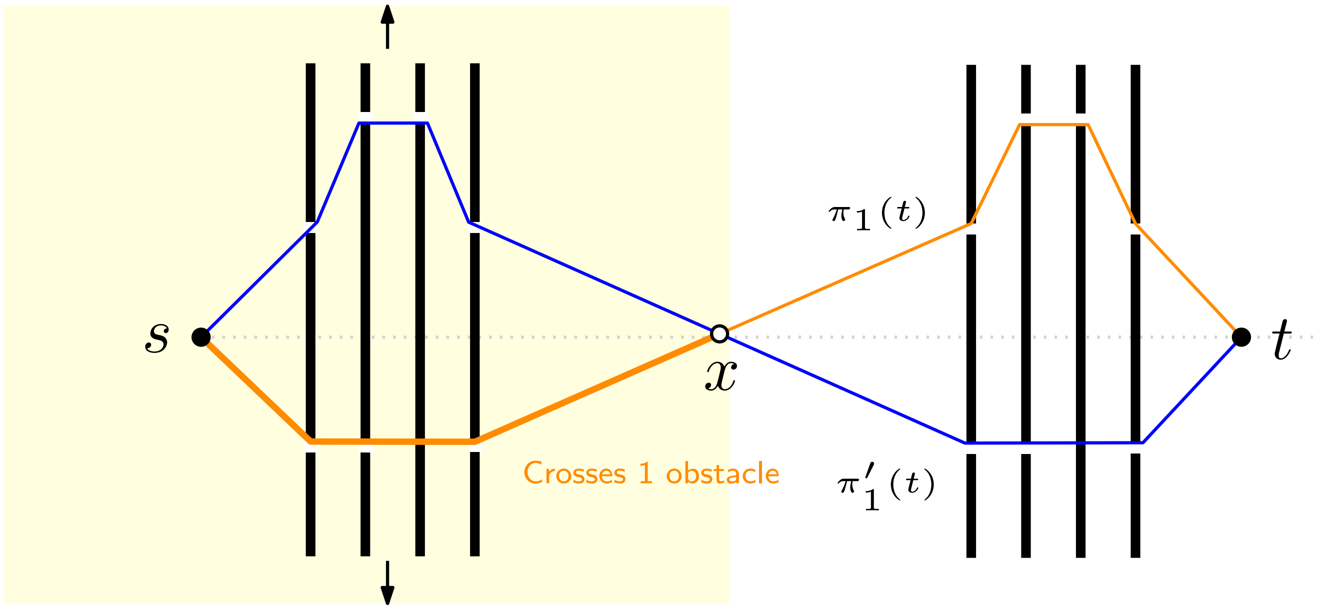
Shortest k-Paths may Cross!



Shortest k-Paths may Cross!



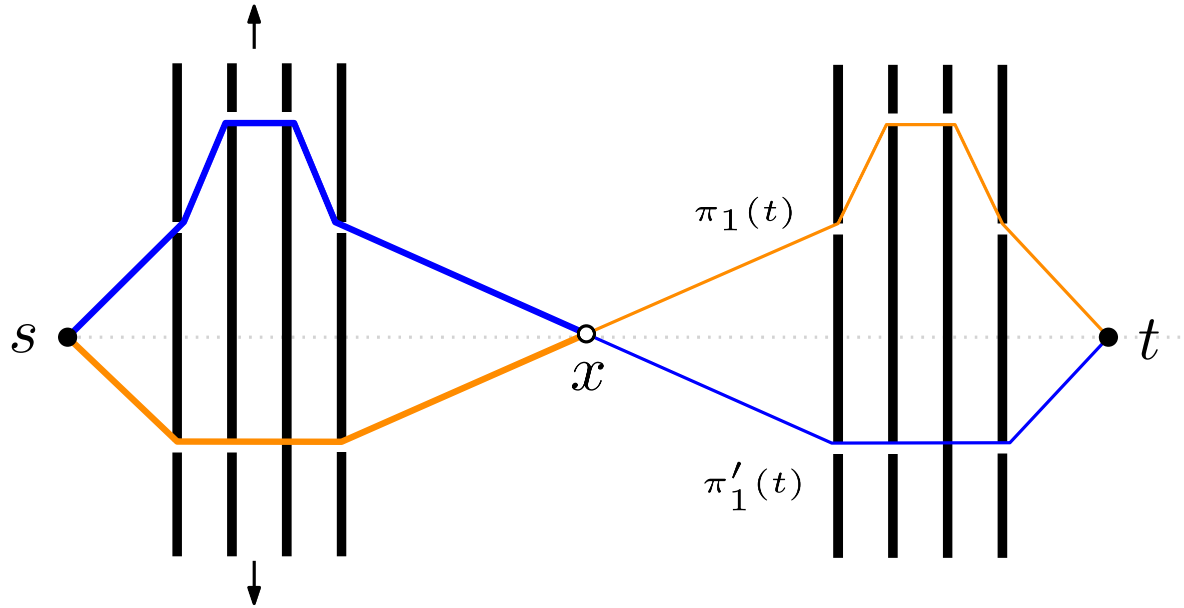
Shortest k-Paths may Cross!



Observation

Prefix of crossing k-paths cross different number of obstacles

Shortest k-Paths may Cross!



Observation

Prefix of crossing k-paths cross different number of obstacles

k-Paths as Non-crossing Subpaths

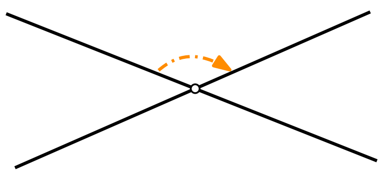
Prefix count of a k-path π_k at some point p on π_k

$\equiv$ Number of obstacles crossed by subpath of π_k until p

k-Paths as Non-crossing Subpaths

Prefix count of a k-path π_k at some point p on π_k

$\equiv$ Number of obstacles crossed by subpath of π_k until p



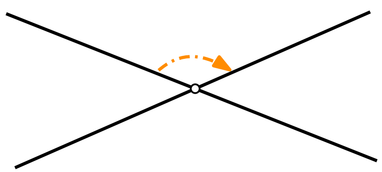
If they did, can locally reconnect to obtain a shorter k-path

Subpaths with same prefix count do not cross!

k-Paths as Non-crossing Subpaths

Prefix count of a k-path π_k at some point p on π_k

$\equiv$ Number of obstacles crossed by subpath of π_k until p



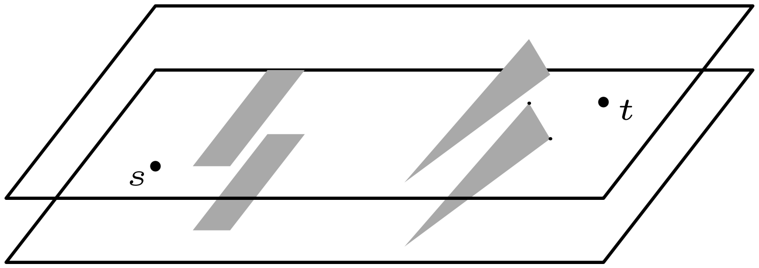
If they did, can locally reconnect to obtain a shorter k-path

Subpaths with same prefix count do not cross!

Use this to 'separate' relevant k-paths with different prefix counts and apply Continuous Dijkstra.

k-Level Garage with Elevators

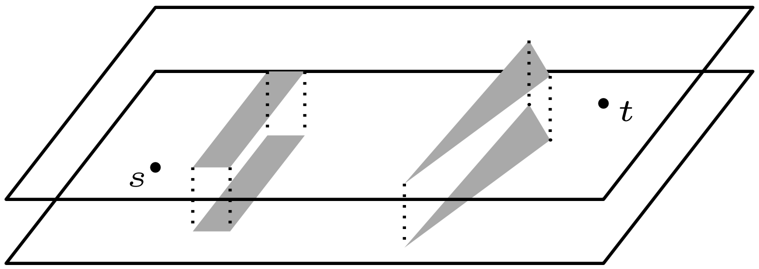
– Stack $(k+1)$ copies of P



k-Level Garage with Elevators

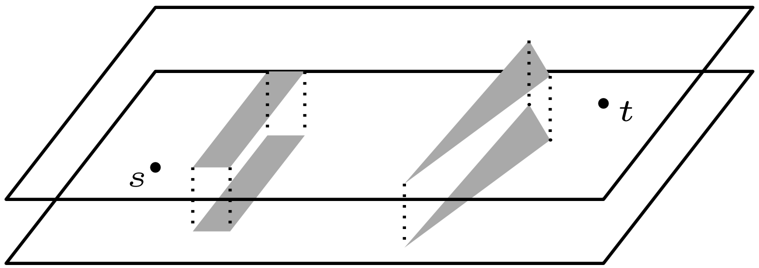
– Stack $(k+1)$ copies of P

Connect at corresponding vertices



k-Level Garage with Elevators

– Stack $(k+1)$ copies of P
  Connect at corresponding vertices

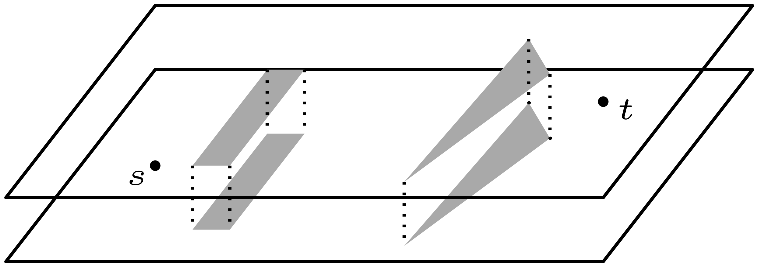


– Paths ascend floors by going inside obstacles (elevators)

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



– Paths ascend floors by going inside obstacles (elevators)

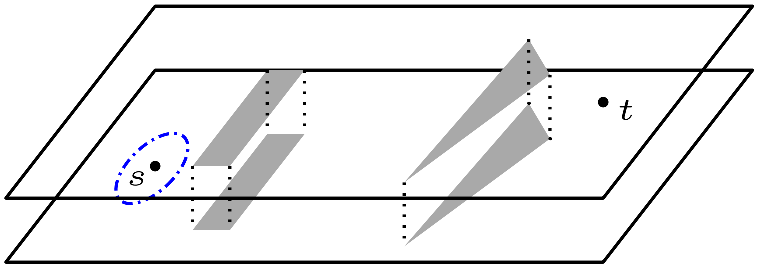
– Floors correspond to prefix counts

All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

k-Level Garage with Elevators

- Stack $(k+1)$ copies of P

  Connect at corresponding vertices



- Paths ascend floors by going inside obstacles (elevators)

- Floors correspond to prefix counts

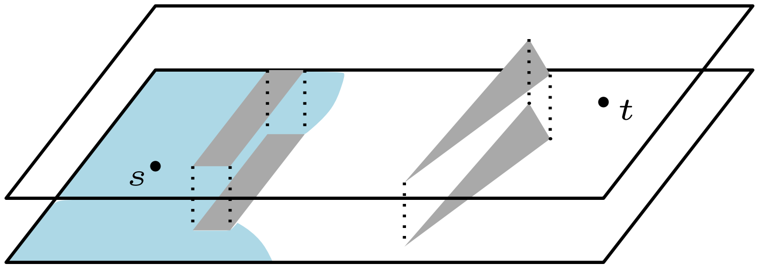
  All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

