

Fast Obstacle k-Nearest Neighbour Query on Navigation Mesh

Final Presentation

Shizhe Zhao (27505928)

Supervisors: David Taniar, Daniel Harabor



Outline

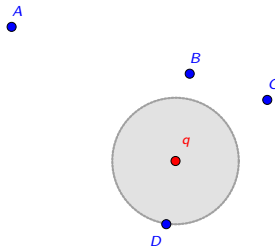
- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research
- 6 Experiments
- 7 Conclusion and future work
- 8 End



Traditional k-Nearest Neighbor

k-Nearest Neighbor:

■ Given:

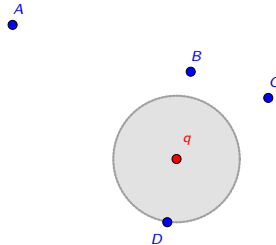


Traditional k-Nearest Neighbor

k-Nearest Neighbor:

- Given:

- q : query point

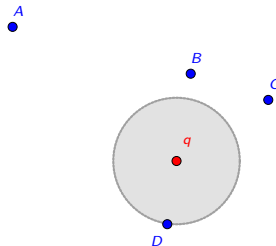


Traditional k-Nearest Neighbor

k-Nearest Neighbor:

■ Given:

- q : query point
- T : target set
(e.g. $\{A, B, C, D\}$)

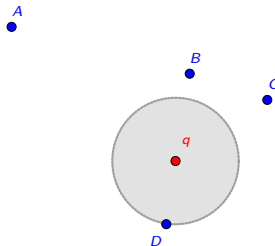


Traditional k-Nearest Neighbor

k-Nearest Neighbor:

■ Given:

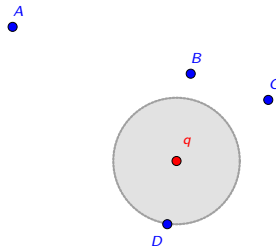
- q : query point
- T : target set
(e.g. $\{A, B, C, D\}$)
- k : number of retrieved targets (e.g. $k = 1$)



Traditional k-Nearest Neighbor

k-Nearest Neighbor:

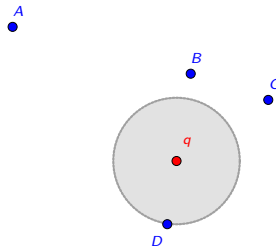
- Given:
 - q : query point
 - T : target set
(e.g. $\{A, B, C, D\}$)
 - k : number of retrieved targets (e.g. $k = 1$)
- Return:
top k nearest targets regarding
Euclidean distance d_e



Traditional k-Nearest Neighbor

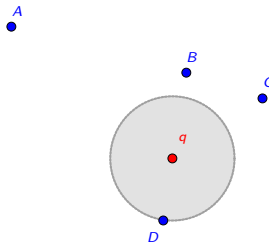
k-Nearest Neighbor:

- Given:
 - q : query point
 - T : target set
(e.g. $\{A, B, C, D\}$)
 - k : number of retrieved targets (e.g. $k = 1$)
- Return:
 - top k nearest targets regarding *Euclidean distance* d_e
- the circle indicates that D is the nearest neighbor of q



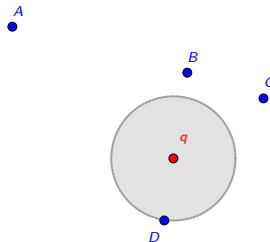
Obstacle k-Nearest Neighbor

- traditional kNN has been well studied.
- when take obstacles into consideration...
- metric: Obstacle distance d_o



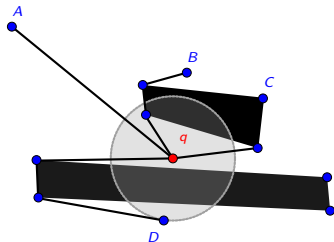
Obstacle k-Nearest Neighbor

- traditional kNN has been well studied.
- when take obstacles into consideration...
- metric: Obstacle distance d_o



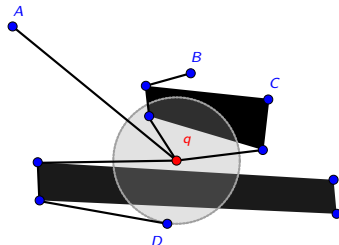
Obstacle k-Nearest Neighbor

- traditional kNN has been well studied.
- when take obstacles into consideration...
- metric: Obstacle distance d_o



Application Scenario

In an industrial warehouse,
 q is a robot.
It's interested in the closest storage
locations,
but it can not cross obstacles



