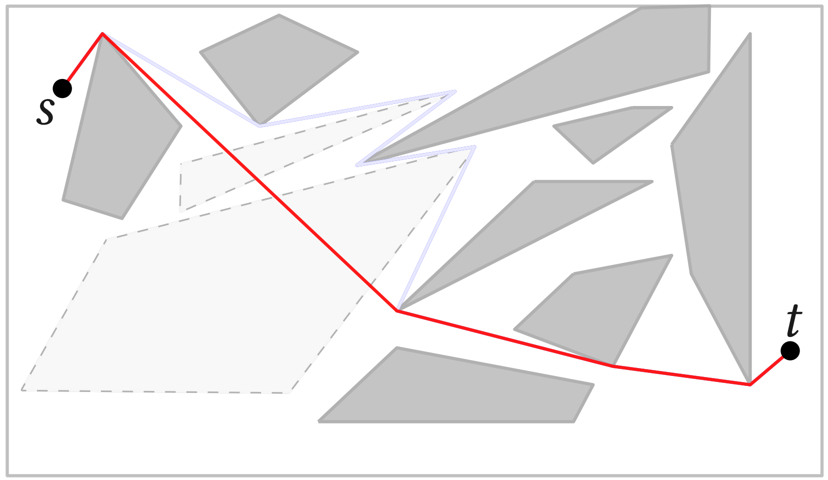
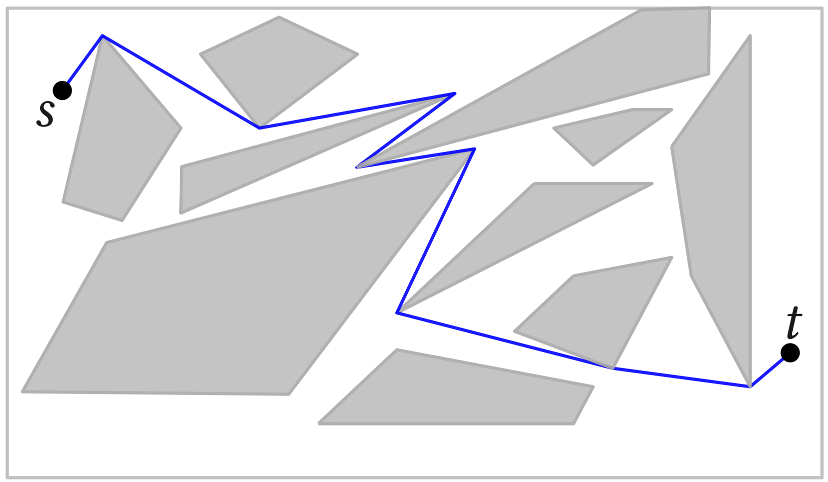


Computing Shortest Paths in the Plane with Removable Obstacles

Pankaj K. Agarwal, Neeraj Kumar, Stavros Sintos and Subhash Suri

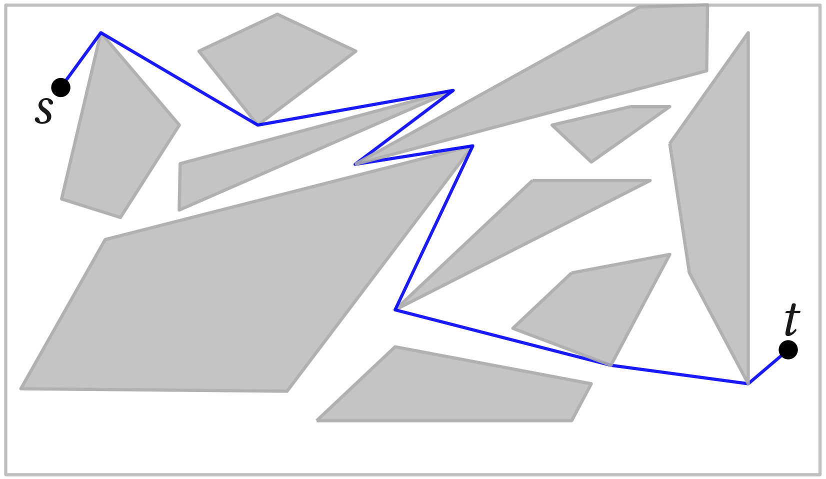
Duke University and UC Santa Barbara

Problem Description



Input: h polygonal obstacles with n vertices, source s and target t

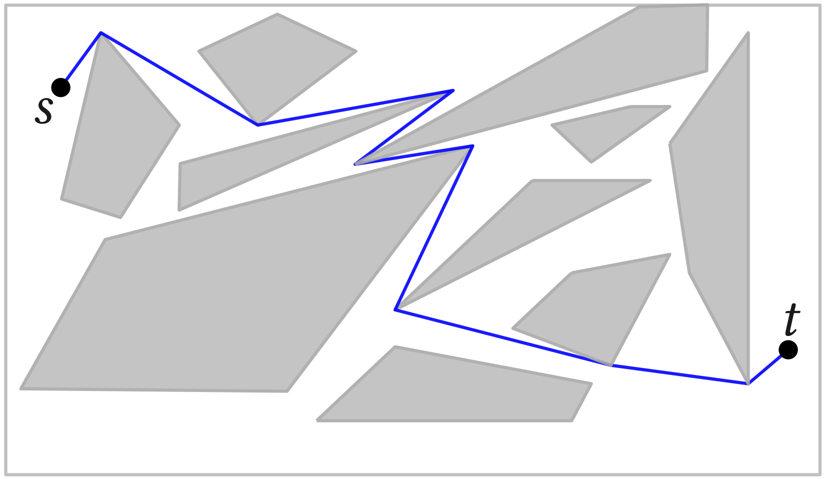
Problem Description



Input: h polygonal obstacles with n vertices, source s and target t

An obstacle P_i can be removed by paying cost c_i

Problem Description

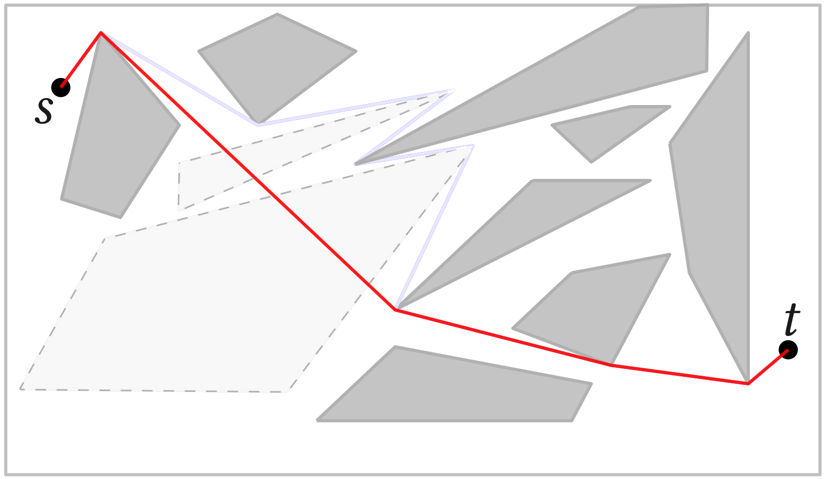


Input: h polygonal obstacles with n vertices, source s and target t

An obstacle P_i can be removed by paying cost c_i

Assumption: Obstacles are mutually disjoint and convex

Problem Description



Input: h polygonal obstacles with n vertices, source s and target t

An obstacle P_i can be removed by paying cost c_i

Given a cost budget C, which obstacles should we remove

so that length of path from s to t is minimized?

Assumption: Obstacles are mutually disjoint and convex

Motivation

Classically, we deal with finding shortest path in a fixed network

Motivation

Classically, we deal with finding shortest path in a fixed network

But what if shortest paths are not good enough!

Motivation

Classically, we deal with finding shortest path in a fixed network

But what if shortest paths are not good enough!

Desire paths

What if we are allowed to modify the environment

What changes should we make?

Motivation

Classically, we deal with finding shortest path in a fixed network

But what if shortest paths are not good enough!

What if we are allowed to modify the environment

What changes should we make?

Applications

Urban Planning

Adding ‘shortcuts” to ease congestion

Motivation

Classically, we deal with finding shortest path in a fixed network

But what if shortest paths are not good enough!

What if we are allowed to modify the environment

What changes should we make?

Applications

Reconfiguring road networks

– such as by adding flyovers

Motivation

Classically, we deal with finding shortest path in a fixed network

But what if shortest paths are not good enough!

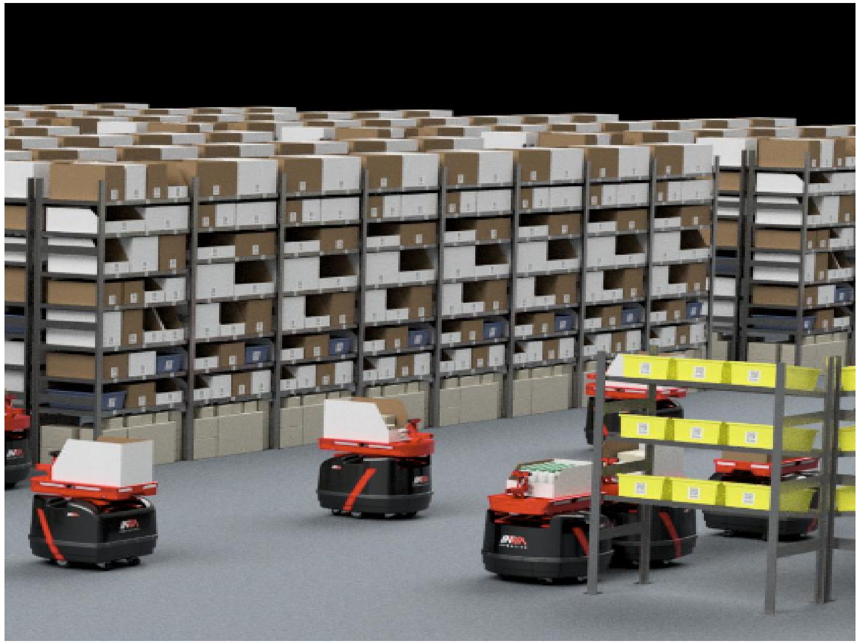
What if we are allowed to modify the environment

What changes should we make?

Applications

Re-organize Warehouse Layout

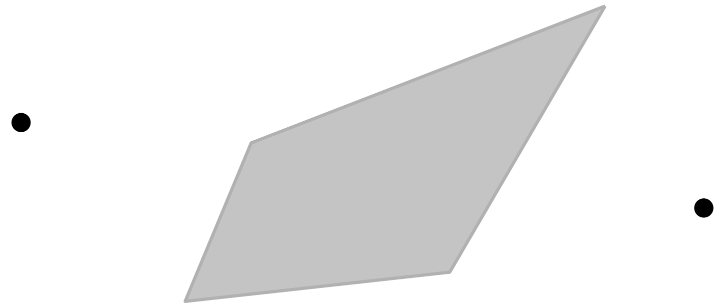
– Shorten frequent paths for robot



Path Planning Under Uncertainity

Obstacle P_i is present in input with independent probability β_i,

not present with probability $(1 - \beta_i)$



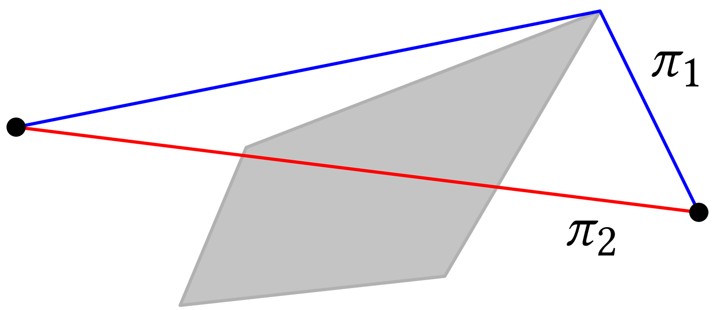
Probability of existence = β_i

Path Planning Under Uncertainity

Obstacle P_i is present in input with independent probability β_i,
not present with probability $(1 - \beta_i)$