  Propagate wavefronts across floors starting at s

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



– Paths ascend floors by going inside obstacles (elevators)

– Floors correspond to prefix counts

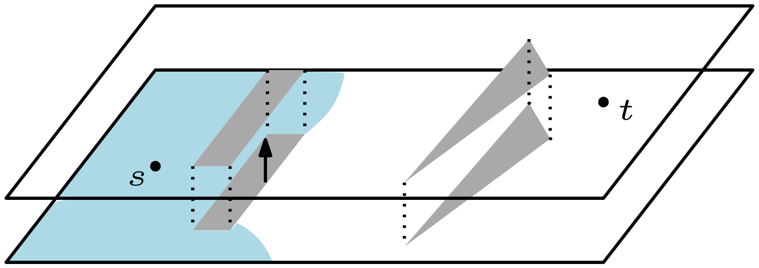
All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

Propagate wavefronts across floors starting at s

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



Wavelets move up floors by entering obstacles

– Paths ascend floors by going inside obstacles (elevators)

– Floors correspond to prefix counts

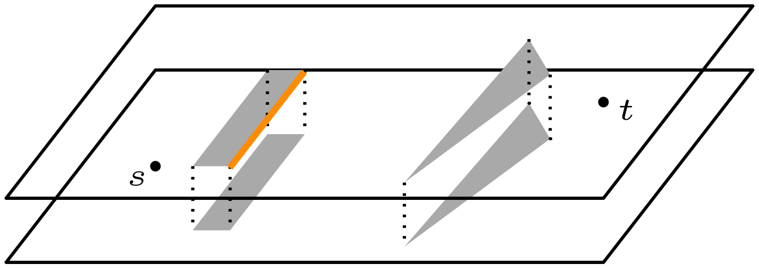
All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

Propagate wavefronts across floors starting at s

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



New 'sources' added at next level

– Paths ascend floors by going inside obstacles (elevators)

– Floors correspond to prefix counts

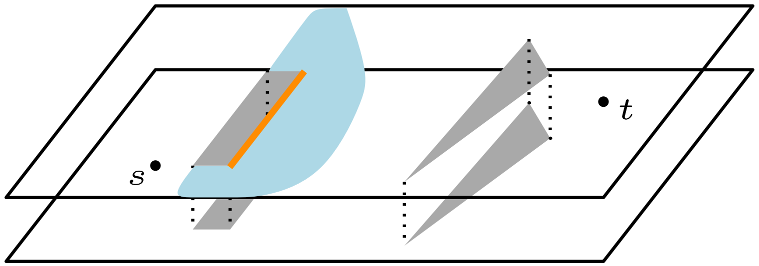
All paths reaching floor i have the prefix count i $\Rightarrow$ non-crossing

Propagate wavefronts across floors starting at s

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



Continue wavefront propagation as usual

– Paths ascend floors by going inside obstacles (elevators)

– Floors correspond to prefix counts

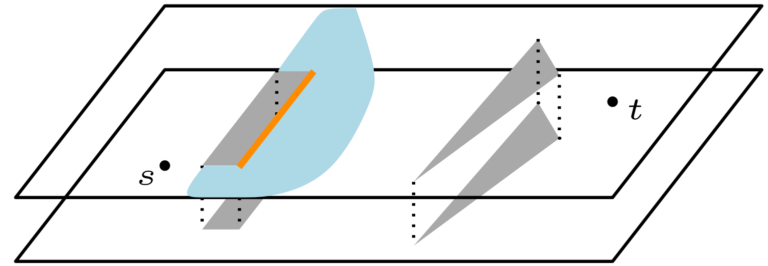
All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

Propagate wavefronts across floors starting at s

k-Level Garage with Elevators

– Stack $(k+1)$ copies of P

Connect at corresponding vertices



Continue wavefront propagation as usual

– Paths ascend floors by going inside obstacles (elevators)

– Floors correspond to prefix counts

All paths reaching floor i have the prefix count $i \Rightarrow$ non-crossing

Propagate wavefronts across floors starting at s

– Planar subdivision at floor k is shortest $(=k)$-path map $SPM_{=k}$.

k-Level Garage with Elevators

k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

Why?

k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

Why? Path Decomposition

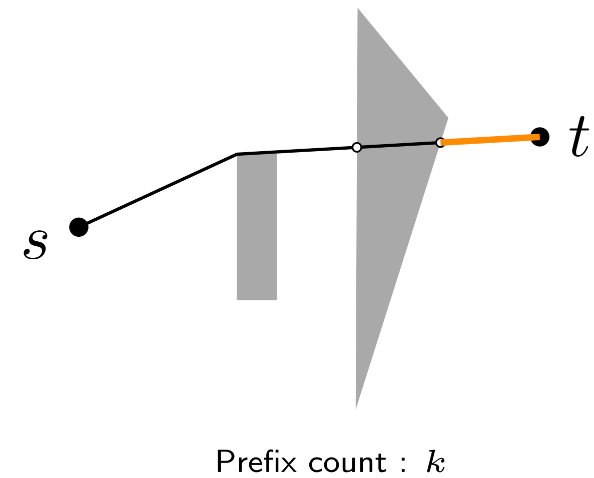
k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

Why? Path Decomposition

Shortest k-path can be decomposed into

– Shortest 0-Path



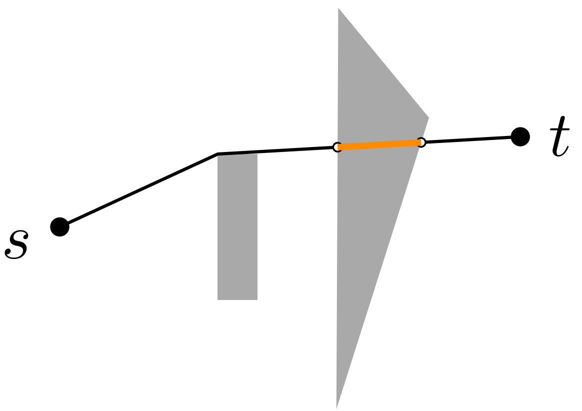
k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

Why? Path Decomposition

Shortest k-path can be decomposed into

- Shortest 0-Path
- Segment inside an obstacle



Prefix count : $(k - 1)$

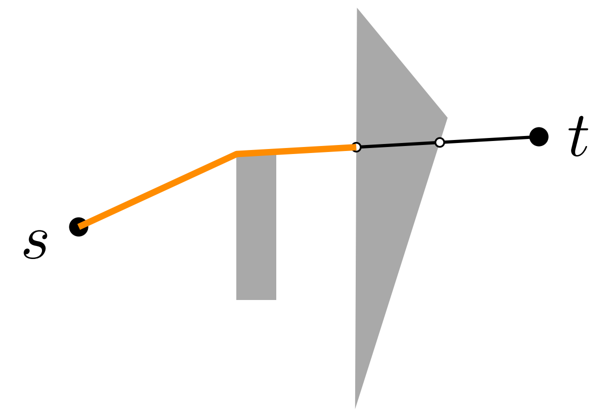
k-Level Garage with Elevators

For wavefront propagation at floor i, sources from floor $i-1$ suffice

Why? Path Decomposition

Shortest k-path can be decomposed into

- Shortest 0-Path
- Segment inside an obstacle
- Shortest $(k-1)$-path



Decompose recursively into $2k-1$ disjoint subpaths

Computing Shortest k-path map (SPM_k)

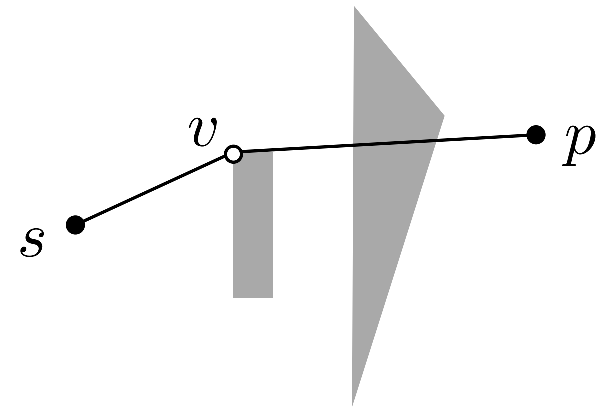
Partition of *free space* into regions with same k-predecessor

Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k - i)$ obstacles

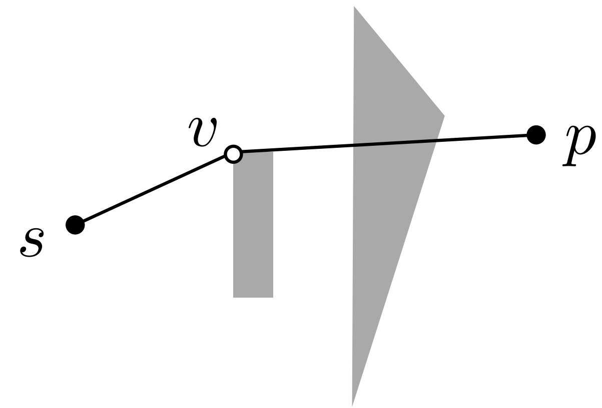


Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k - i)$ obstacles



$(v\,,\,0)$ is 1-predecessor of p

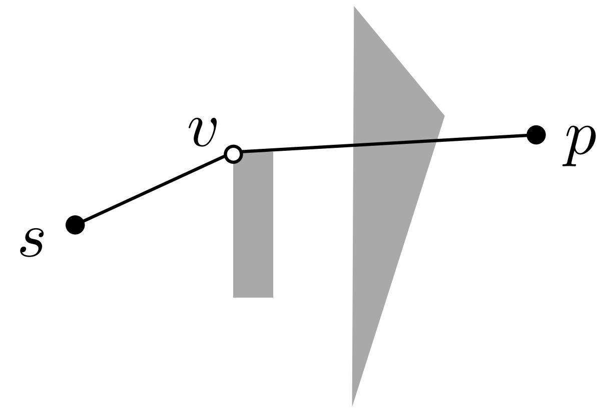
Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k-i)$ obstacles

$$|\pi_k(p)| \ = \ |\pi_i(v)| + |\overline{vp}|$$



$(v, 0)$ is 1-predecessor of p

Computing Shortest k-path map (SPM_k)

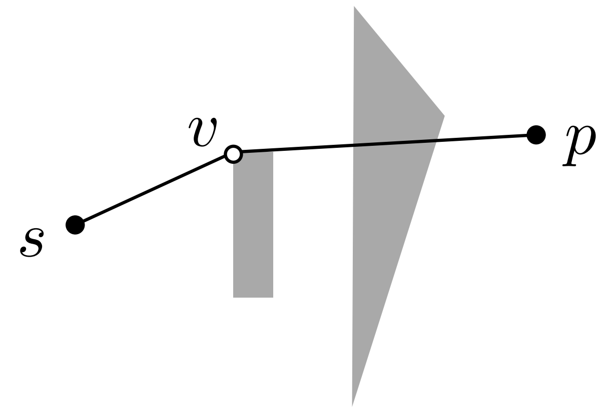
Partition of *free space* into regions with same k-predecessor

– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k - i)$ obstacles

$$|\pi_k(p)| \;=\; |\pi_i(v)| + |\overline{vp}|$$

Total $O(kn)$ predecessors



$(v\,,\,0)$ is 1-predecessor of p

Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

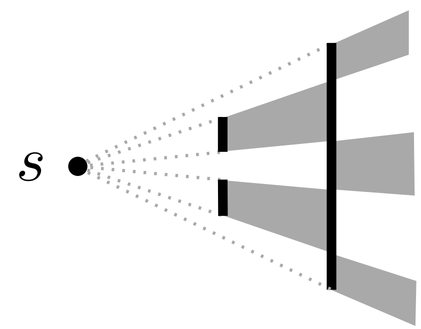
– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k - i)$ obstacles

$$|\pi_k(p)| \;=\; |\pi_i(v)| + |\overline{vp}|$$

Total $O(kn)$ predecessors

$\not\Rightarrow O(kn)$ regions



Multiple regions can have the same k-predecessor

Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

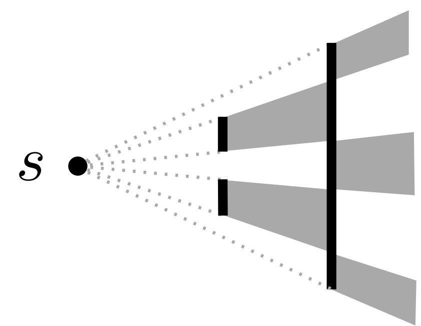
– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k-i)$ obstacles

$$|\pi_k(p)| \ = \ |\pi_i(v)| + |\overline{vp}|$$

Total $O(kn)$ predecessors

$\not\Rightarrow O(kn)$ regions



Multiple regions can have the same k-predecessor

– Comprises of two distinct regions

V_{k-1} region visible from s by crossing fewer than k obstacles

Computing Shortest k-path map (SPM_k)

Partition of *free space* into regions with same k-predecessor

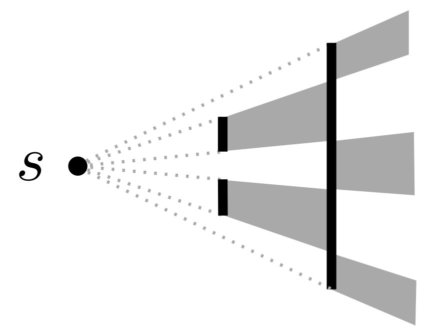
– Identify k-predecessor of p as (v, i)

v is adjacent to p on $\pi_k(p)$ and $\overline{vp}$ crosses $(k-i)$ obstacles

$$|\pi_k(p)| = |\pi_i(v)| + |\overline{vp}|$$

Total $O(kn)$ predecessors

$\not\Rightarrow O(kn)$ regions



Multiple regions can have the same k-predecessor

– Comprises of two distinct regions

V_{k-1} region visible from s by crossing fewer than k obstacles

$SPM_{=k}$ Rest of free space

Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

Uses algorithm by Hershberger and Suri for wavefront propagation

Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

Uses algorithm by Hershberger and Suri for wavefront propagation

Can be adapted to handle boundary sources

Computing SPM_k

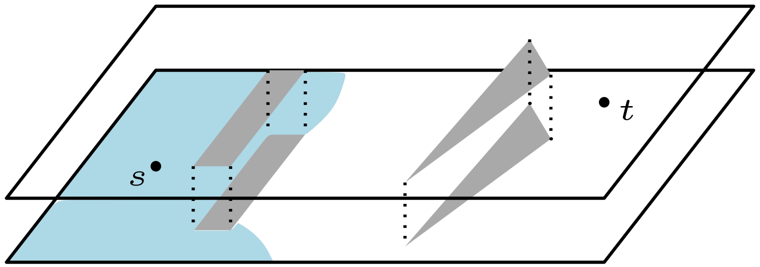
Computes both V_{k-1} and $SPM_{=k}$ one level at a time

Uses algorithm by Hershberger and Suri for wavefront propagation

Can be adapted to handle boundary sources

Propagate wavefront

$\Rightarrow$ At each garage floor



Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

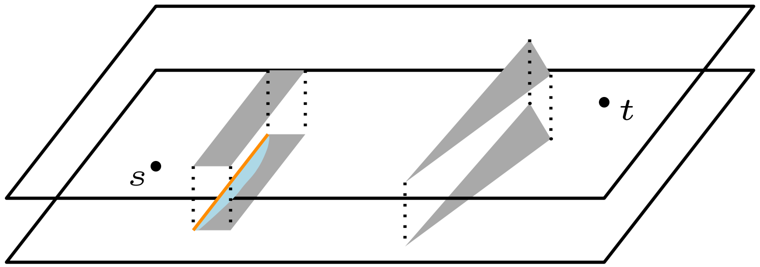
Uses algorithm by Hershberger and Suri for wavefront propagation

Can be adapted to handle boundary sources

Propagate wavefront

$\Rightarrow$ At each garage floor

$\Rightarrow$ Inside the obstacles (elavators)



Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

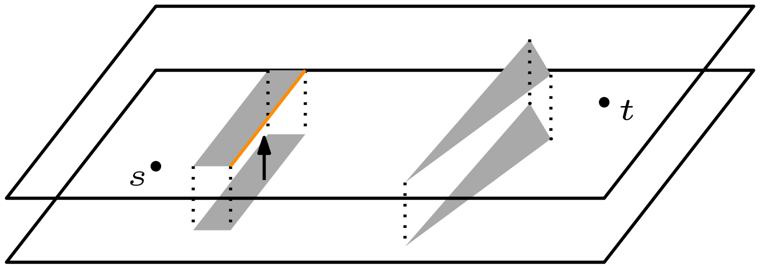
Uses algorithm by Hershberger and Suri for wavefront propagation

Can be adapted to handle boundary sources

Propagate wavefront

$\Rightarrow$ At each garage floor

$\Rightarrow$ Inside the obstacles (elavators)



Computing SPM_k

Computes both V_{k-1} and $SPM_{=k}$ one level at a time

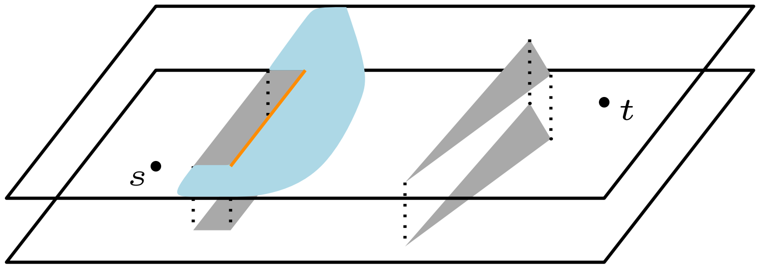
Uses algorithm by Hershberger and Suri for wavefront propagation

Can be adapted to handle boundary sources

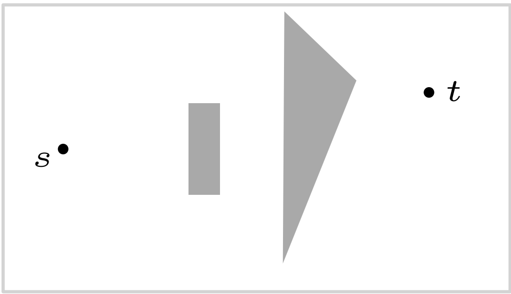
Propagate wavefront

$\Rightarrow$ At each garage floor

$\Rightarrow$ Inside the obstacles (elavators)



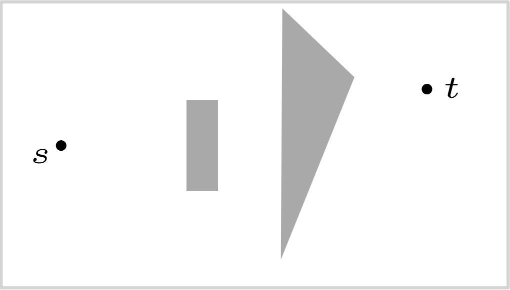
Computing SPM_k : Algorithm Outline



Computing SPM_k : Algorithm Outline

1. Set $\underline{M = \{s\}}$ and $V = \emptyset$.

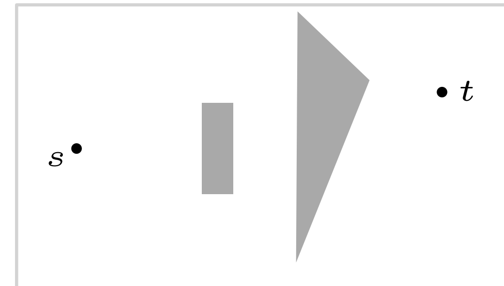
Sources passed to HS-algorithm



Computing SPM_k : Algorithm Outline

1. Set $M = \{s\}$ and $V = \emptyset$.

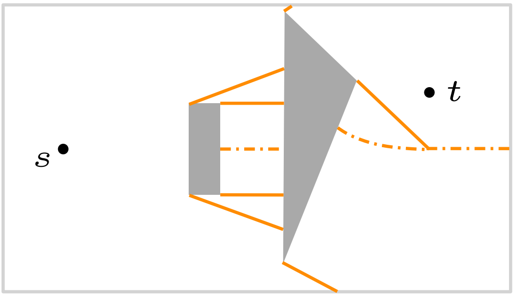
Visibility region V_{i-1}



Computing SPM_k : Algorithm Outline

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0



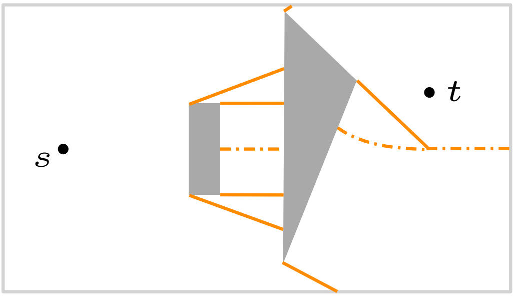
Sources claim intervals on domain boundary