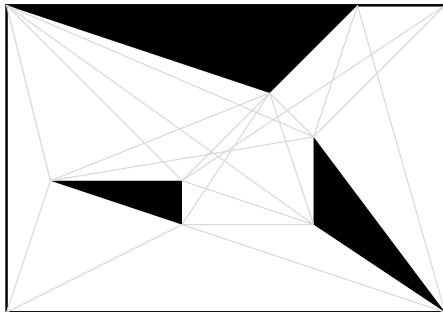
Outline

- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research
- 6 Experiments
- 7 Conclusion and future work
- 8 End



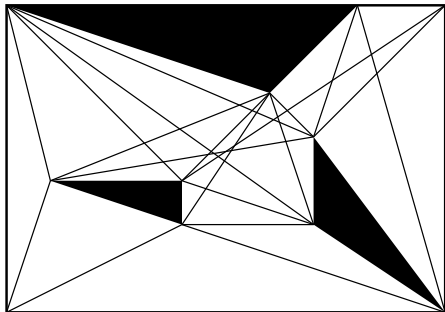
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
 - any pair of visible points has an edge
- Run shortest path algorithm on VG (e.g. *Dijkstra*)
- Number of edge: up to $O(V^2)$
(V : the number of vertex)



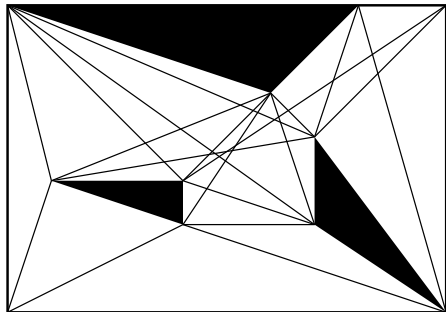
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
 - any pair of visible points has an edge
- Run shortest path algorithm on VG (e.g. *Dijkstra*)
- Number of edge: up to $O(V^2)$
(V : the number of vertex)



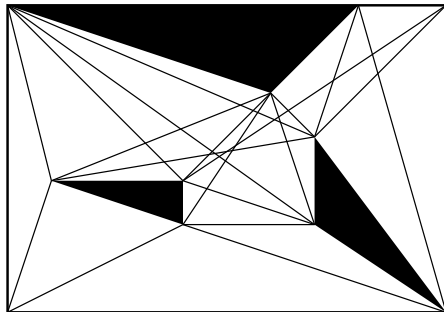
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
 - any pair of visible points has an edge
- Run shortest path algorithm on VG (e.g. *Dijkstra*)
- Number of edge: up to $O(V^2)$
(V : the number of vertex)



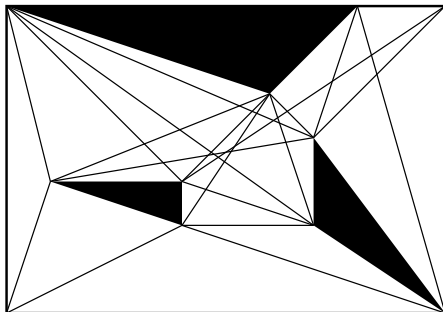
How to compute Obstacle Distance

- Existing works rely on *visibility graph* (VG)
 - any pair of visible points has an edge
- Run shortest path algorithm on VG
(e.g. *Dijkstra*)
- Number of edge: up to $O(V^2)$
(V : the number of vertex)



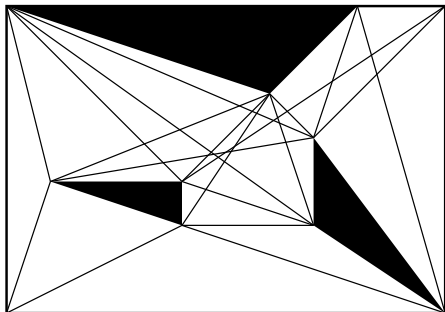
How to compute Obstacle Distance

- Global VG: expansive
- Motivation: only consider query related area
- *Zhang, EDBT 2004: Local Visibility Graph (LVG)*



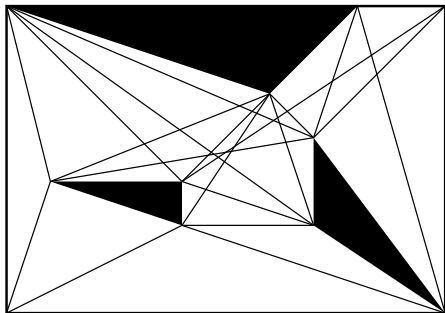
How to compute Obstacle Distance

- Global VG: expansive
- Motivation: only consider query related area
- *Zhang, EDBT 2004: Local Visibility Graph (LVG)*