Path π_1 has probability 1

Path π_2 has probability $(1 - \beta_i)$

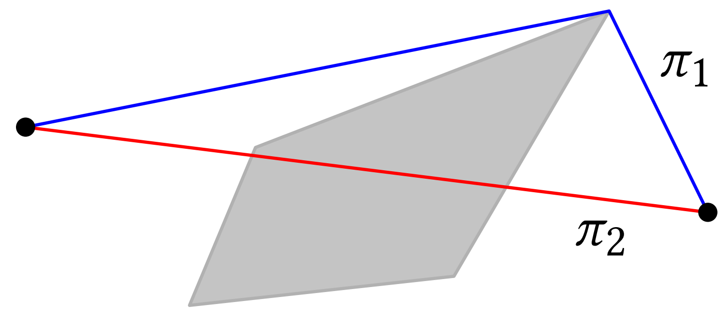


Probability of existence = β_i

Path Planning Under Uncertaininity

Obstacle P_i is present in input with independent probability β_i,
not present with probability $(1 - \beta_i)$

Path π_1 has probability 1
Path π_2 has probability $(1 - \beta_i)$



Probability of existence = β_i

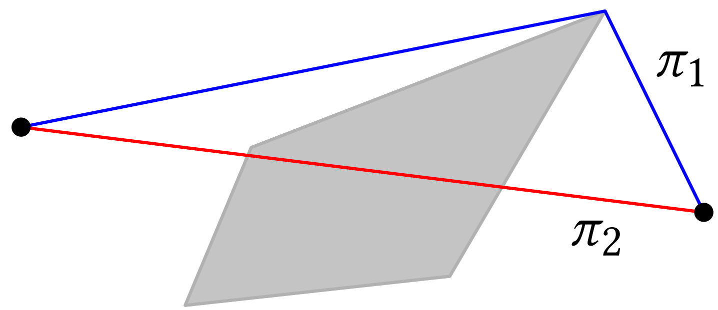
What is the shortest s–t path that has probability at least β?

Path Planning Under Uncertainity

Obstacle P_i is present in input with independent probability β_i,
not present with probability $(1 - \beta_i)$

Path π_1 has probability 1

Path π_2 has probability $(1 - \beta_i)$



Probability of existence = β_i

What is the shortest s–t path that has probability at least β?

What is the most likely s–t path with length at most L?

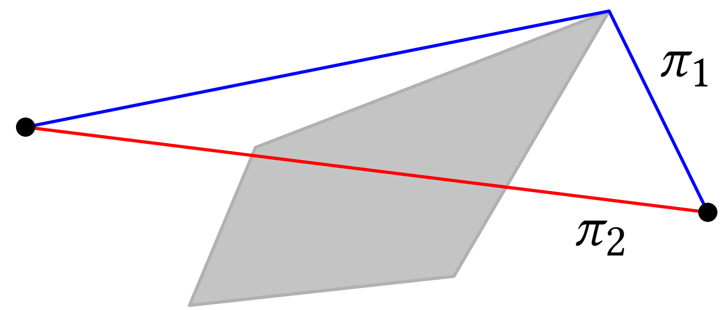
(one that has highest probability)

Path Planning Under Uncertainity

Obstacle P_i is present in input with independent probability β_i,
not present with probability $(1 - \beta_i)$

Path π_1 has probability 1

Path π_2 has probability $(1 - \beta_i)$



Probability of existence = β_i

What is the shortest s–t path that has probability at least β?

What is the most likely s–t path with length at most L?

(one that has highest probability)

Can reduce to the earlier cost-based model by taking negative logarithms

Related Work

[HKS'17] Shortest Paths in the Plane with Obstacle Violations

Computes shortest s–t path in the plane that removes at most k obstacles
$O(k^2 n \log n)$ algorithm using Continuous Dijkstra

Related Work

[HKS'17] Shortest Paths in the Plane with Obstacle Violations

Computes shortest s–t path in the plane that removes at most k obstacles

$O(k^2 n \log n)$ algorithm using Continuous Dijkstra

Cardinality Model

Obstacles have unit cost of removal, budget = k

Related Work

[HKS'17] Shortest Paths in the Plane with Obstacle Violations

Computes shortest s–t path in the plane that removes at most k obstacles
$O(k^2 n \log n)$ algorithm using Continuous Dijkstra

Cardinality Model
Obstacles have unit cost of removal, budget = k

We study the more general **cost-based model** of obstacle removal

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

- An FPTAS inspired from [HKS'17] running in worst case $\tilde{O}(n^2 h/\epsilon)$ time

  Minimum Path length with cost at most $(1+\epsilon)C$

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

- An FPTAS inspired from [HKS'17] running in worst case $\tilde{O}(n^2 h/\epsilon)$ time

  Minimum Path length with cost at most $(1+\epsilon)C$

- A faster FPTAS running in worst case $\tilde{O}(nh/\epsilon^2)$ time

  Length $(1+\epsilon)$ times optimal, cost at most $(1+\epsilon)C$

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

- An FPTAS inspired from [HKS'17] running in worst case $\tilde{O}(n^2 h/\epsilon)$ time

  Minimum Path length with cost at most $(1+\epsilon)C$

- A faster FPTAS running in worst case $\tilde{O}(nh/\epsilon^2)$ time

  Length $(1+\epsilon)$ times optimal, cost at most $(1+\epsilon)C$

- Data structures for **fixed source** approximate shortest path queries in $O(\log^2 n/\epsilon)$ time, **two-point queries** in $O(\log^2 n/\epsilon^2)$ time

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

  Simple reduction from PARTITION

- An FPTAS inspired from [HKS'17] running in worst case $\tilde{O}(n^2 h/\epsilon)$ time

  Minimum Path length with cost at most $(1+\epsilon)C$

- A faster FPTAS running in worst case $\tilde{O}(nh/\epsilon^2)$ time

  Length $(1+\epsilon)$ times optimal, cost at most $(1+\epsilon)C$

- Data structures for **fixed source** approximate shortest path queries in $O(\log^2 n/\epsilon)$ time, **two-point queries** in $O(\log^2 n/\epsilon^2)$ time

Our Results

- Problem of computing a path of length at most L and cost at most C is NP-HARD even for vertical segment obstacles.

Simple reduction from PARTITION

- An FPTAS inspired from [HKS'17] running in worst case $\tilde{O}(n^2 h/\epsilon)$ time

  Minimum Path length with cost at most $(1+\epsilon)C$

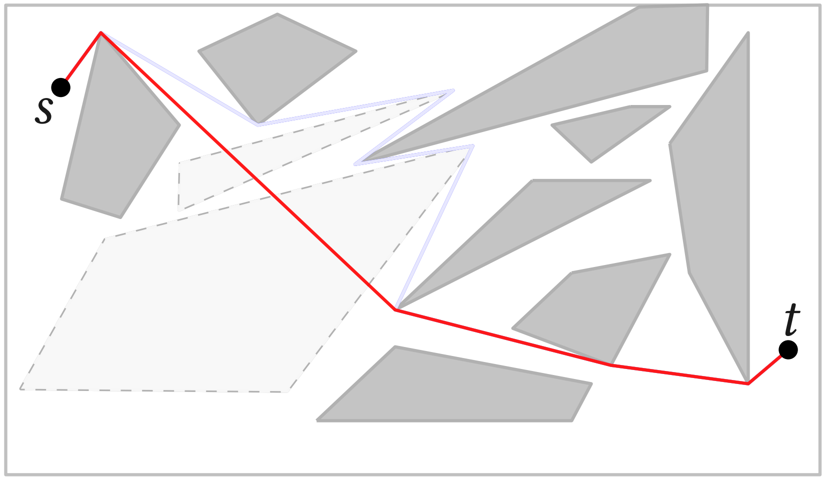
- A faster FPTAS running in worst case $\tilde{O}(nh/\epsilon^2)$ time

  Length $(1+\epsilon)$ times optimal, cost at most $(1+\epsilon)C$

- Data structures for **fixed source** approximate shortest path queries in $O(\log^2 n/\epsilon)$ time, **two-point queries** in $O(\log^2 n/\epsilon^2)$ time

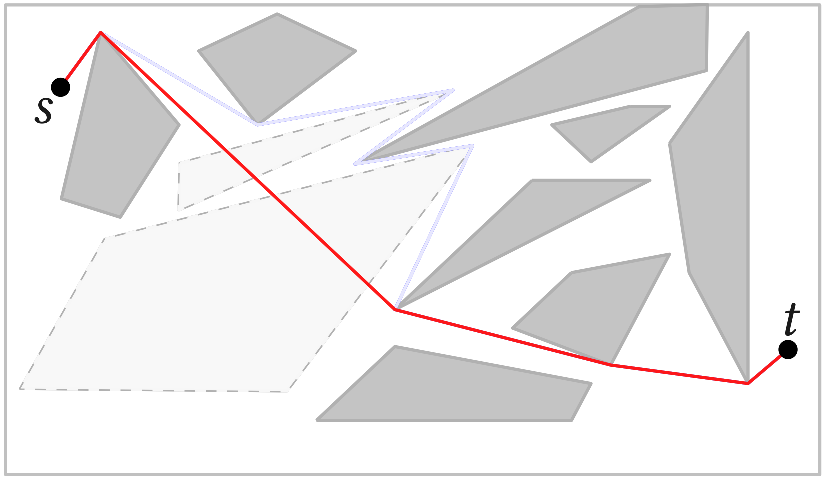
Rest of the talk

A Simple FPTAS



Observations

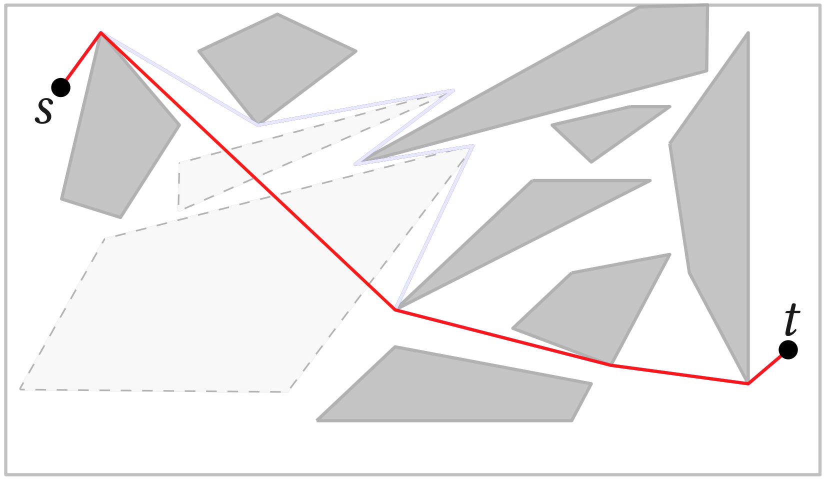
A Simple FPTAS



Observations

- The desired path π must ‘cross’ all the obstacles that were removed
- Since π has minimum length, it must only turn at obstacle vertices

A Simple FPTAS

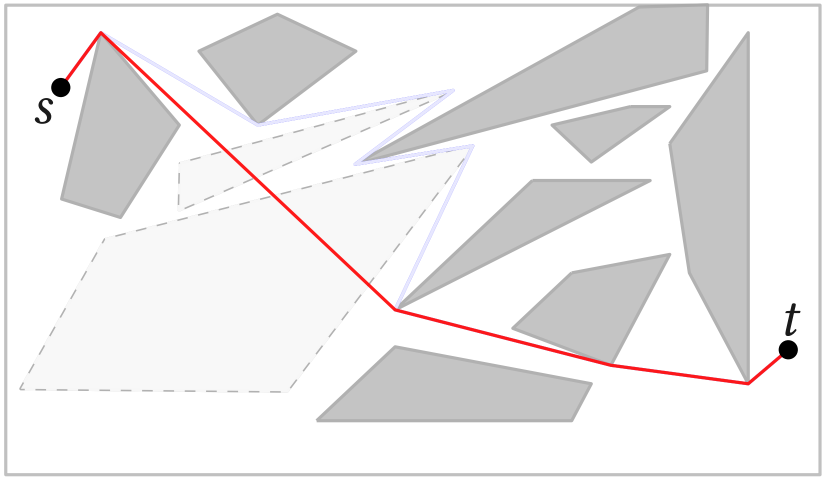


Observations

- The desired path π must 'cross' all the obstacles that were removed
- Since π has minimum length, it must only turn at obstacle vertices

Define $cost(\pi)$ = sum of costs of all obstacles crossed

A Simple FPTAS



Observations

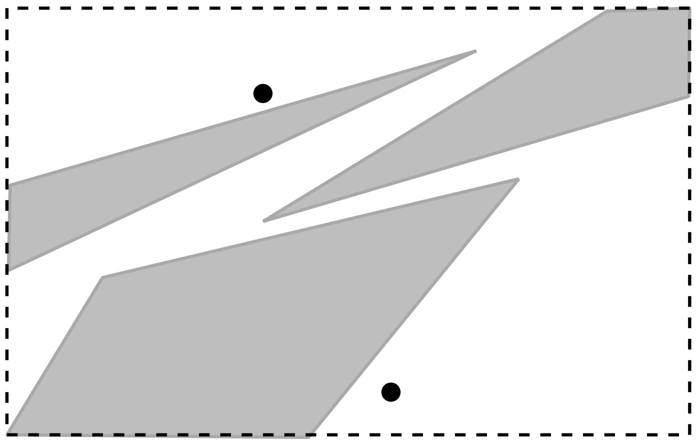
- The desired path π must 'cross' all the obstacles that were removed
- Since π has minimum length, it must only turn at obstacle vertices

Define $cost(\pi)$ = sum of costs of all obstacles crossed

What is the shortest s–t path that has cost at most C?