Computing SPM_k : Algorithm Outline

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0

2. For $i \in 1, 2, \ldots, k$

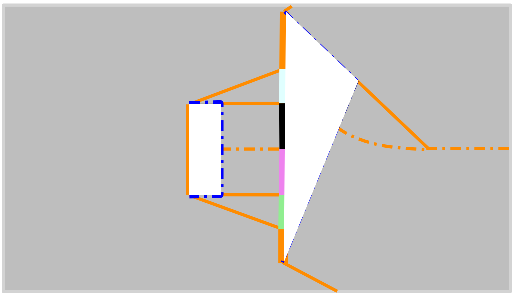


Sources claim intervals on domain boundary

Computing SPM_k : Algorithm Outline

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0

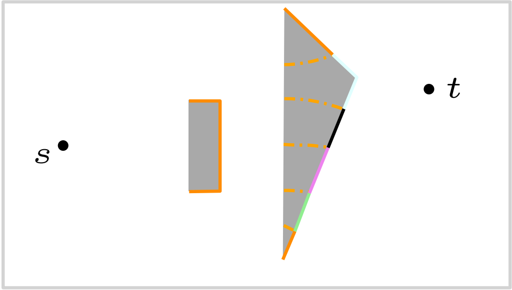


'Boundary sources' for propagation within obstacles

2. For $i \in 1, 2, \ldots, k$

   $\Rightarrow$ Propagate 'claims' in SPM_{i-1} inside the obstacles

Computing SPM_k : Algorithm Outline



Boundary sources for next level

1. Set $M = \{s\}$ and $V = \emptyset$.

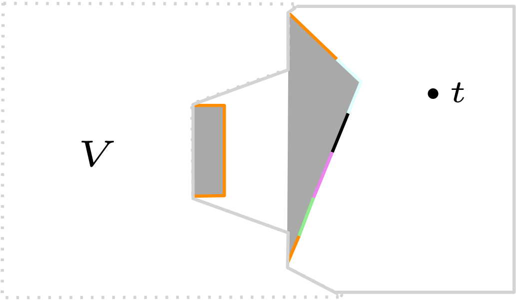
   Call HS-algorithm to compute SPM_0

2. For $i \in 1, 2, \ldots, k$

   $\Rightarrow$ Propagate 'claims' in SPM_{i-1} inside the obstacles

   Gives new boundary sources say M_{new}

Computing SPM_k : Algorithm Outline



$P = P - V$

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0

2. For $i \in 1, 2, \ldots, k$

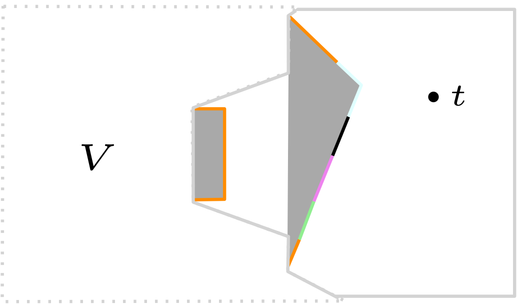
   $\Rightarrow$ Propagate 'claims' in SPM_{i-1} inside the obstacles

   Gives new boundary sources say M_{new}

   $\Rightarrow$ Drop regions of SPM_{i-1} with s as predecessor from P

   k-visible from s, include this region to V.

Computing SPM_k : Algorithm Outline



Continue propagation with new sources

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0

2. For $i \in 1, 2, \ldots, k$

   $\Rightarrow$ Propagate 'claims' in SPM_{i-1} inside the obstacles

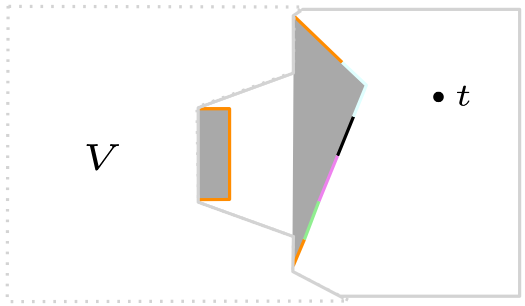
   Gives new boundary sources say M_{new}

   $\Rightarrow$ Drop regions of SPM_{i-1} with s as predecessor from P

   k-visible from s, include this region to V.

   $\Rightarrow$ With M as M_{new}, call HS-algorithm on P to compute $SPM_{=i}$.

Computing SPM_k : Algorithm Outline



Continue propagation with new sources

1. Set $M = \{s\}$ and $V = \emptyset$.

   Call HS-algorithm to compute SPM_0

2. For $i \in 1, 2, \ldots, k$

   $\Rightarrow$ Propagate 'claims' in SPM_{i-1} inside the obstacles

   Gives new boundary sources say M_{new}

   $\Rightarrow$ Drop regions of SPM_{i-1} with s as predecessor from P

   k-visible from s, include this region to V.

   $\Rightarrow$ With M as M_{new}, call HS-algorithm on P to compute $SPM_{=i}$.

3. At this point we have V as V_{k-1} and $SPM_{=k}$.

   Merge them to obtain SPM_k

Computing SPM_k : Running Time

If S_i is the size of SPM_i

$$\text{Total Running time} \quad = \quad \sum_{i\,=\,1,\,2,\,..k} O(S_i \log S_i)$$

Computing SPM_k : Running Time

If S_i is the size of SPM_i

$$\text{Total Running time} \quad = \quad \sum_{i\,=\,1,\,2,\,..k} \underline{O(S_i \log S_i)}$$

Calls HS-Algorithm

Computing SPM_k : Running Time

If S_i is the size of SPM_i

$$\text{Total Running time} \quad = \quad \sum_{i\,=\,1,\,2,\,..k} O(S_i \log S_i)$$

$$\leq \quad O(k \cdot S_k \log S_k)$$

Computing SPM_k : Running Time

If S_i is the size of SPM_i

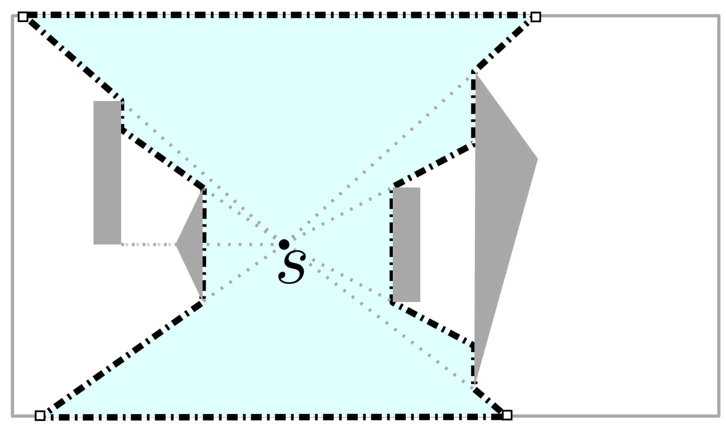
$$\text{Total Running time} = \sum_{i=1,2,..k} O(S_i \log S_i)$$

$$\leq O(k \cdot S_k \log S_k)$$

Compute an upper bound on size of SPM_k

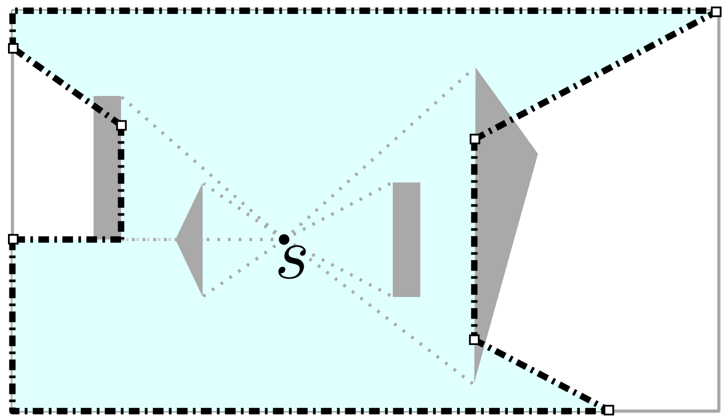
Bound sizes of V_{k-1} and $SPM_{=k}$

Complexity of k-visibility region V_k



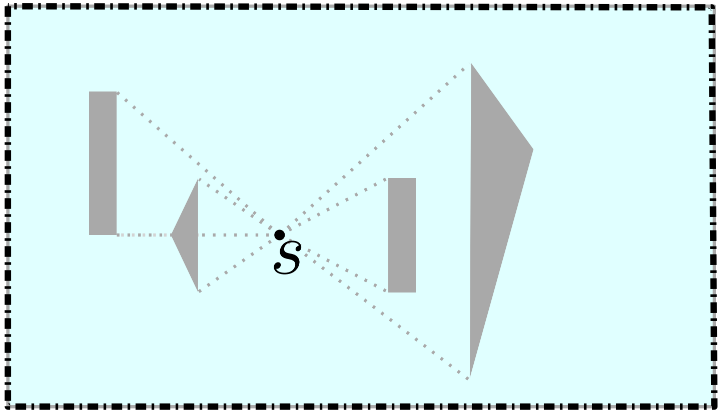
Region V_0

Complexity of k-visibility region V_k



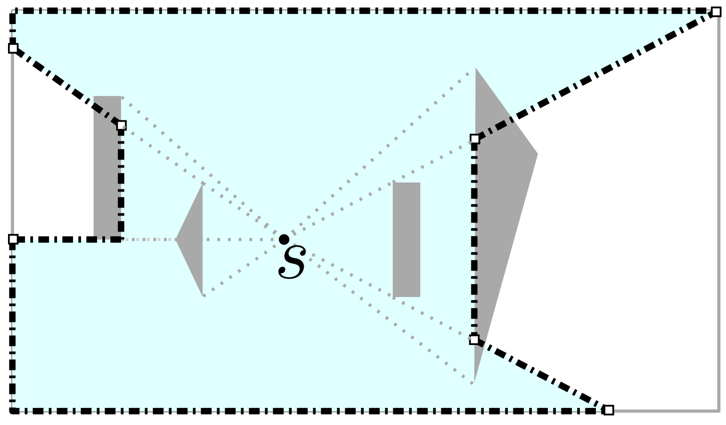
Region V_1

Complexity of k-visibility region V_k



Region V_2

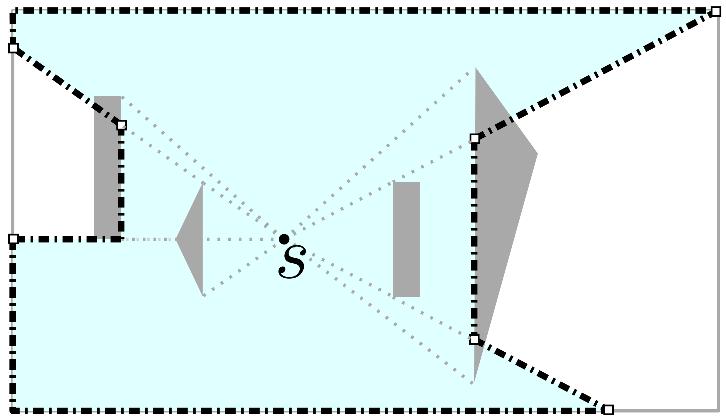
Complexity of k-visibility region V_k



The number of edges on boundary of V_k is $O(n+h)$

Vertex on ∂V_k is an obstacle vertex or projection of a tangent

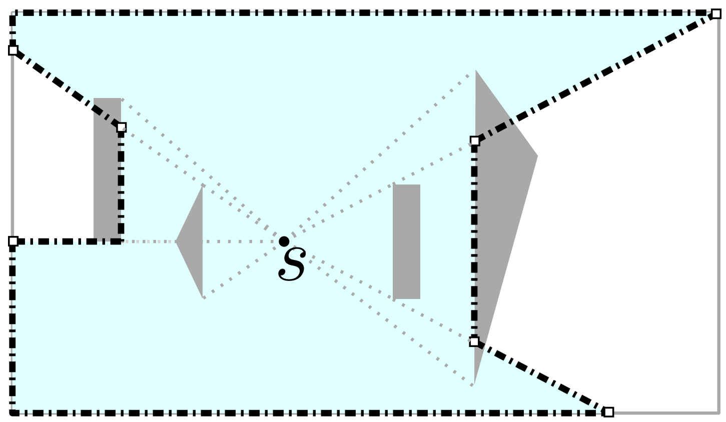
Complexity of k-visibility region V_k



The number of edges on boundary of V_k is $O(n + h)$

Vertex on ∂V_k is an $\underbrace{\text{obstacle vertex}}_{\text{at most } n}$ or $\underbrace{\text{projection of a tangent}}_{\text{at most } 2h}$