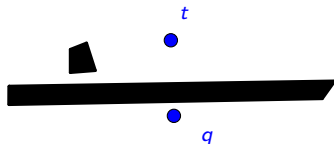
How to compute Obstacle Distance

- Global VG: expansive
- Motivation: only consider query related area
- *Zhang, EDBT 2004: Local Visibility Graph (LVG)*



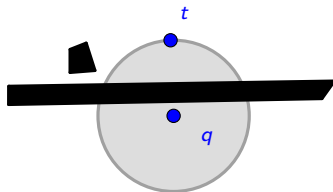
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_o(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



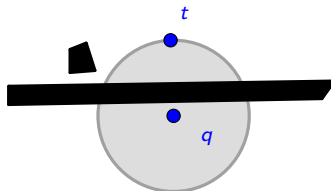
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



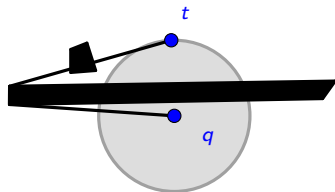
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



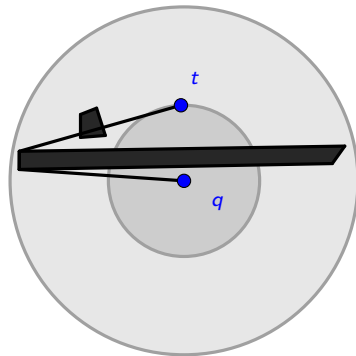
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



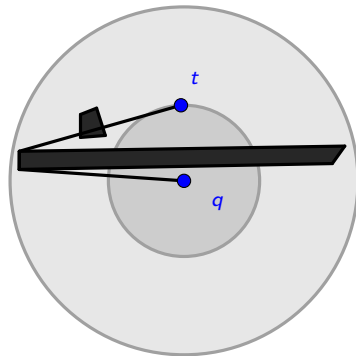
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



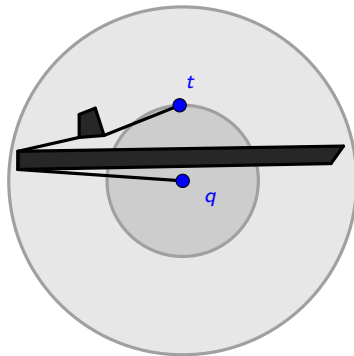
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



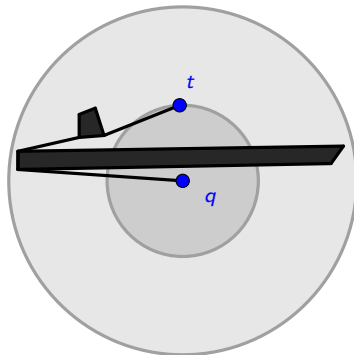
Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



Obstacle Distance Computation: *LVG*

- Given: q, t
- Start with a small VG in $circle(q, r)$
 - $r = d_e(q, t)$
- Compute shortest path on current VG
- Enlarge the circle
 - update VG incrementally
 - compute new shortest path
- Terminate when $r > d_o(q, t)$



Outline

- 1 Introduction
- 2 Related works
- 3 Challenges**
- 4 New Framework
- 5 My research
- 6 Experiments
- 7 Conclusion and future work
- 8 End



State of the art

- The *LVG* algorithm is widely used in many Obstacle Spatial Query Processing.



State of the art

- The *LVG* algorithm is widely used in many Obstacle Spatial Query Processing.
 - It can be easily extended to multi-targets scenario



State of the art

- The *LVG* algorithm is widely used in many Obstacle Spatial Query Processing.
 - It can be easily extended to multi-targets scenario
- It's still the state-of-the-art.



State of the art

- The *LVG* algorithm is widely used in many Obstacle Spatial Query Processing.
 - It can be easily extended to multi-targets scenario
- It's still the state-of-the-art.
- However ...



Disadvantages

It has some disadvantages:

- Costly online visibility checking
- An incremental construction can easily reach to $O(V^2)$ edges
- Duplicated effort:
the VG is discarded each time the q changes



Disadvantages

It has some disadvantages:

- Costly online visibility checking
- An incremental construction can easily reach to $O(V^2)$ edges
- Duplicated effort:
the VG is discarded each time the q changes



Disadvantages

It has some disadvantages:

- Costly online visibility checking
- An incremental construction can easily reach to $O(V^2)$ edges
- Duplicated effort:
the VG is discarded each time the q changes



Outline

- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework**
- 5 My research
- 6 Experiments
- 7 Conclusion and future work
- 8 End



Navigation Mesh

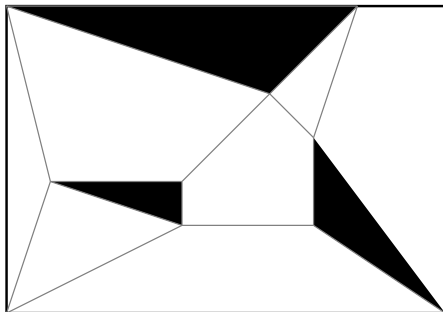
Finally, navigation mesh comes to our sight.