A Simple FPTAS

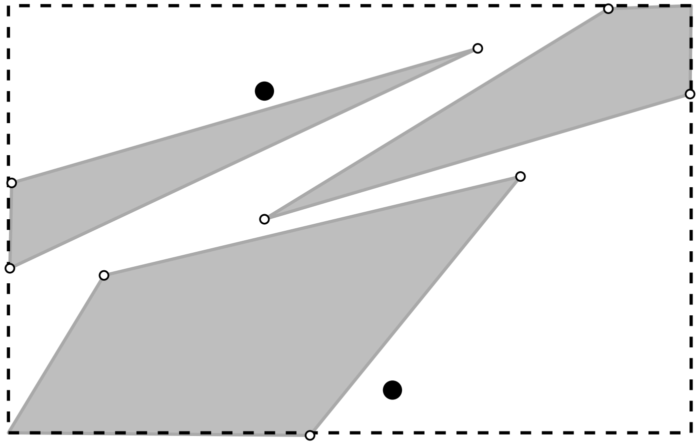
Scale all costs such that budget $C = h$, the number of obstacles



A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles

Model as a graph problem:



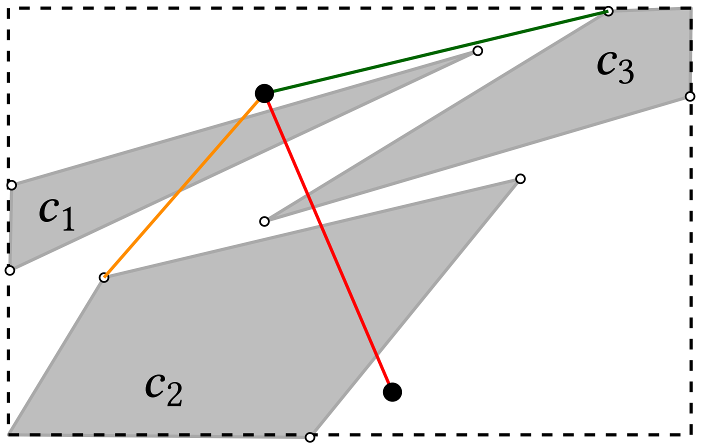
– **Nodes** : s, t, obstacle vertices

A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles

Model as a graph problem:

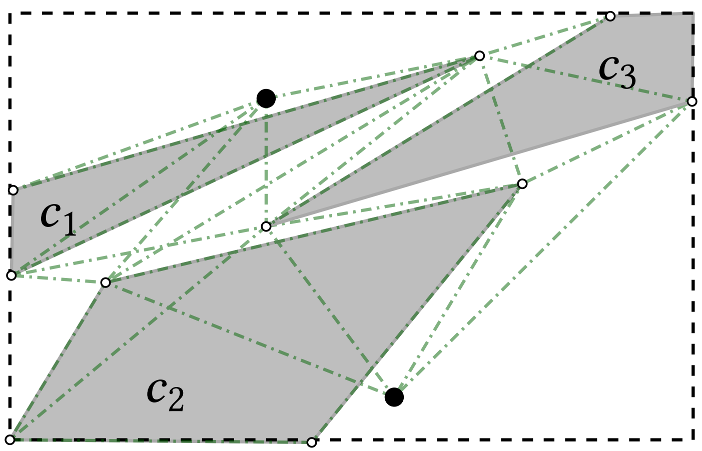
- **Nodes** : s, t, obstacle vertices
- **Edges** : between any pair of vertices (u, v) such that $cost(\overline{uv})$ is at most C



Cost of red edge is $c_1 + c_2 + c_3$

A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles



Model as a graph problem:

- **Nodes** : s, t, obstacle vertices
- **Edges** : between any pair of vertices (u, v)
  such that $cost(\overline{uv})$ is at most C

An edge e has two parameters:

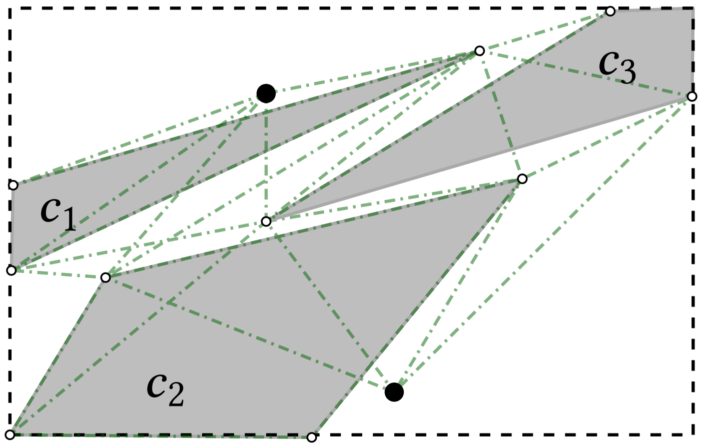
Euclidean length ℓ_e and cost of the edge c_e

A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles

Model as a graph problem:

- **Nodes** : s, t, obstacle vertices
- **Edges** : between any pair of vertices (u, v)
  such that $cost(\overline{uv})$ is at most C

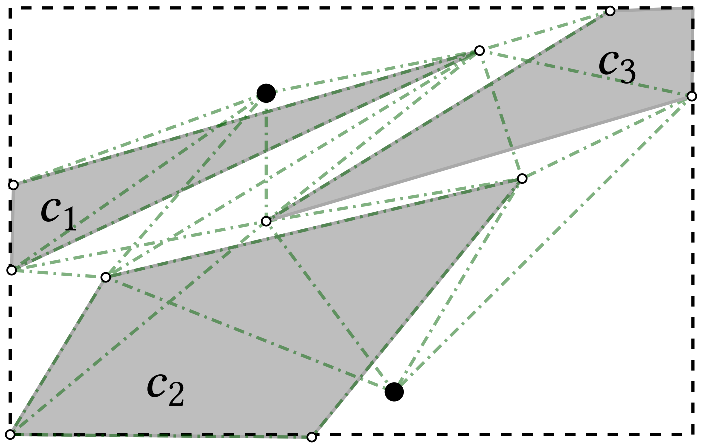


An edge e has two parameters:
Euclidean length ℓ_e and cost of the edge c_e

Mimics the notion of visibility graphs, we call it a **viability graph**

A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles



Model as a graph problem:

- **Nodes** : s, t, obstacle vertices
- **Edges** : between any pair of vertices (u, v) such that $cost(\overline{uv})$ is at most C

An edge e has two parameters:

Euclidean length ℓ_e and cost of the edge c_e

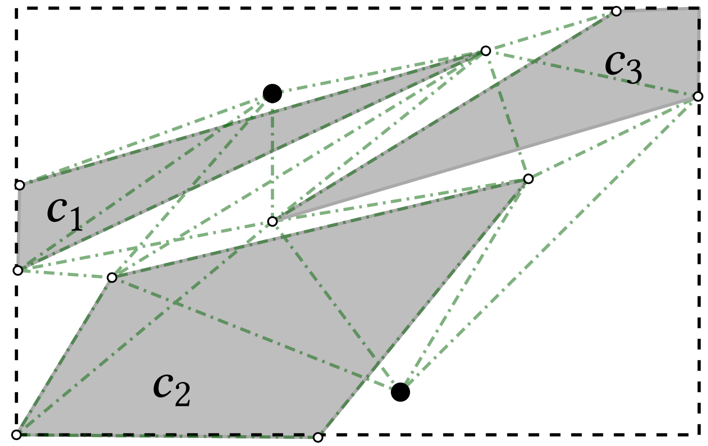
Mimics the notion of visibility graphs, we call it a **viability graph**

Find shortest s–t path in this graph such that the cost of the path is at most C

A Simple FPTAS

Scale all costs such that budget $C = h$, the number of obstacles

Model as a graph problem:



- **Nodes** : s, t, obstacle vertices
- **Edges** : between any pair of vertices (u, v) such that $cost(\overline{uv})$ is at most C

An edge e has two parameters:

Euclidean length ℓ_e and cost of the edge c_e

Mimics the notion of visibility graphs, we call it a **viability graph**

Find shortest s–t path in this graph such that the cost of the path is at most C

$(1+\epsilon)$-approximation using Dijkstra's algorithm in $\tilde{O}(\frac{n^2 h}{\epsilon})$ time by creating $O(\frac{h}{\epsilon})$ copies of vertices and edges

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1 + \epsilon)$ in path length for efficiency

Roadmap

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1 + \epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1+\epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

– Create $O(\frac{1}{\epsilon})$ copies of VG_1, one per direction and combine them to obtain VG_ϵ

Preserves pairwise distances within a factor of $(1+\epsilon)$

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1+\epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

– Create $O(\frac{1}{\epsilon})$ copies of VG_1, one per direction and combine them to obtain VG_ϵ

Preserves pairwise distances within a factor of $(1+\epsilon)$

– Run the modified Dijkstra on $O(\frac{n}{\epsilon} \log n)$ size viability graph VG_ϵ

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1+\epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

– Create $O(\frac{1}{\epsilon})$ copies of VG_1, one per direction and combine them to obtain VG_ϵ

Preserves pairwise distances within a factor of $(1+\epsilon)$

– Run the modified Dijkstra on $O(\frac{n}{\epsilon} \log n)$ size viability graph VG_ϵ

Primary challenge is to construct the graph VG_1

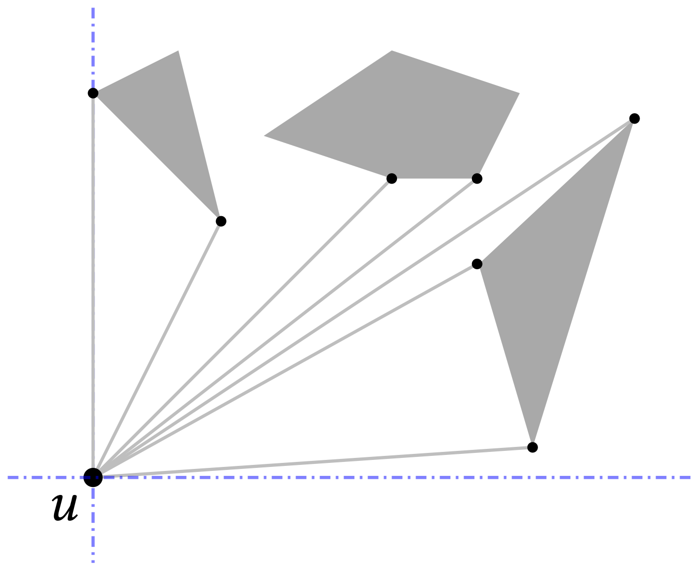
A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