Complexity of k-visibility region V_k



The number of edges on boundary of V_k is $O(n+h)$

Vertex on ∂V_k is an $\underbrace{\text{obstacle vertex}}_{\text{at most } n}$ or $\underbrace{\text{projection of a tangent}}_{\text{at most } 2h}$

Total complexity summed over all ∂V_i for $0 \leq i \leq k$ is $O(n+hk)$

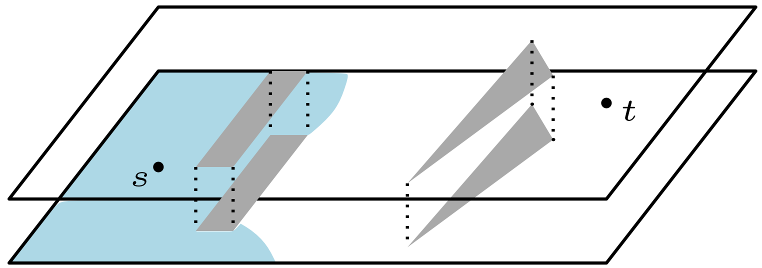
Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m+n)$

Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m+n)$

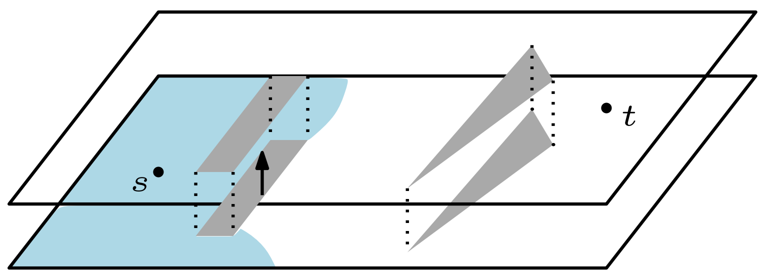
$SPM_{=k}$ may contain sources for wavelets from lower floor



Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m+n)$

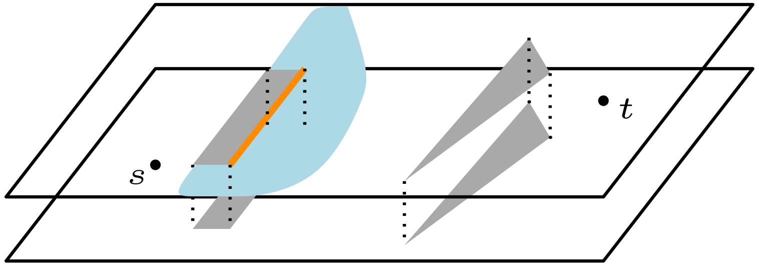
$SPM_{=k}$ may contain sources for wavelets from lower floor



Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m+n)$

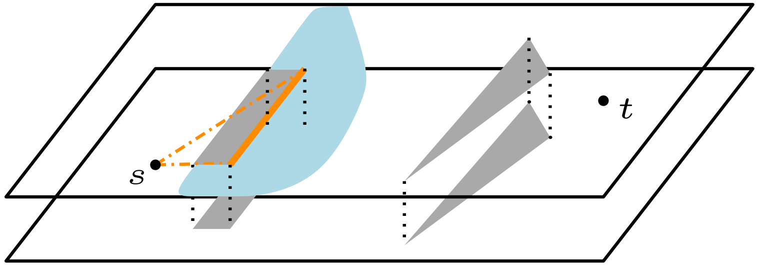
$SPM_{=k}$ may contain sources for wavelets from lower floor



Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m+n)$

$SPM_{=k}$ may contain sources for wavelets from lower floor



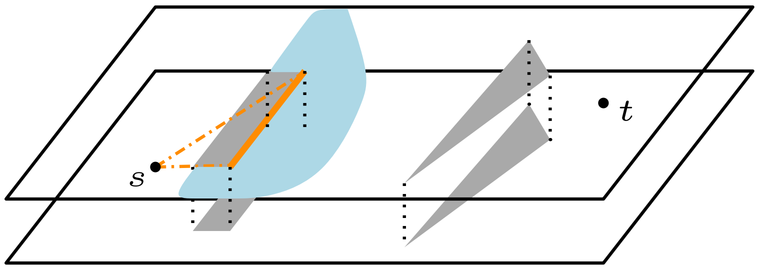
boundary sources : wavefront travel to obstacle boundary

by ascending one or more elevators

Complexity of $SPM_{=k}$

Complexity of a map with m sources and n vertices is $O(m + n)$

$SPM_{=k}$ may contain sources for wavelets from lower floor

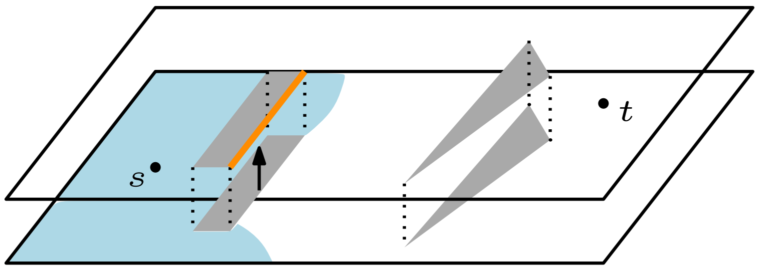


boundary sources : wavefront travel to obstacle boundary by ascending one or more elevators

Need to bound the number of such sources at each level

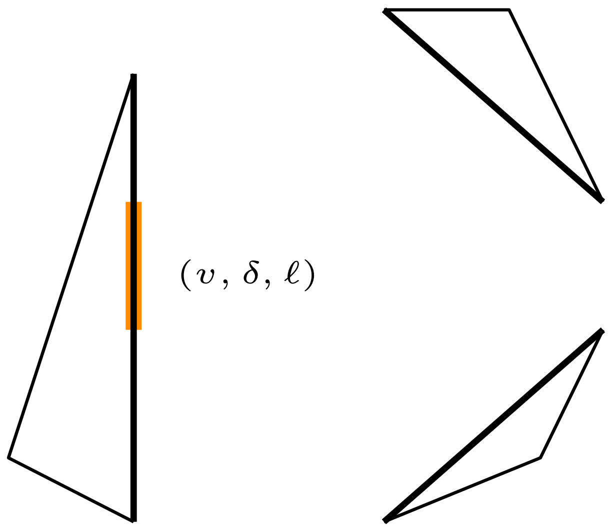
Complexity of $SPM_{=k}$: Boundary Sources

Boundary sources on level i are created by 'claims' at level $i-1$ when propagated inside the obstacle



Complexity of $SPM_{=k}$: Boundary Sources

Boundary sources on level i are created by 'claims' at level $i-1$ when propagated inside the obstacle

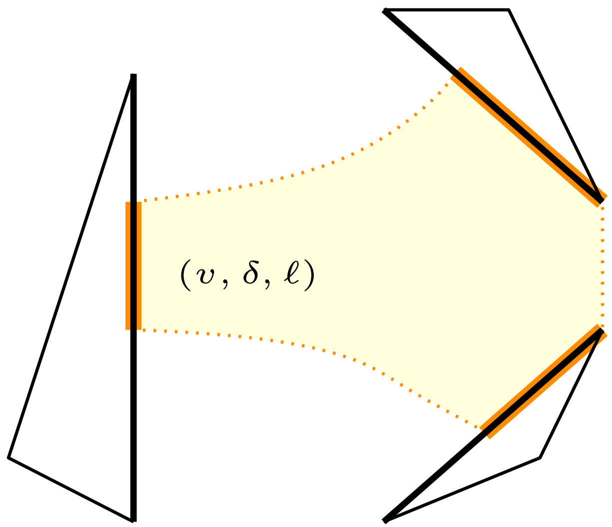


Two types of claims

– On the same edge : Entry Claim Cluster

Complexity of $SPM_{=k}$: Boundary Sources

Boundary sources on level i are created by 'claims' at level $i-1$ when propagated inside the obstacle

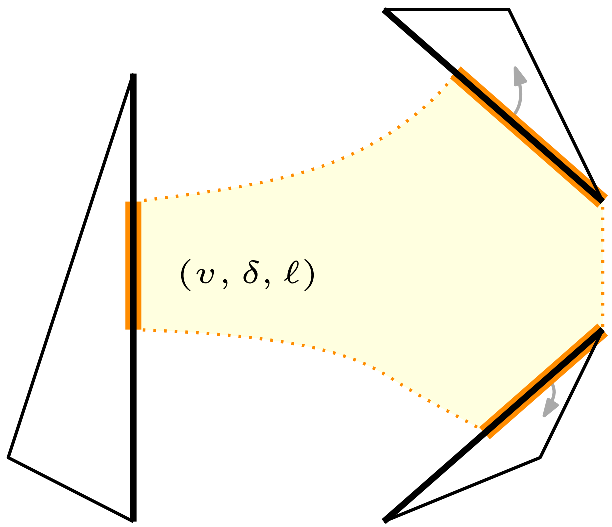


Two types of claims

- On the same edge : Entry Claim Cluster
- On another edge : Exit Claim Cluster

Complexity of $SPM_{=k}$: Boundary Sources

Boundary sources on level i are created by 'claims' at level $i-1$ when propagated inside the obstacle



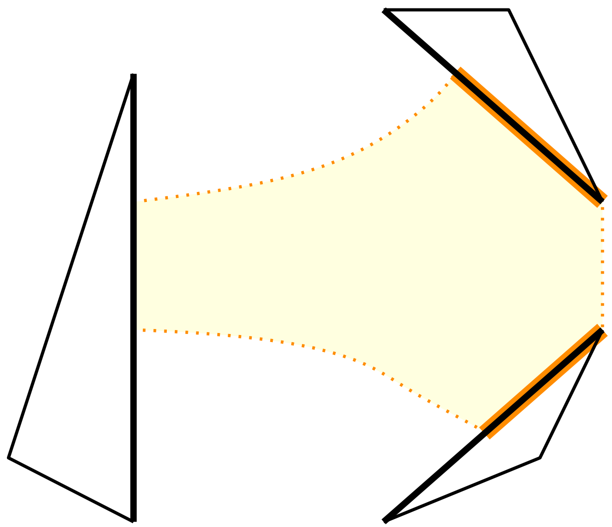
Two types of claims

- On the same edge : Entry Claim Cluster
- On another edge : Exit Claim Cluster

Only exit claims need to be propagated inside obstacles

Complexity of SPM_k

Lemma : The total number of exit claim clusters obtained by propagating m boundary sources in a domain is $m + O(n)$