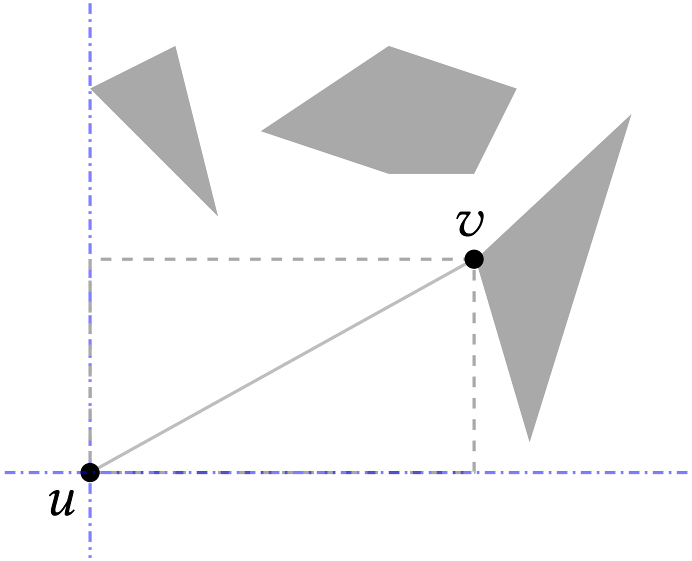


- Do not need all $O(n)$ edges adjacent to u

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

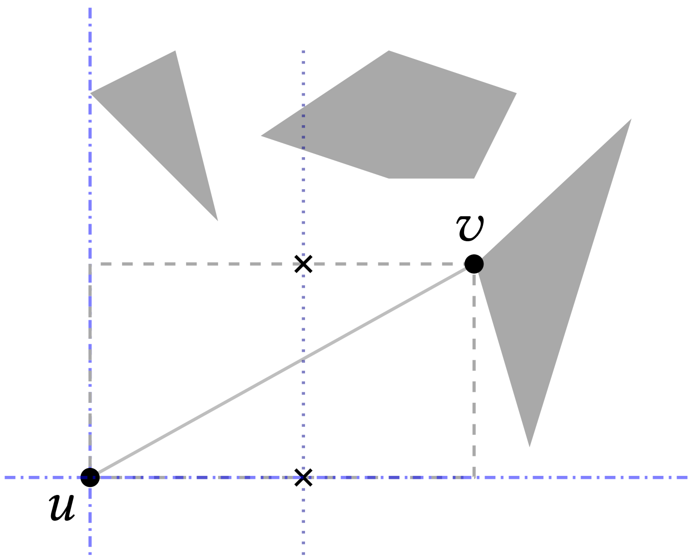


- Do not need all $O(n)$ edges adjacent to u

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

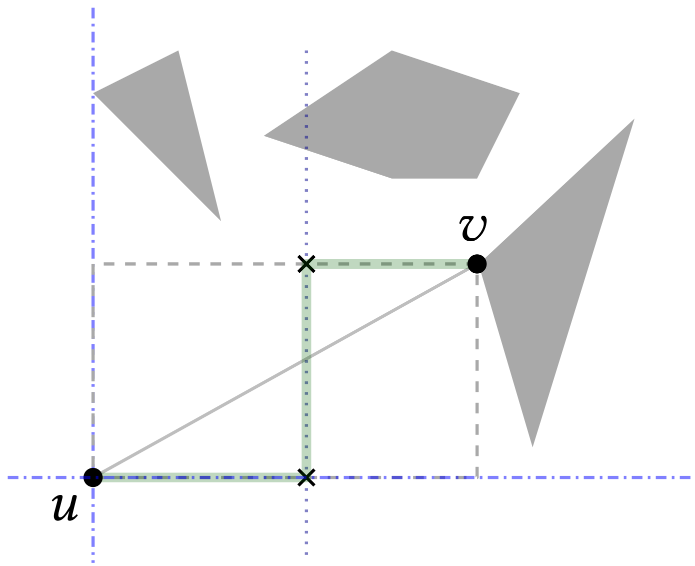


- Do not need all $O(n)$ edges adjacent to u
- Take projections on a 'split line' ℓ

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

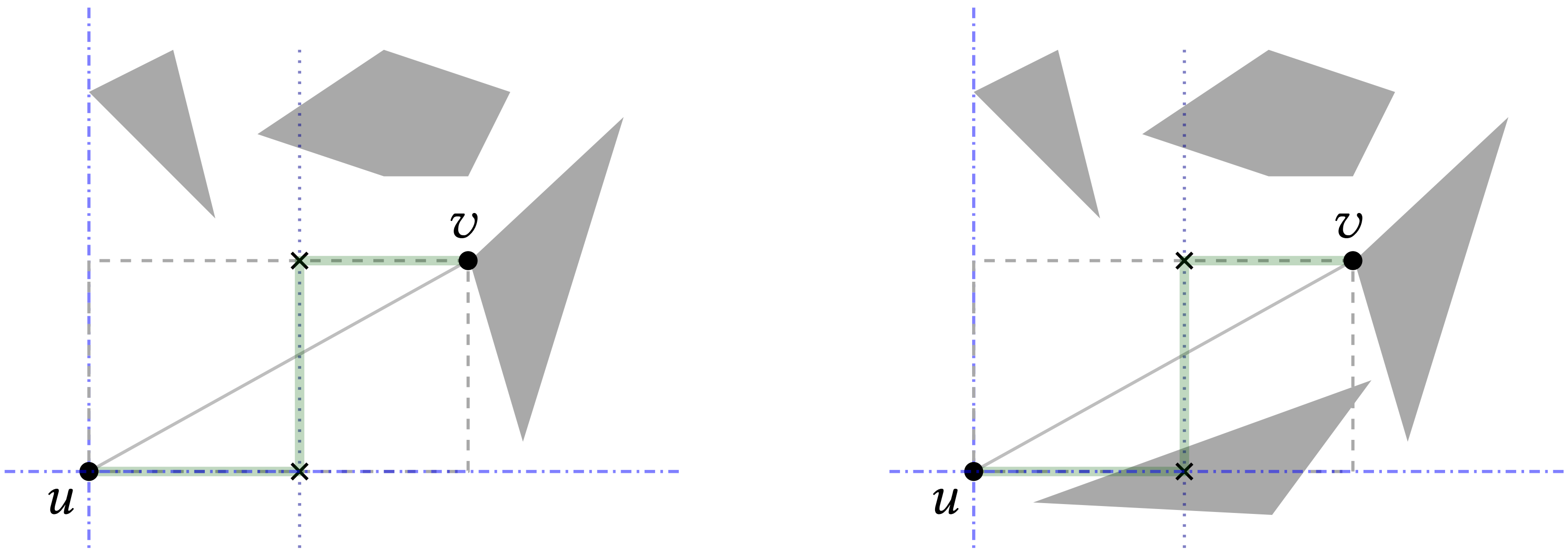


- Do not need all $O(n)$ edges adjacent to u
- Take projections on a 'split line' ℓ

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:



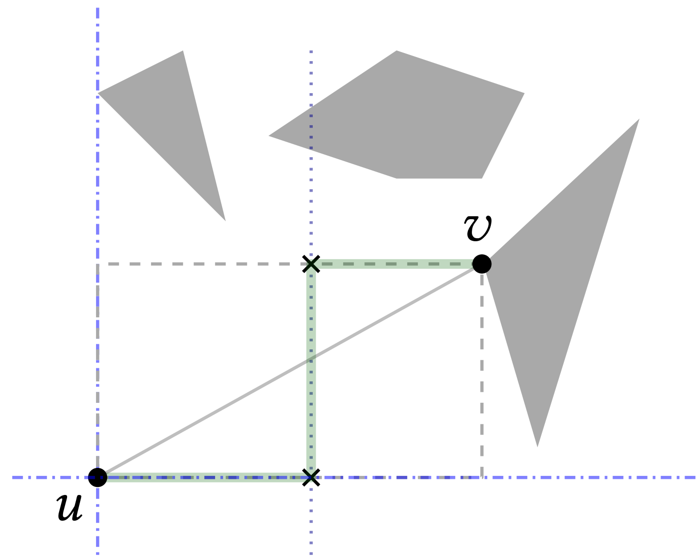
- Do not need all $O(n)$ edges adjacent to u
- Take projections on a 'split line' ℓ

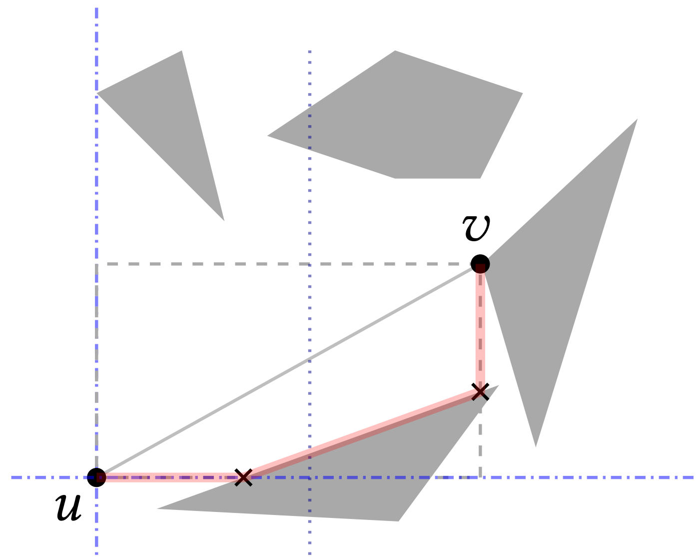
A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

Total size: $O(n \log n)$





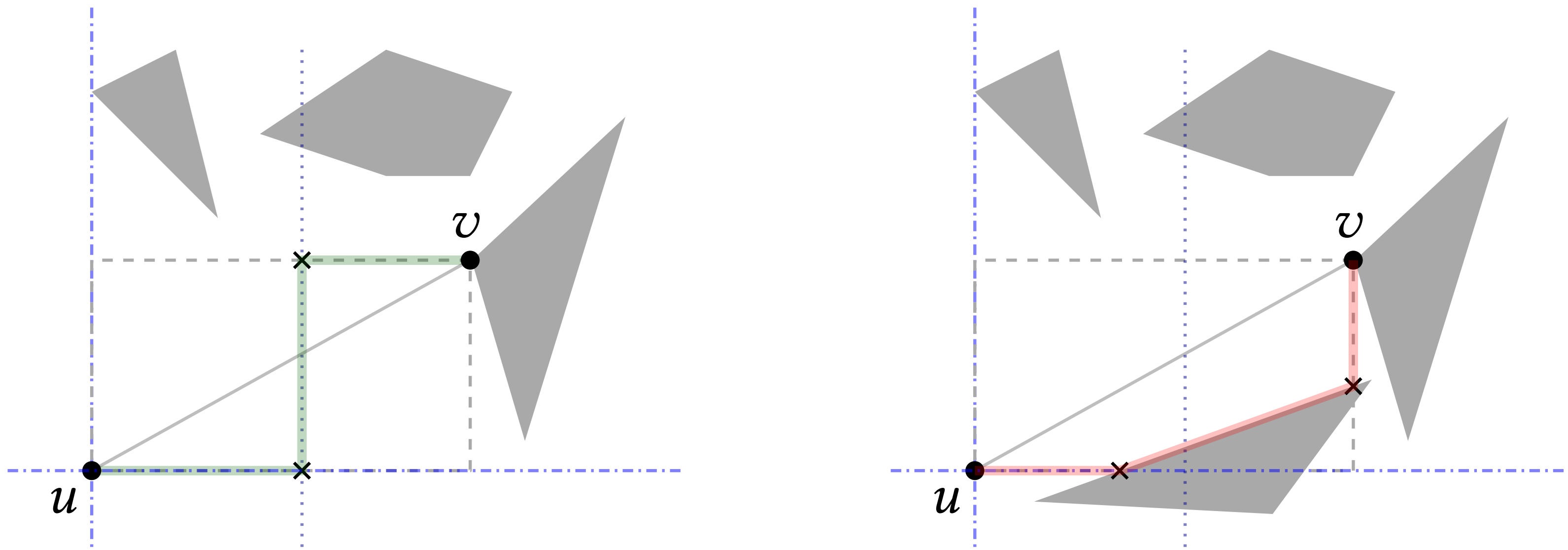
- Do not need all $O(n)$ edges adjacent to u
- Take projections on a 'split line' ℓ
- Also need at most four projections on neighboring obstacles

A Sparse Viability Graph

Inspiration: L_1 Visibility Graphs : [Clarkson-Kapoor-Vaidya'87]

Main Idea:

Total size: $O(n \log n)$



- Do not need all $O(n)$ edges adjacent to u
- Take projections on a 'split line' ℓ
- Also need at most four projections on neighboring obstacles

We show how to construct $O(n \log n)$ **size** L_1 **Viability Graphs**

edges can go through obstacles

Algorithm Overview

- Will reuse idea of taking split line projections for [CKV'87]

Algorithm Overview

- Will reuse idea of taking split line projections for [CKV'87]

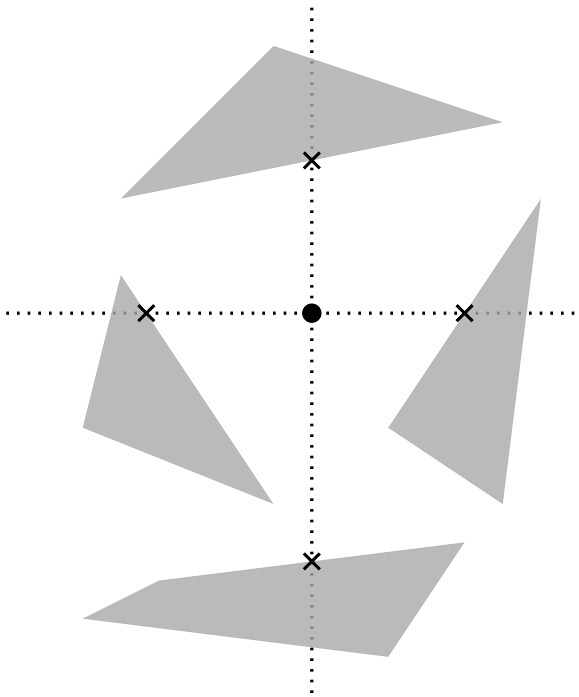
  can be naturally extended to edges of viability graph

Algorithm Overview

- Will reuse idea of taking split line projections for [CKV'87]

  can be naturally extended to edges of viability graph

However, taking projections on four neighboring obstacles is not sufficient



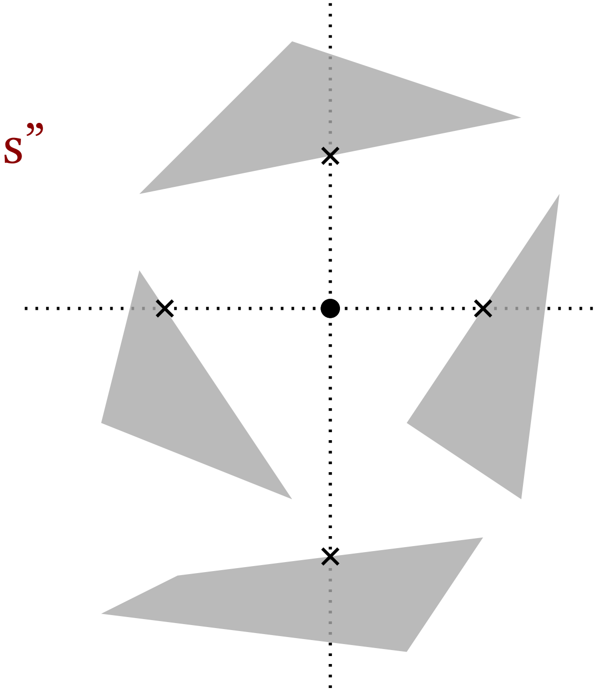
Algorithm Overview

- Will reuse idea of taking split line projections for [CKV'87]

  can be naturally extended to edges of viability graph

However, taking projections on four neighboring obstacles is not sufficient

We circumvent this problem by adding "bypass vertices"



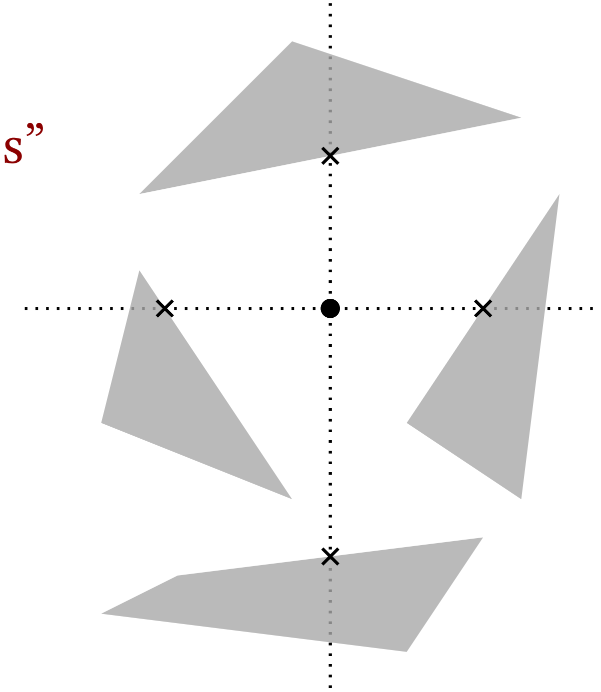
Using this model we give an arguably simpler proof of correctness for the more general **viability graphs**

Algorithm Overview

- Will reuse idea of taking split line projections for [CKV'87]

  can be naturally extended to edges of viability graph

However, taking projections on four neighboring obstacles is not sufficient



We circumvent this problem by adding "bypass vertices"

Using this model we give an arguably simpler proof of correctness for the more general **viability graphs**

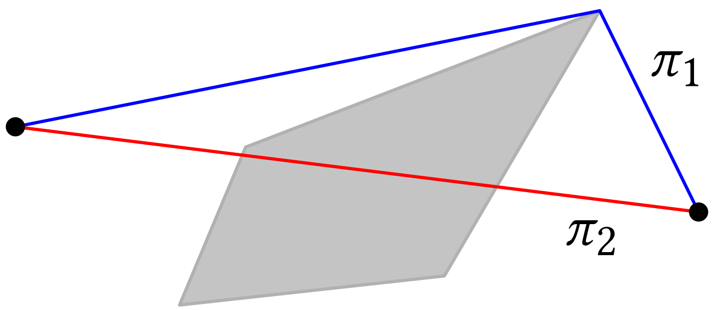
Next we describe an algorithm for constructing an L_1 **viability graph** VG_1

Obstacles with Costs ⇒ Segments with Costs

Obstacles with Costs ⇒ Segments with Costs

Obstacles are **convex** and **disjoint**

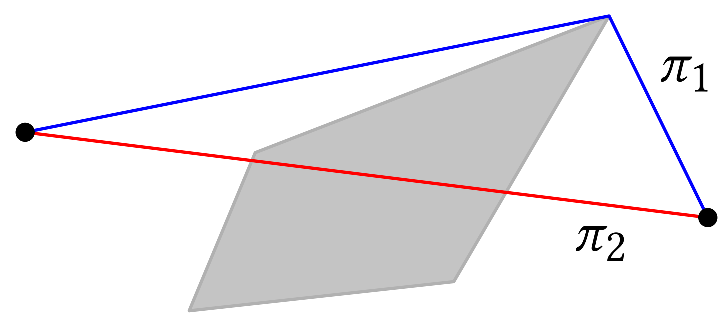
⇒ Shortest paths intersect zero or exactly two sides of an obstacle



Obstacles with Costs ⇒ Segments with Costs

Obstacles are **convex** and **disjoint**

⇒ Shortest paths intersect zero or exactly two sides of an obstacle

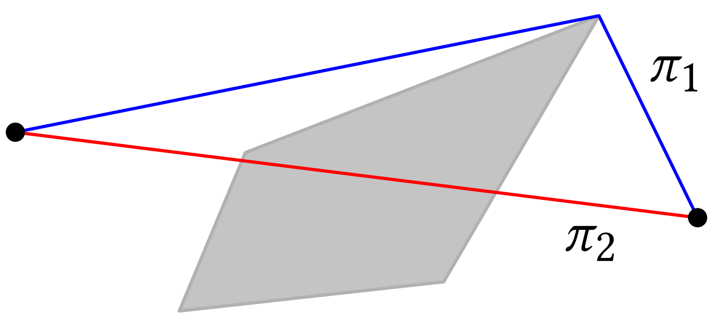


Assign cost $c_i/2$ to all segments of an obstacle that has cost c_i

Obstacles with Costs ⇒ Segments with Costs

Obstacles are **convex** and **disjoint**

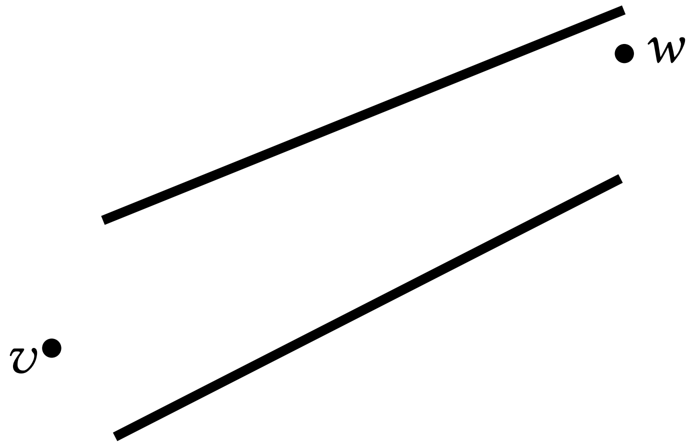
⇒ Shortest paths intersect zero or exactly two sides of an obstacle



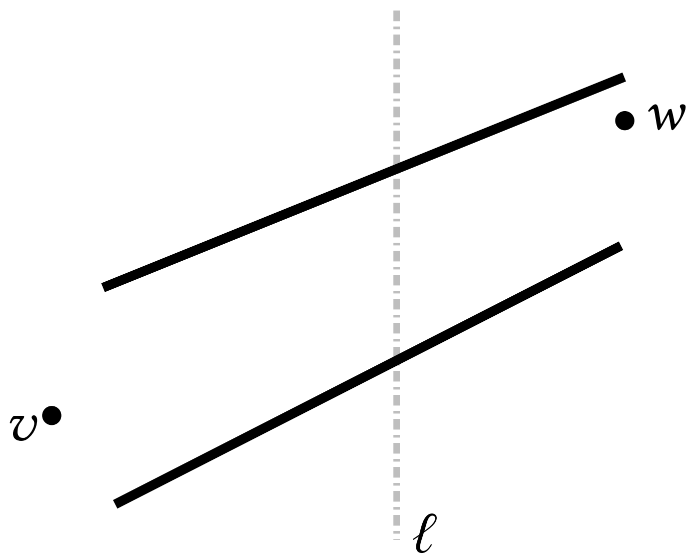
Assign cost $c_i/2$ to all segments of an obstacle that has cost c_i