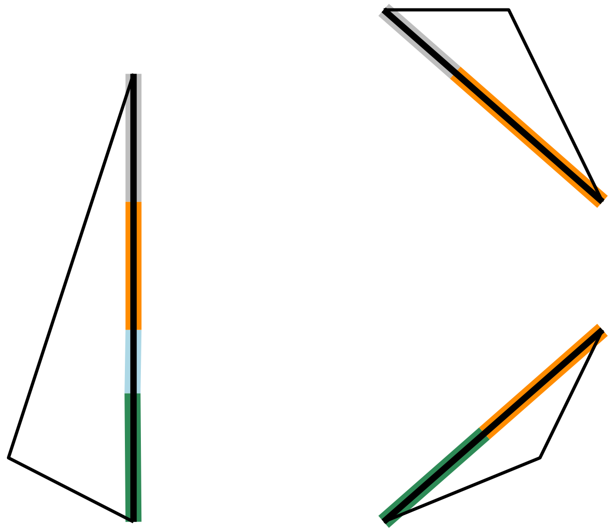


Complexity of $SPM_{\boldsymbol{k}}$

Lemma : The total number of exit claim clusters obtained by propagating m boundary sources in a domain is $m + O(n)$

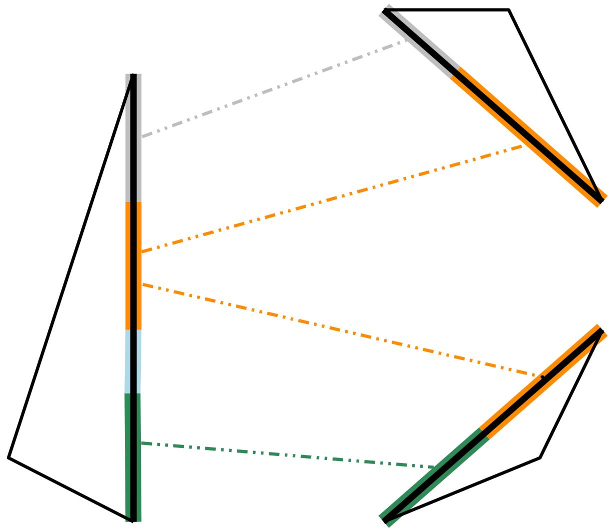
Proof sketch:

Connect exit claims of each of the m sources



Complexity of SPM_k

Lemma : The total number of exit claim clusters obtained by propagating m boundary sources in a domain is $m + O(n)$



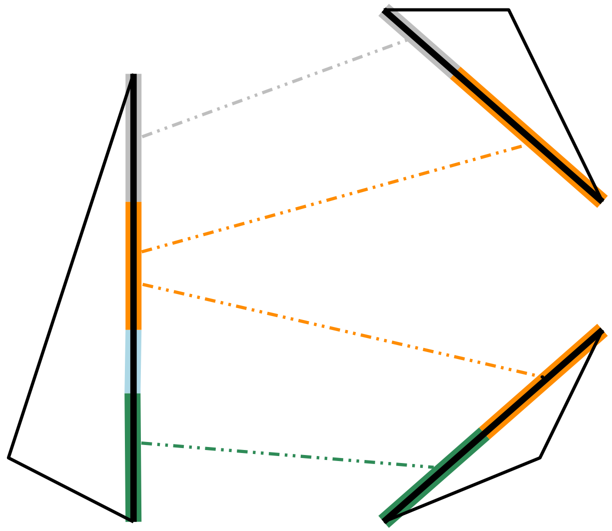
Proof sketch:

Connect exit claims of each of the m sources

Claims are non-crossing as k-paths at same level are non-crossing

Complexity of SPM_k

Lemma : The total number of exit claim clusters obtained by propagating m boundary sources in a domain is $m + O(n)$



Proof sketch:

Connect exit claims of each of the m sources

Claims are non-crossing as k-paths at same level are non-crossing

With a planarity based 'technical' argument, we get the said bound

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

If m is the number of sources at level $i - 1$

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

If m is the number of sources at level $i-1$

$m' = m + O(n)$ sources for propagation inside obstacles

By Previous Lemma

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

If m is the number of sources at level $i-1$

$m' = m + O(n)$ sources for propagation inside obstacles

$m'' = m' + O(n)$ sources for propagation at level i

By Previous Lemma

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

If m is the number of sources at level $i-1$

$m' = m + O(n)$ sources for propagation inside obstacles

$m'' = m' + O(n)$ sources for propagation at level i

By Previous Lemma

Number of sources at level k

Complexity of SPM_k

The complexity of SPM_k is $O(kn)$

We construct SPM_k inductively one level at a time

If m is the number of sources at level $i-1$

$m' = m + O(n)$ sources for propagation inside obstacles

$m'' = m' + O(n)$ sources for propagation at level i

By Previous Lemma

Number of sources at level k

$$n + \underbrace{Cn + Cn + \ldots + Cn}$$

k times, one per level; C is some constant

Computing SPM_k : Running Time

If S_i is the size of SPM_i

$$\text{Total Running time} \quad = \quad \sum_{i=1,2,..k} O(S_i \log S_i)$$

$$\leq \quad O(k \cdot S_k \log S_k)$$

Computing SPM_k : Running Time

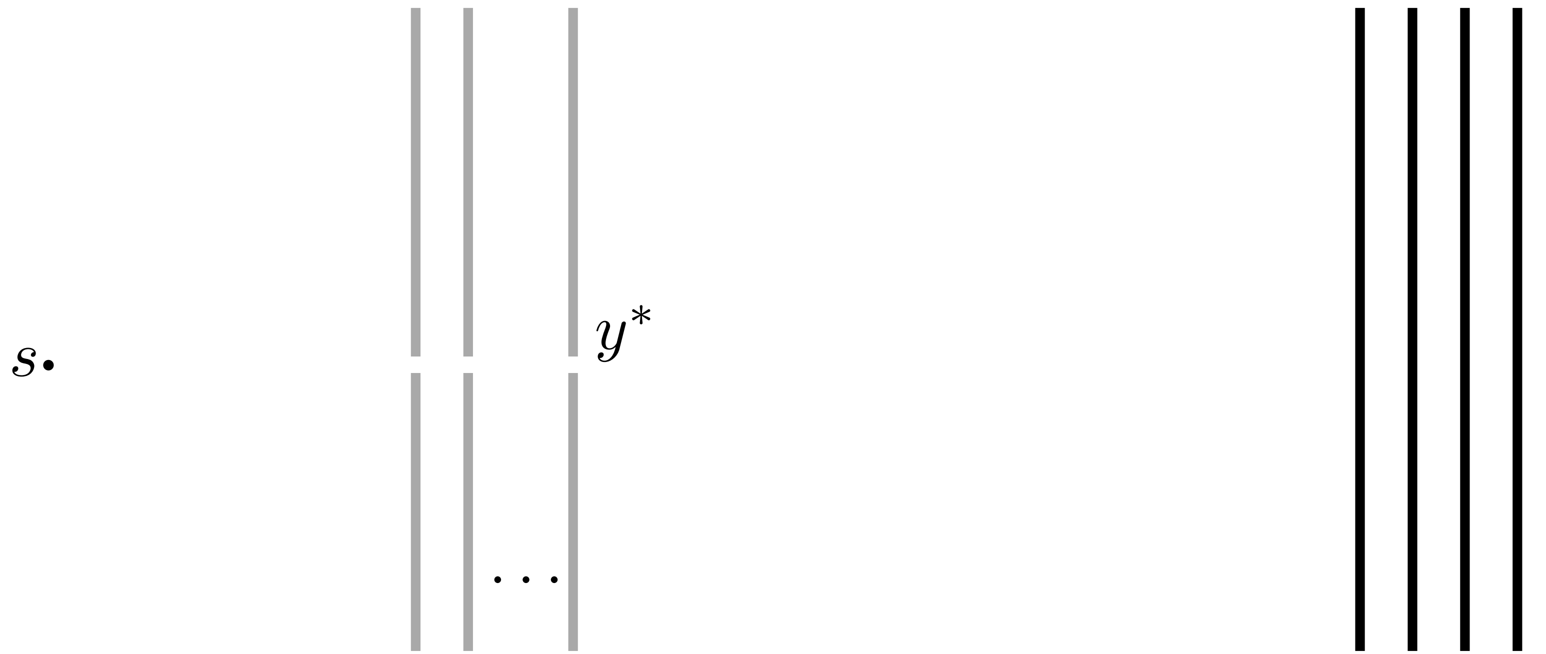
If S_i is the size of SPM_i

$$\text{Total Running time} \quad = \quad \sum_{i\,=\,1,\,2,\,..k} O(S_i \log S_i)$$

$$\leq \quad O(k \cdot S_k \log S_k)$$

Theorem: SPM_k can be computed in $O(k^2 n \log n)$ total time and $O(kn \log n)$ space