Allows us to reason about geometry of line segments

Algorithm Description

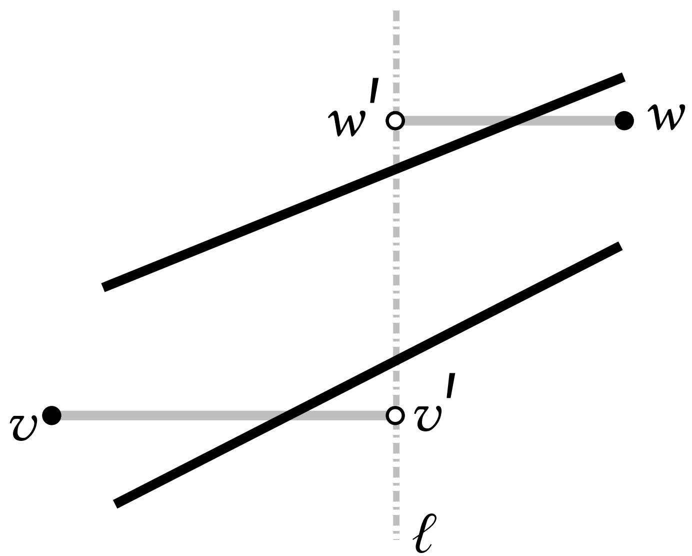


Algorithm Description



- Find the split line ℓ that splits vertices into equal sized sets V_l and V_r

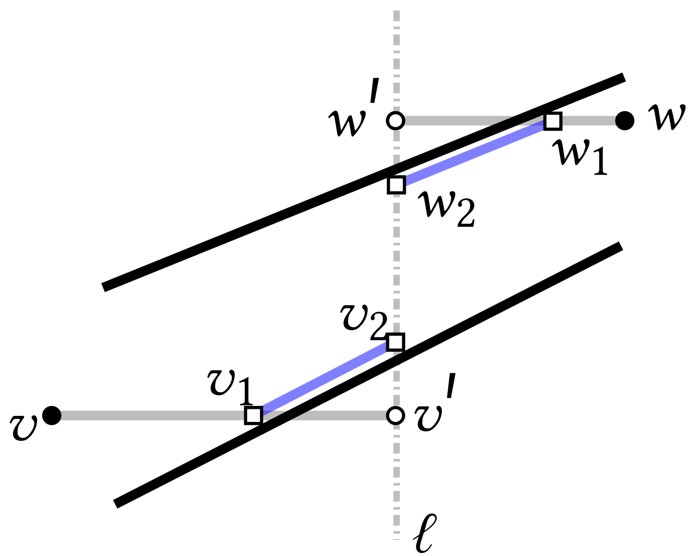
Algorithm Description



– Cost of **projection segment** is the sum of costs of all obstacle segments that it intersects

- Find the split line ℓ that splits vertices into equal sized sets V_l and V_r
- Let v' be the projection of v on this split line ℓ
  - Add vertex v' and edge vv' with cost $c(vv')$

Algorithm Description

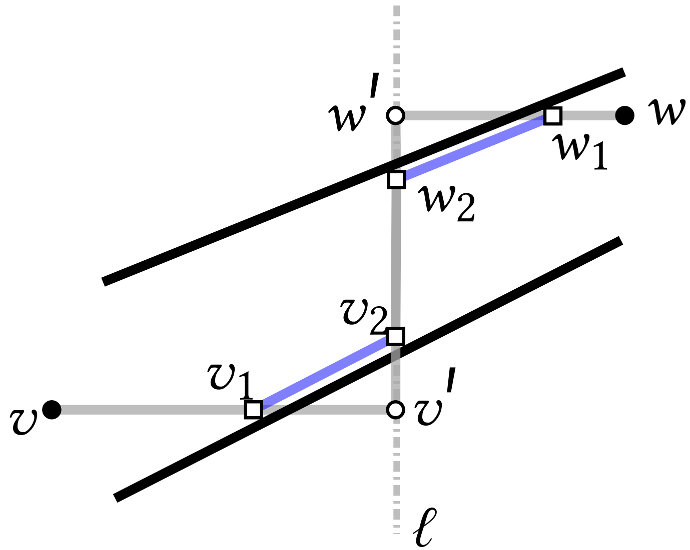


– Cost of **projection segment** is the sum of costs of all obstacle segments that it intersects

– Bypass edges $v_1 v_2$ have cost $c(v_1 v_2) = 0$

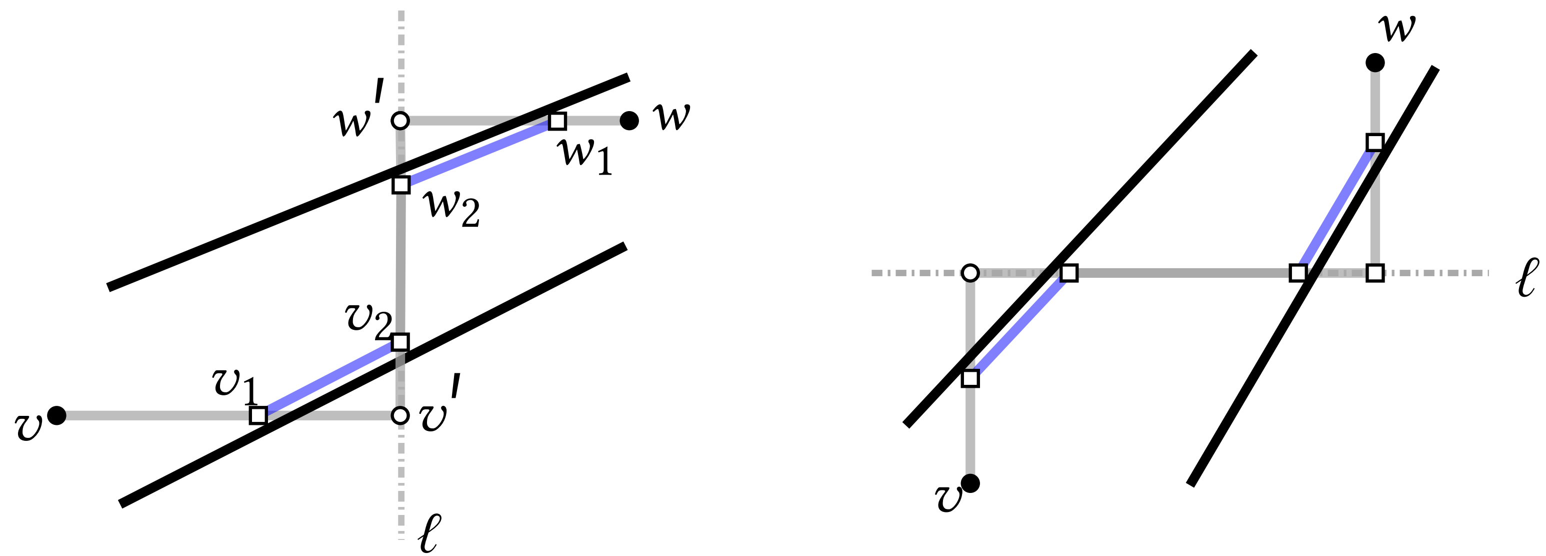
- Find the split line ℓ that splits vertices into equal sized sets V_l and V_r
- Let v' be the projection of v on this split line ℓ
  - Add vertex v' and edge vv' with cost $c(vv')$
- Find first segment s' of *positive slope* that intersects ℓ
  - Add two bypass vertices v_1, v_2 and edges vv' and $v_1 v_2$

Algorithm Description



- Find the split line ℓ that splits vertices into equal sized sets V_l and V_r
- Let v' be the projection of v on this split line ℓ
  - Add vertex v' and edge vv' with cost $c(vv')$
- Find first segment s' of *positive slope* that intersects ℓ
  - Add two bypass vertices v_1, v_2 and edges vv' and v_1v_2
- Connect consecutive vertices on ℓ and obstacle boundary

Algorithm Description



- Find the split line ℓ that splits vertices into equal sized sets V_l and V_r
- Let v' be the projection of v on this split line ℓ
  - Add vertex v' and edge vv' with cost $c(vv')$
- Find first segment s' of *positive slope* that intersects ℓ
  - Add two bypass vertices v_1, v_2 and edges vv' and v_1v_2
- Connect consecutive vertices on ℓ and obstacle boundary
- Repeat for a horizontal split line. RECURSE on sets V_l and V_r.

Correctness

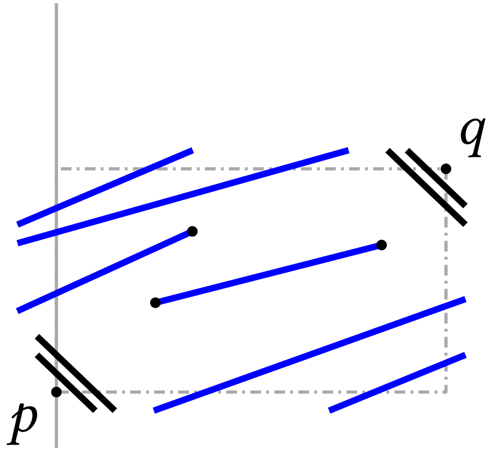
Suffices to show: for any pair of obstacle vertices p, q there exists a path π_{pq} such that the L_1 length $\|\pi_{pq}\|_1 \leq \|pq\|_1$ and cost $c(\pi_{pq}) \leq c(pq)$

Proof Idea:

Correctness

Suffices to show: for any pair of obstacle vertices p, q there exists a path π_{pq} such that the L_1 length $\|\pi_{pq}\|_1 \le \|pq\|_1$ and cost $c(\pi_{pq}) \le c(pq)$

Proof Idea:

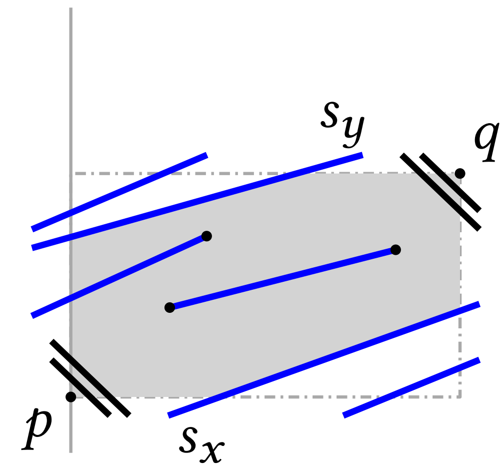


Correctness

Suffices to show: for any pair of obstacle vertices p, q there exists a path π_{pq} such that the L_1 length $||\pi_{pq}||_1 \leq ||pq||_1$ and cost $c(\pi_{pq}) \leq c(pq)$

Proof Idea:

Define a region (clipped rectangle) R_{pq}



Correctness

Suffices to show: for any pair of obstacle vertices p, q there exists a path π_{pq}

such that the L_1 length $||\pi_{pq}||_1 \leq ||pq||_1$ and cost $c(\pi_{pq}) \leq c(pq)$

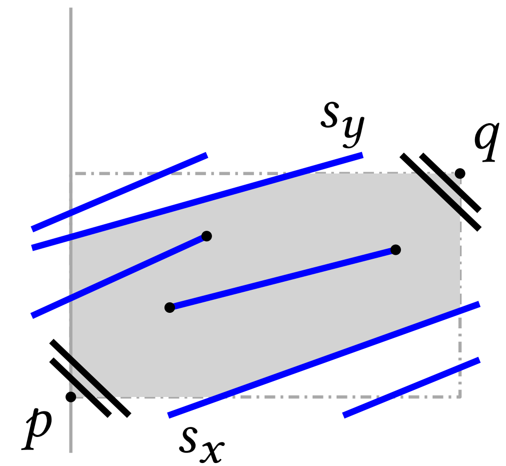
Proof Idea:

Define a region (clipped rectangle) R_{pq}

Proof by induction on number of vertices in R_{pq}

– If R_{pq} contains an obstacle vertex

– If R_{pq} does not contains an obstacle vertex



Correctness

Suffices to show: for any pair of obstacle vertices p, q there exists a path π_{pq} such that the L_1 length $\|\pi_{pq}\|_1 \leq \|pq\|_1$ and cost $c(\pi_{pq}) \leq c(pq)$

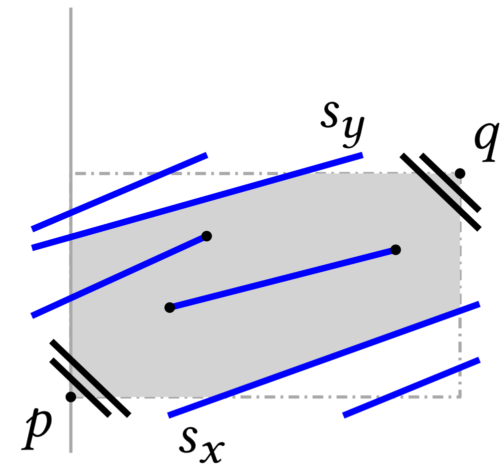
Proof Idea:

Define a region (clipped rectangle) R_{pq}

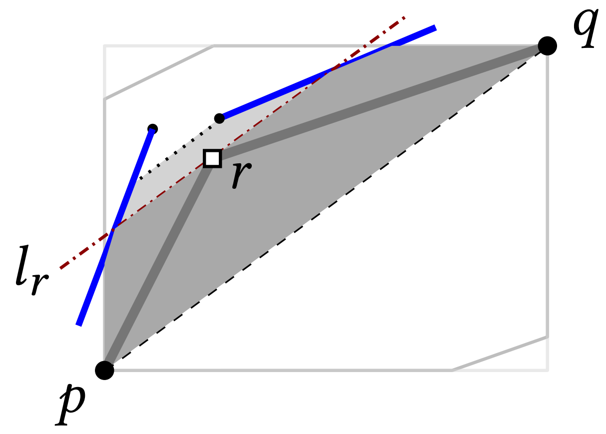
Proof by induction on number of vertices in R_{pq}

– If R_{pq} contains an obstacle vertex
(Induction Step)

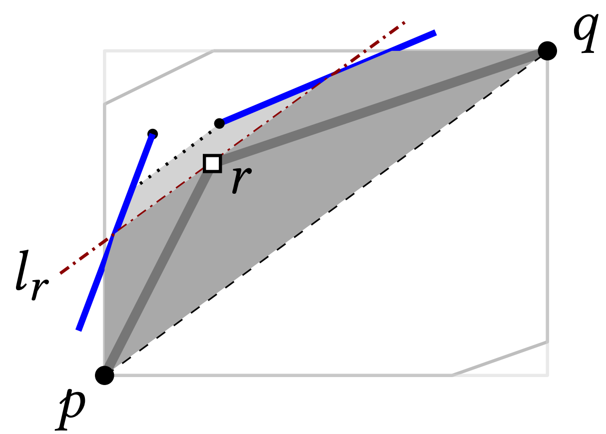
– If R_{pq} does not contains an obstacle vertex
(Base Case)



Induction Step

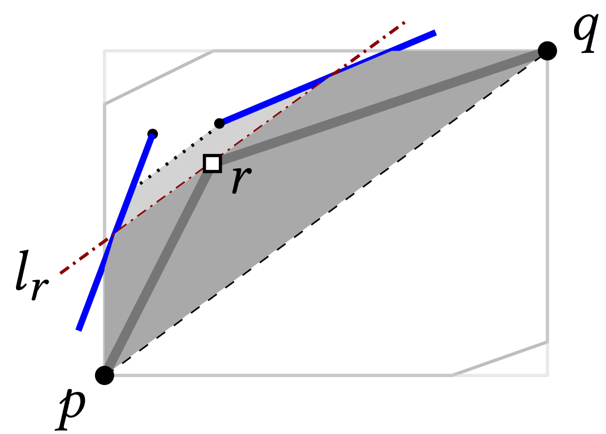


Induction Step



We show how to find an **intermediate vertex** r such that

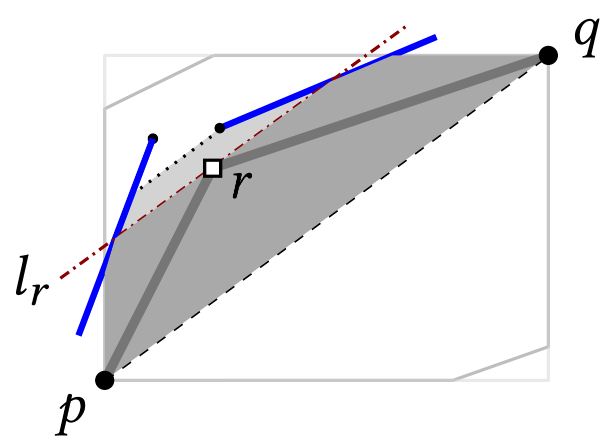
Induction Step



We show how to find an **intermediate vertex** r such that

such that $||pr||_1 + ||rq||_1 \leq ||pq||_1$ and cost $c(pr) + c(rq) \leq c(pq)$

Induction Step

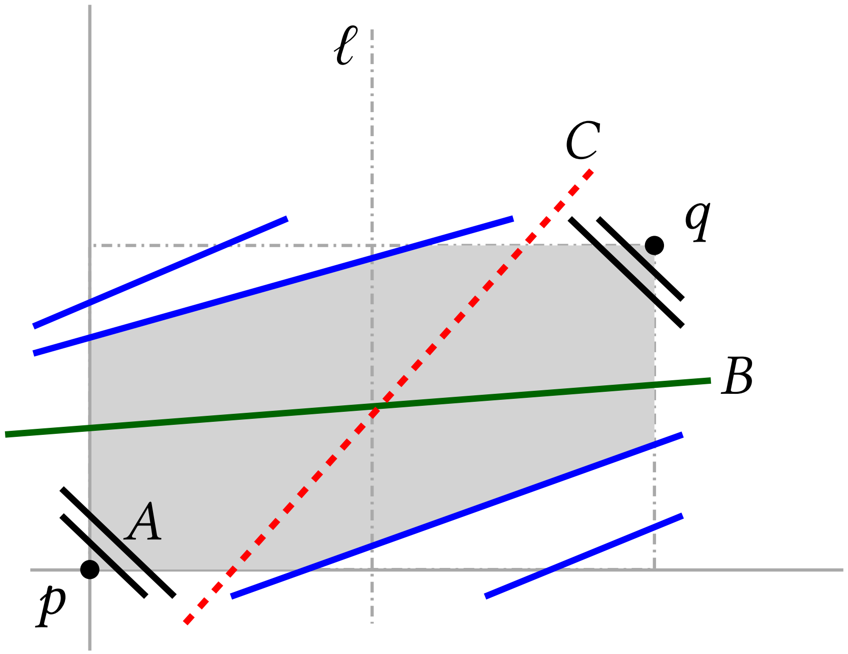


We show how to find an **intermediate vertex** r such that

such that $\|pr\|_1 + \|rq\|_1 \leq \|pq\|_1$ and cost $c(pr) + c(rq) \leq c(pq)$

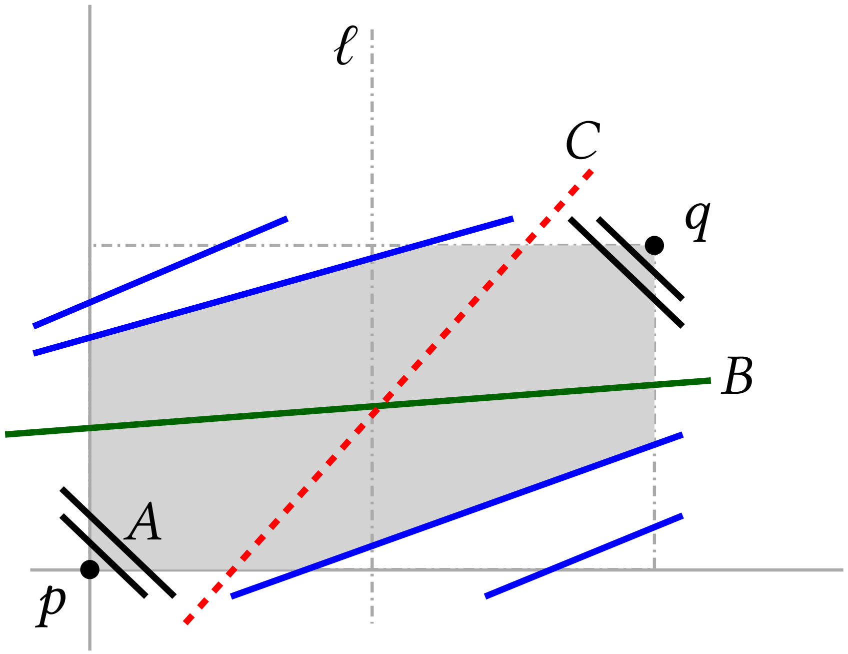
Details are technical, relies heavily on **disjointness of obstacle segments**

Base Case



Identify three types of edges A, B and C

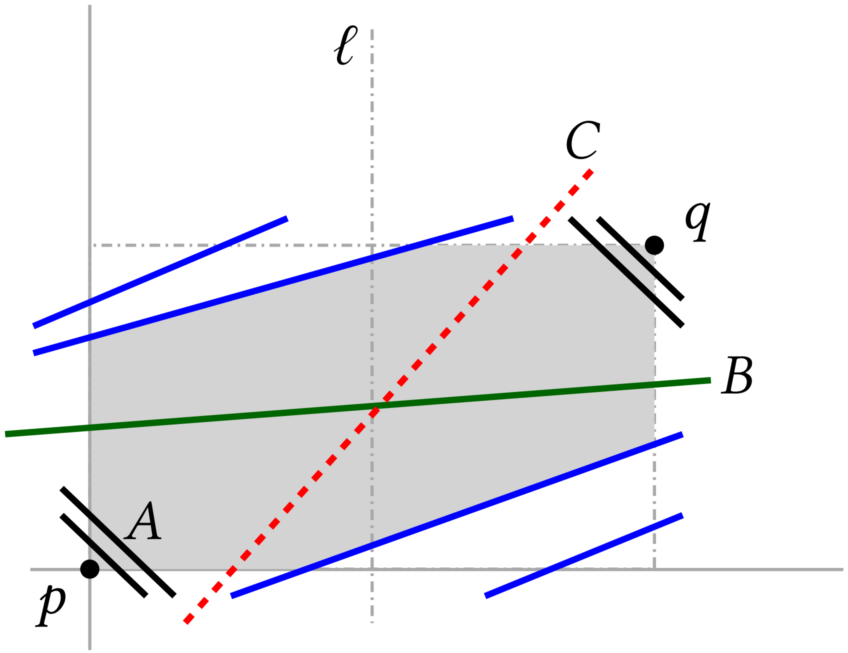
Base Case



Identify three types of edges A, B and C

– Type B and C edges cannot co-exist

Base Case



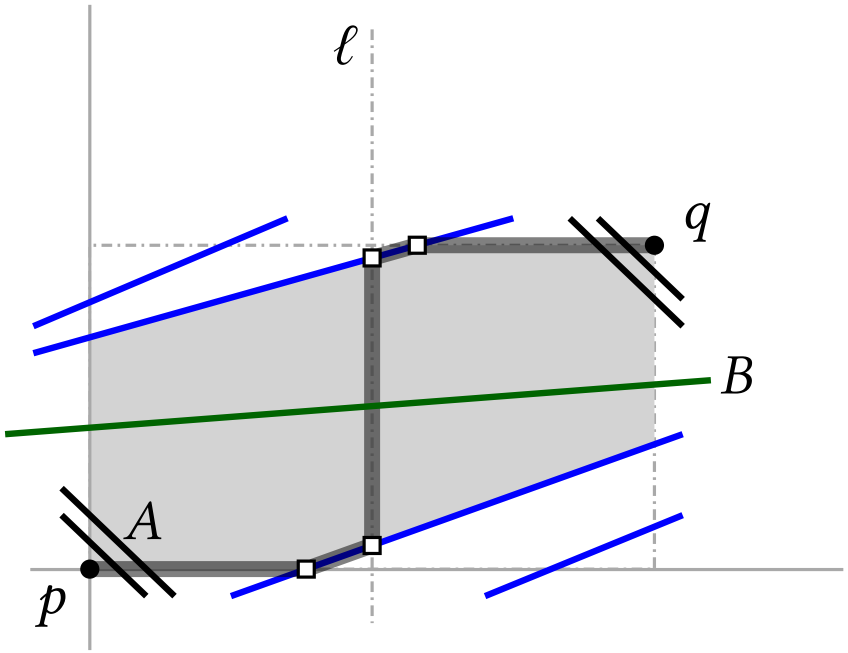
Identify three types of edges A, B and C