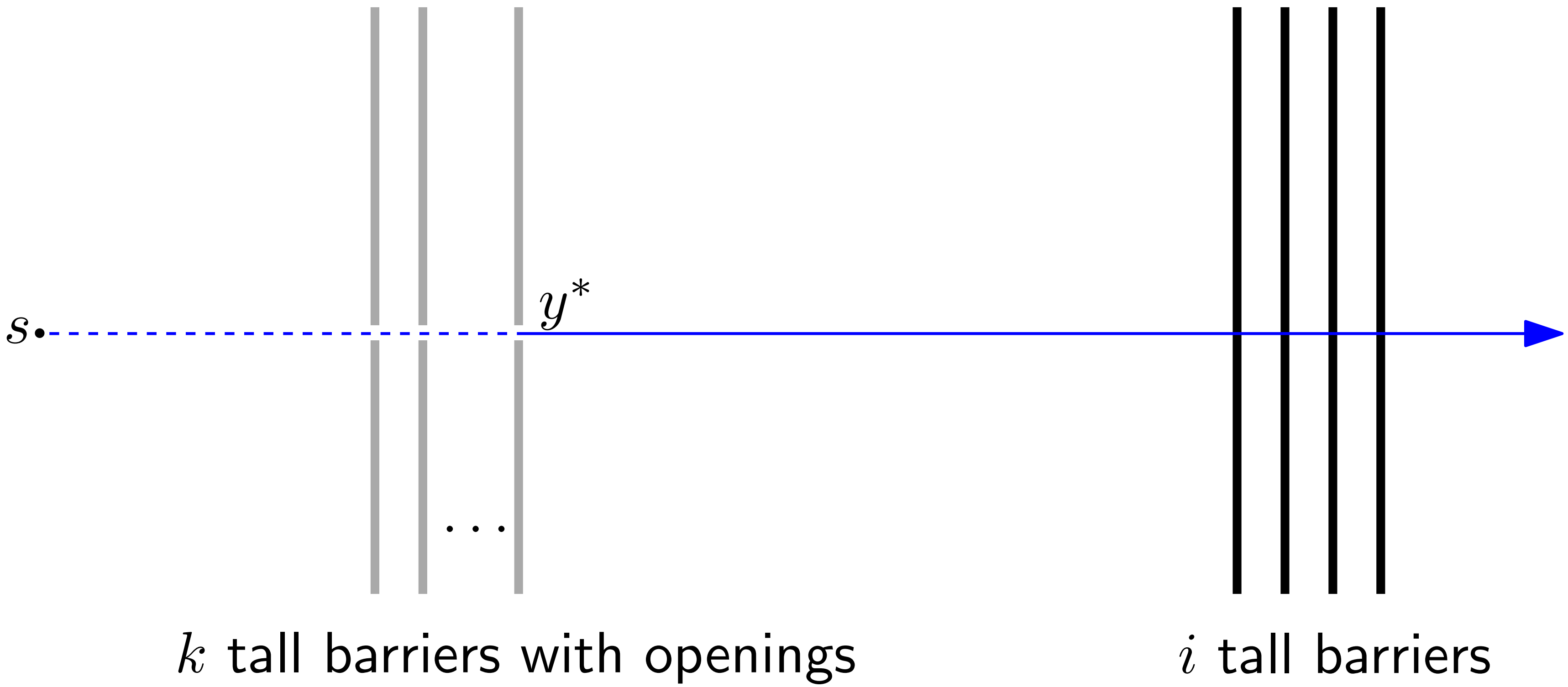
An $\Omega(kn)$ lower bound for SPM_k



k tall barriers with openings

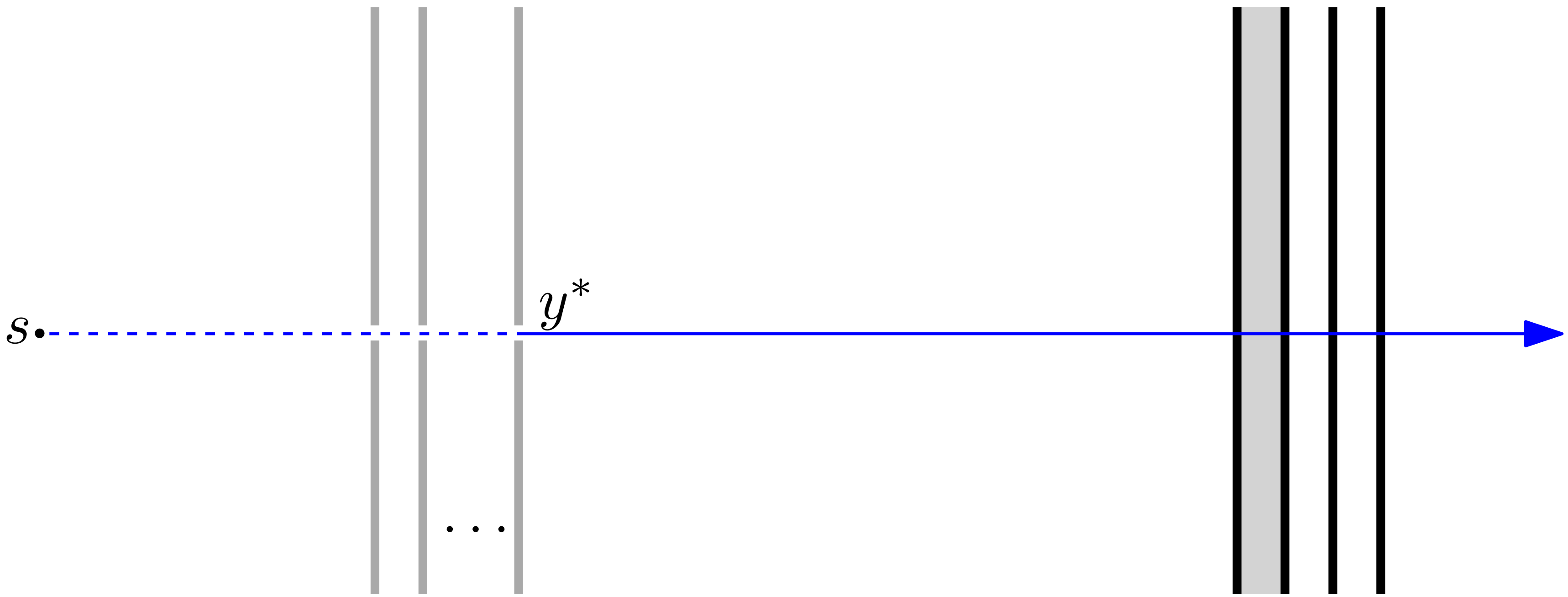
i tall barriers

An $\Omega(kn)$ lower bound for SPM_k



Can have at most $k - i$ crossings before y^*

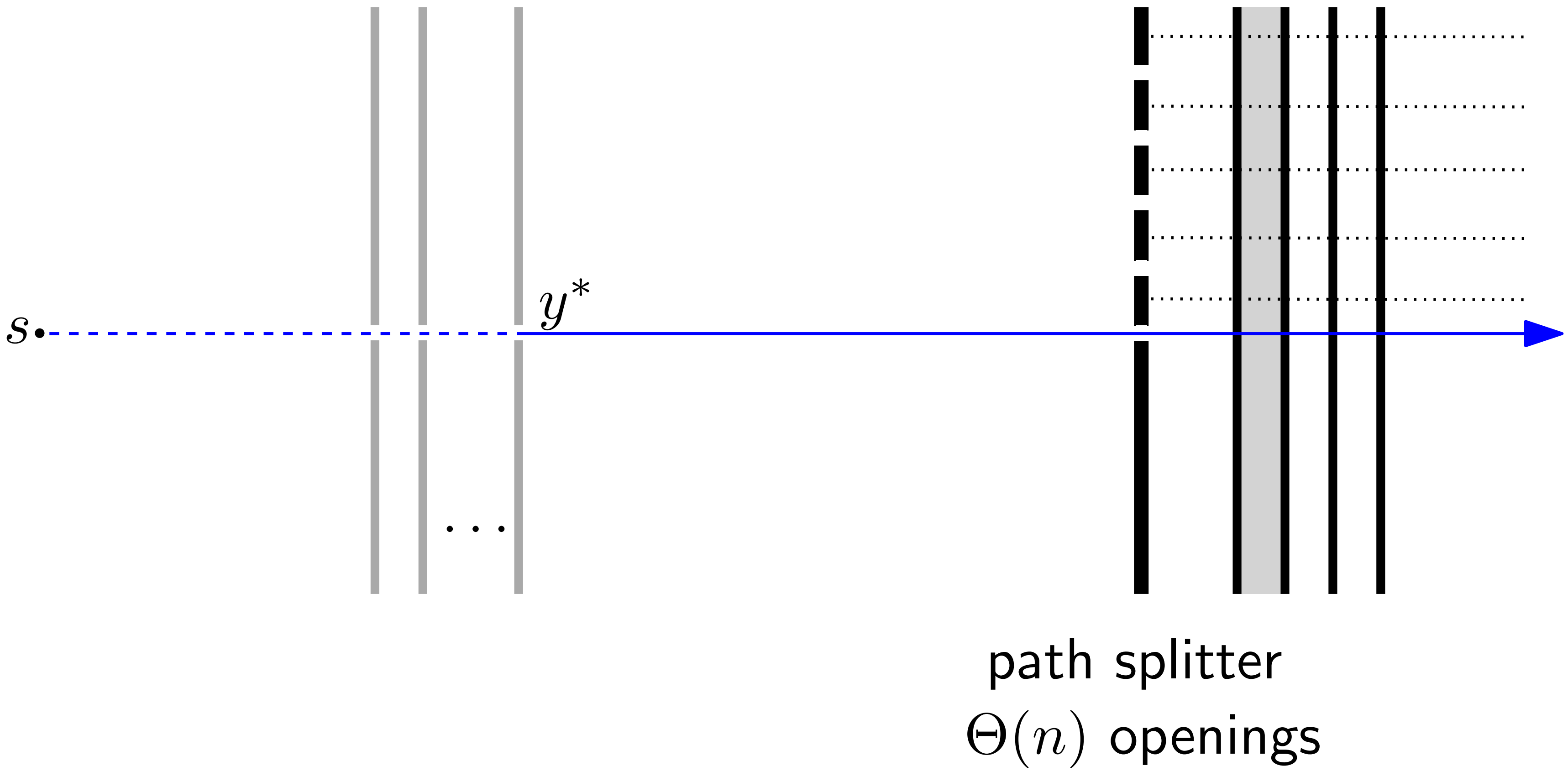
An $\Omega(kn)$ lower bound for SPM_k



Can have at most $k - i$ crossings before y^*

$\Rightarrow$ Each barrier creates one region to its right

An $\Omega(kn)$ lower bound for SPM_k

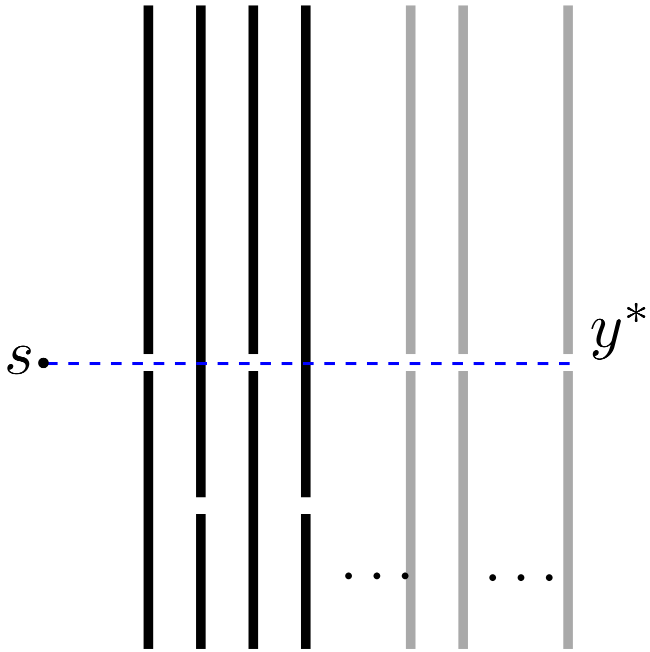


Can have at most $k - i$ crossings before y^*

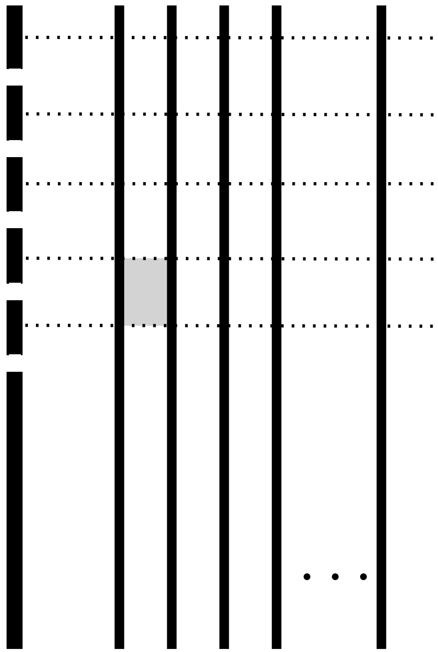
$\Rightarrow$ Each barrier creates one region to its right

Creates $\Theta(n)$ sub-regions for every such region

An $\Omega(kn)$ lower bound for SPM_k



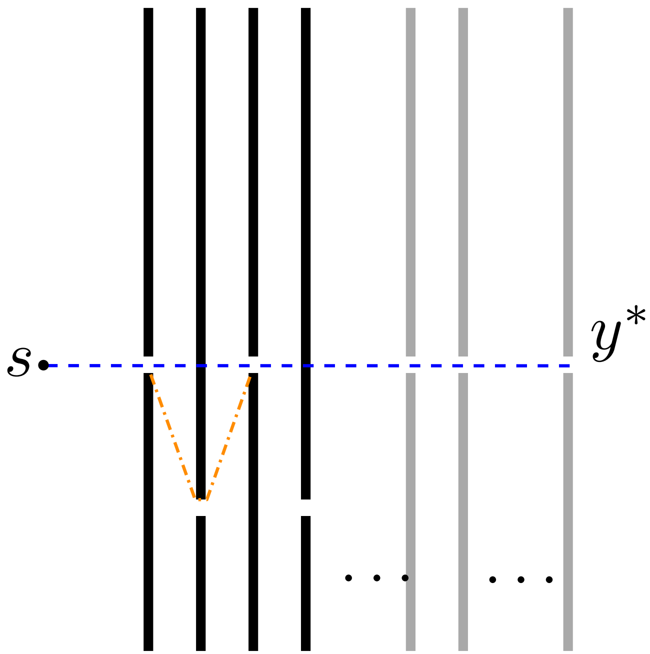
$2k$ strips



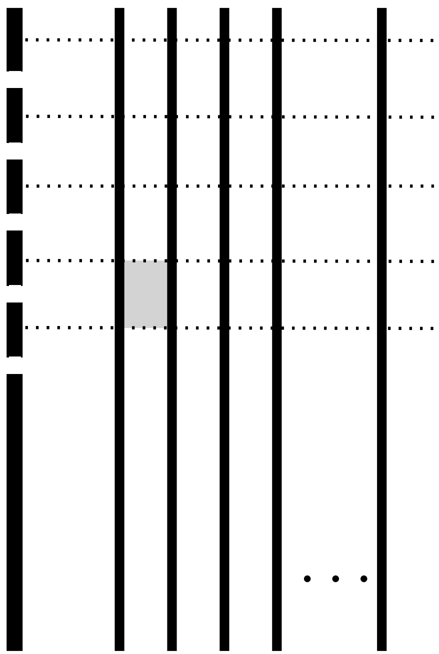
k strips

k-path from s to y^* can have 0 to k crossings

An $\Omega(kn)$ lower bound for SPM_k



$2k$ strips

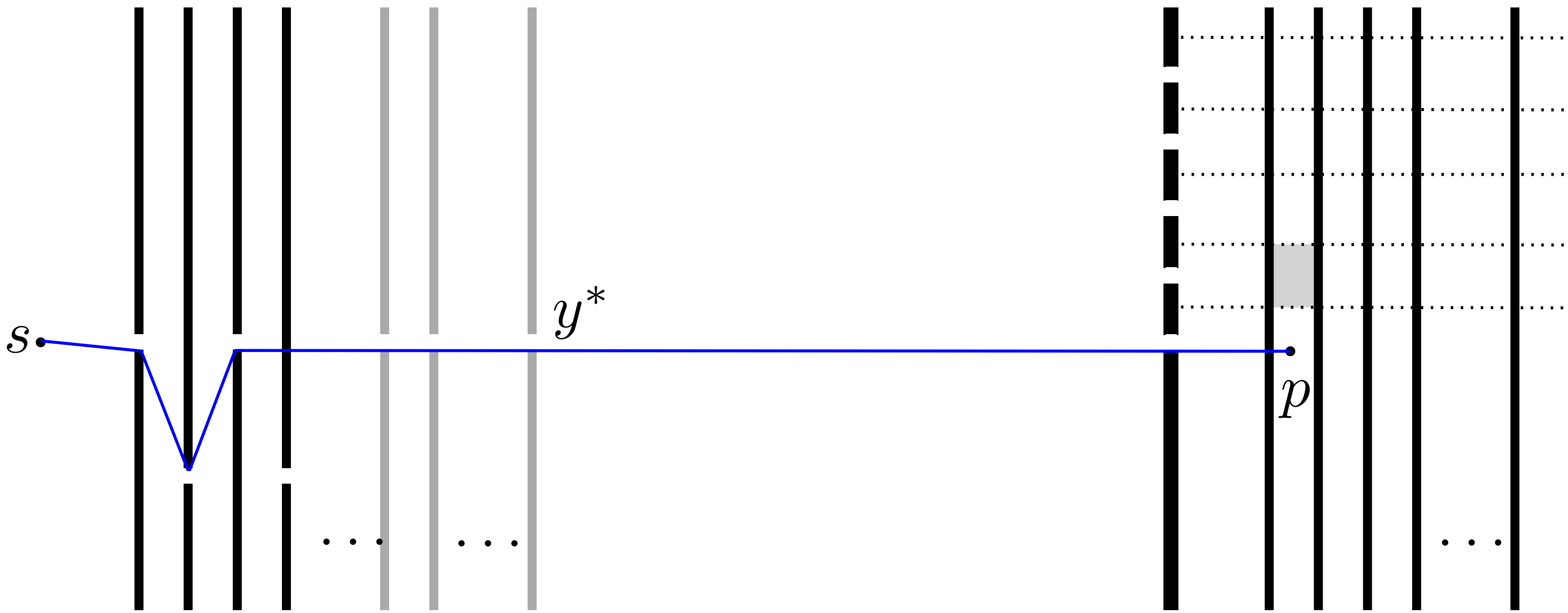


k strips

k-path from s to y^* can have 0 to k crossings

j crossings $\Rightarrow$ $(k-j)$ detours

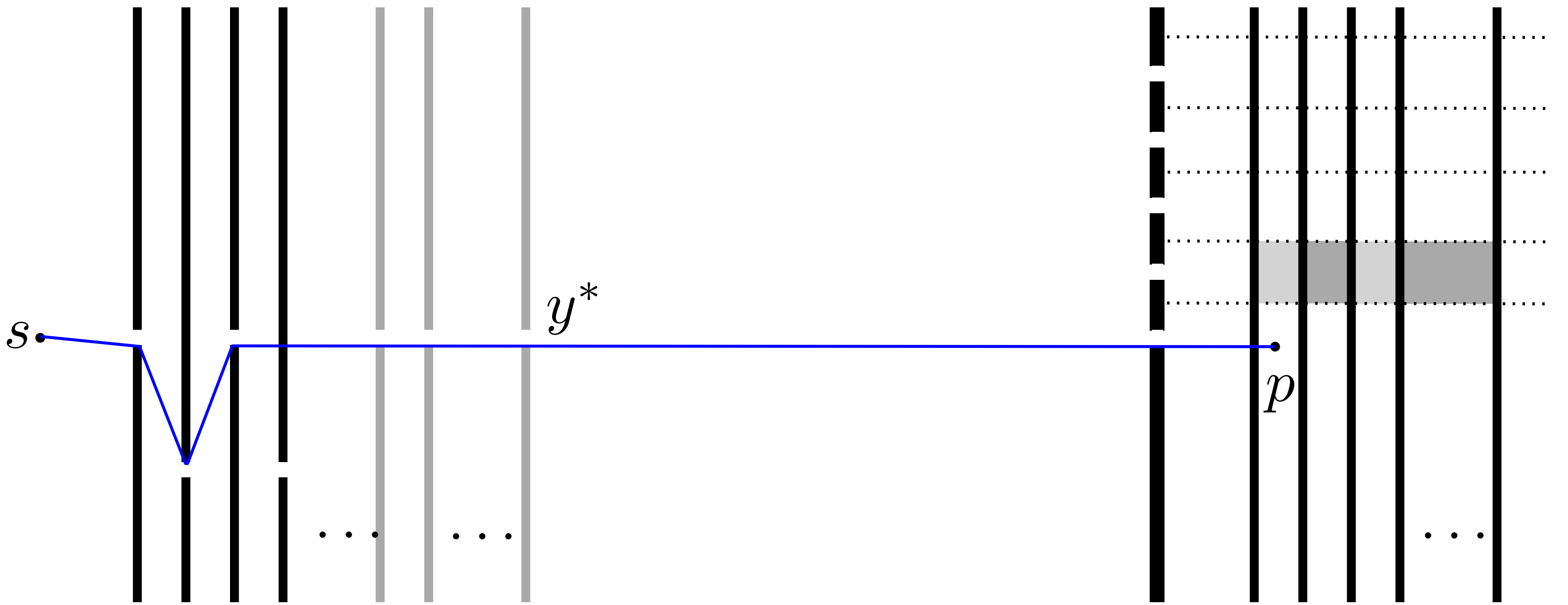
An $\Omega(kn)$ lower bound for SPM_k



k-path from s to y^* can have 0 to k crossings

j crossings $\Rightarrow (k - j)$ detours

An $\Omega(kn)$ lower bound for SPM_k



k-path from s to y^* can have 0 to k crossings

j crossings $\Rightarrow$ $(k-j)$ detours

Total $\Theta(k)$ regions per splitter opening, $\Theta(kn)$ regions in total.

In Summary

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

$\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

$\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

Use number of obstacles crossed by prefix path

In Summary

⇒ Study the problem of Shortest paths that violate $\leq k$ obstacles

⇒ Simple $O(kn^2)$ algorithm using visibility graphs

⇒ Decompose shortest k-paths into non-crossing subpaths

Use number of obstacles crossed by prefix path

Segregate relevant sub-paths on floors of k-garage

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

$\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

Use number of obstacles crossed by prefix path

Segregate relevant sub-paths on floors of k-garage

$\Rightarrow$ Apply Continuous Dijkstra at each floor with $O(kn)$ sources

In Summary

- $\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles
- $\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs
- $\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

  Use number of obstacles crossed by prefix path

  Segregate relevant sub-paths on floors of k-garage
- $\Rightarrow$ Apply Continuous Dijkstra at each floor with $O(kn)$ sources
- $\Rightarrow$ Establish a tight bound of $\Theta(kn)$ on size of map SPM_k.

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

$\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

Use number of obstacles crossed by prefix path

Segregate relevant sub-paths on floors of k-garage

$\Rightarrow$ Apply Continuous Dijkstra at each floor with $O(kn)$ sources

$\Rightarrow$ Establish a tight bound of $\Theta(kn)$ on size of map SPM_k.

$\Rightarrow$ Running time is $O(kn \log n)$ per floor, $O(k^2 n \log n)$ total.

In Summary

$\Rightarrow$ Study the problem of Shortest paths that violate $\leq k$ obstacles

$\Rightarrow$ Simple $O(kn^2)$ algorithm using visibility graphs

$\Rightarrow$ Decompose shortest k-paths into non-crossing subpaths

Use number of obstacles crossed by prefix path

Segregate relevant sub-paths on floors of k-garage

$\Rightarrow$ Apply Continuous Dijkstra at each floor with $O(kn)$ sources

$\Rightarrow$ Establish a tight bound of $\Theta(kn)$ on size of map SPM_k.

$\Rightarrow$ Running time is $O(kn \log n)$ per floor, $O(k^2 n \log n)$ total.

Thanks!