– Type B and C edges cannot co-exist

Type B edge + vertical split line

≡ Type C edge + horizontal split line

Base Case



Identify three types of edges A, B and C

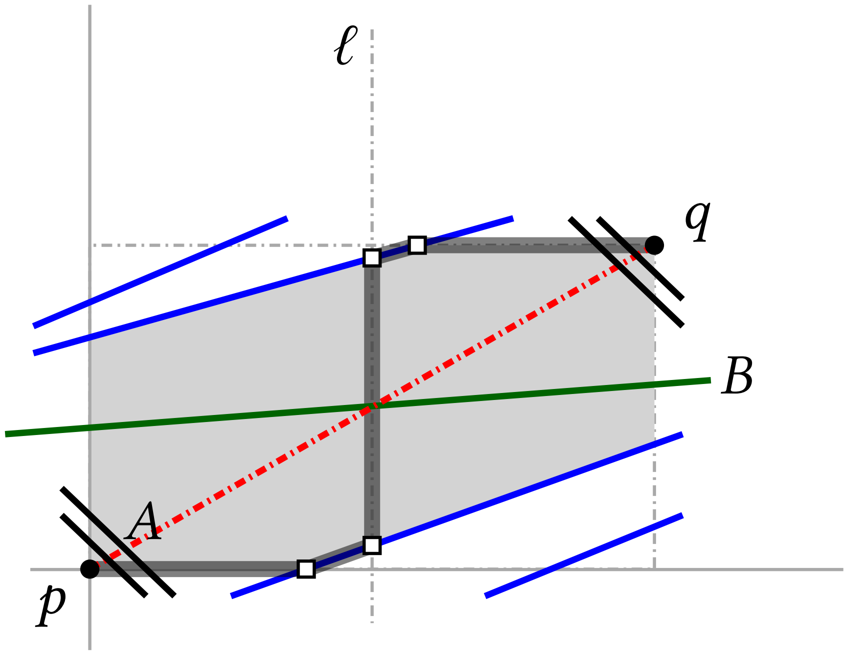
– Type B and C edges cannot co-exist

Type B edge + vertical split line

≡ Type C edge + horizontal split line

Exists a path π_{pq} in VG_1 using projection and bypass vertices

Base Case



Identify three types of edges A, B and C

– Type B and C edges cannot co-exist

Type B edge + vertical split line

$\equiv$ Type C edge + horizontal split line

Exists a path π_{pq} in VG_1 using projection and bypass vertices

Easy to verify that π_{pq} has same L_1 length and cost as the segment pq

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**

– trade a factor $(1+\epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

– Create $1/\epsilon$ copies of VG_1, one per direction and combine them to obtain VG_ϵ

Preserves pairwise distances within a factor of $(1+\epsilon)$

– Run the slower FPTAS on $O(\frac{n}{\epsilon} \log n)$ size viability graph VG_ϵ

Primary challenge is to construct the graph VG_1

A Faster FPTAS

Key idea is to construct a sparse **Viability Graph**
– trade a factor $(1 + \epsilon)$ in path length for efficiency

Roadmap

– Construct an $O(n \log n)$ size viability graph VG_1 preserving L_1 distances

– Create $1/\epsilon$ copies of VG_1, one per direction and combine them to obtain VG_ϵ

Preserves pairwise distances within a factor of $(1 + \epsilon)$

– Run the slower FPTAS on $O(\frac{n}{\epsilon} \log n)$ size viability graph VG_ϵ

Primary challenge is to construct the graph VG_1 ✓

In Summary

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.
- A simple $(1+\epsilon)$-approximation follows using the idea of **viability graphs** (an extension of visibility graphs)

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.
- A simple $(1+\epsilon)$-approximation follows using the idea of **viability graphs** (an extension of visibility graphs)
- Obtain a faster approximation by constructing a sparse viability graph

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.
- A simple $(1+\epsilon)$-approximation follows using the idea of **viability graphs** (an extension of visibility graphs)
- Obtain a faster approximation by constructing a sparse viability graph
- Obtained data structures for approximate shortest path queries running in polylogarithmic time

In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.
- A simple $(1+\epsilon)$-approximation follows using the idea of **viability graphs** (an extension of visibility graphs)
- Obtain a faster approximation by constructing a sparse viability graph
- Obtained data structures for approximate shortest path queries running in polylogarithmic time
- Applied these results to solve a shortest path problem in a **stochastic model** of obstacles

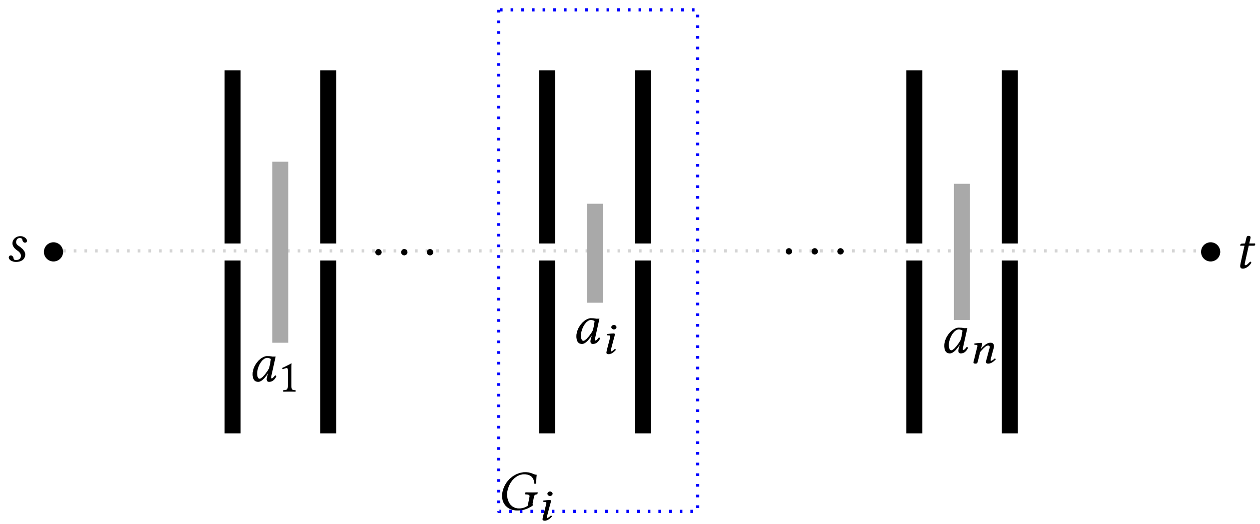
In Summary

- Motivated by applications, we study the problem of computing shortest path among removable obstacles
- We show that computing such a path exactly is NP-hard.
- A simple $(1 + \epsilon)$-approximation follows using the idea of **viability graphs** (an extension of visibility graphs)
- Obtain a faster approximation by constructing a sparse viability graph
- Obtained data structures for approximate shortest path queries running in polylogarithmic time
- Applied these results to solve a shortest path problem in a **stochastic model** of obstacles

Thanks!

Backup : NP-Hardness

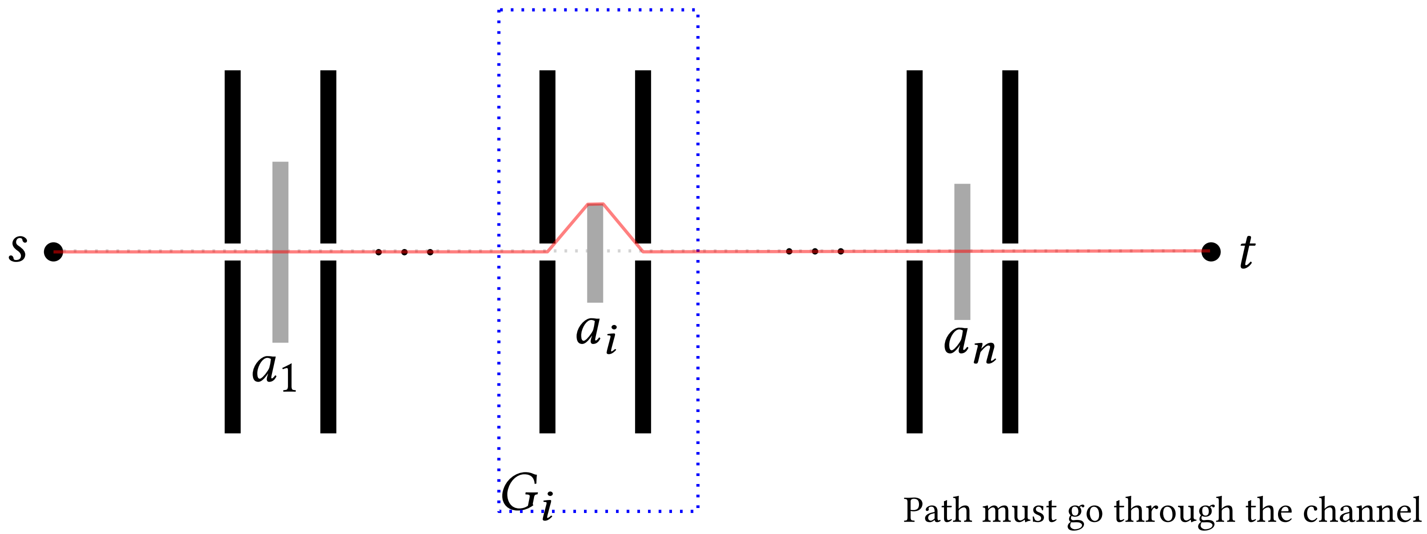
Simple reduction from PARTITION



A is a set of integers, a_i corresponds to group G_i

Backup : NP-Hardness

Simple reduction from PARTITION

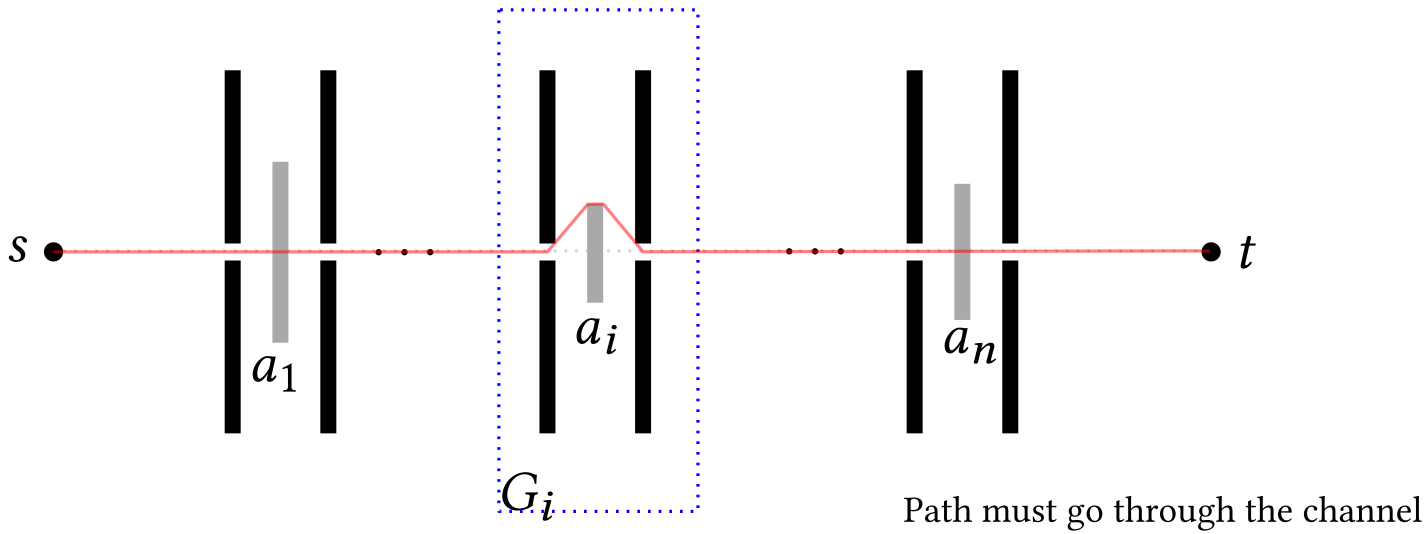


A is a set of integers, a_i corresponds to group G_i

- Go through the middle obstacle by paying cost a_i
- Go around it by adding a detour of length a_i

Backup : NP-Hardness

Simple reduction from Partition



A is a set of integers, a_i corresponds to group G_i

- Go through the middle obstacle by paying cost a_i
- Go around it by adding a detour of length a_i

An s–t path with length at most $L = \frac{1}{2}\sum a_i$ and cost at most $C = \frac{1}{2}\sum a_i$ exists if and only if set A can be partitioned into two equal groups.