Navigation Mesh

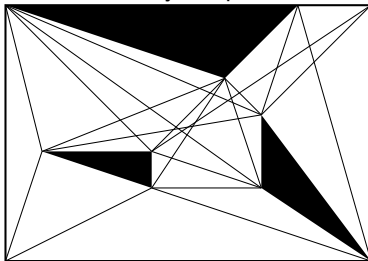
Finally, navigation mesh comes to our sight.

- traversable space \Rightarrow convex polygons

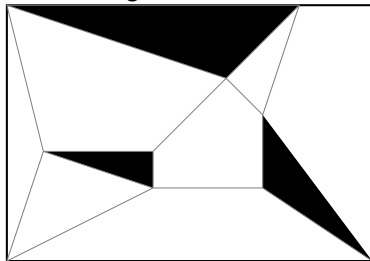


Advantage

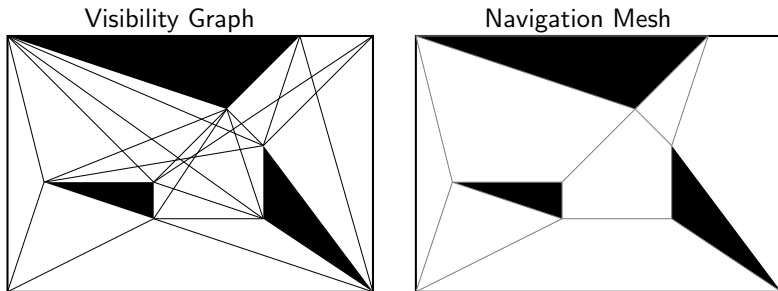
Visibility Graph



Navigation Mesh



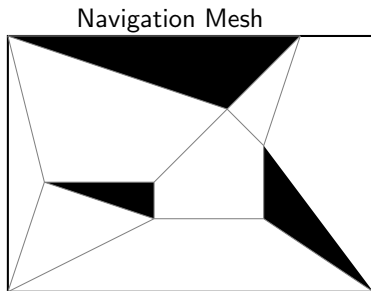
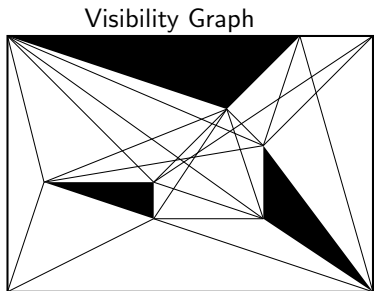
Advantage



We can easily preprocess the entire map!



Advantage



How to compute obstacle distance on a navigation mesh?



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal
 - inefficient



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal
 - inefficient
 - requiring costly preprocessing



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal
 - inefficient
 - requiring costly preprocessing
- But a recent work in 2017: *Polyanya*



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal
 - inefficient
 - requiring costly preprocessing
- But a recent work in 2017: *Polyanya*
 - fast, optimal, flexible



How to compute Obstacle Distance on Navigation Mesh?

- Previous works are not suitable for database scenario
 - not optimal
 - inefficient
 - requiring costly preprocessing
- But a recent work in 2017: *Polyanya*
 - fast, optimal, flexible
 - a new direction for Obstacle kNN query



What's the *Polyanya*?

- Given:
 - a map with polygonal obstacles
 - q : query point
 - t : target
 - a precomputed navigation mesh
 - convex polygon: all inside points are visible
- find the shortest path along meshes

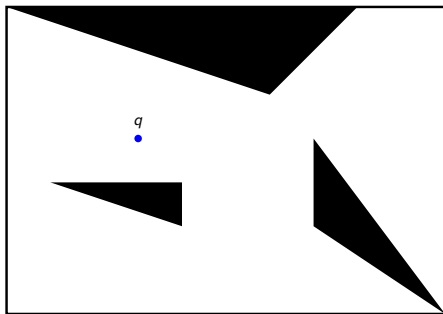


What's the *Polyanya*?

■ Given:

- a map with polygonal obstacles
- q : query point
- t : target
- a precomputed navigation mesh
- convex polygon: all inside points are visible

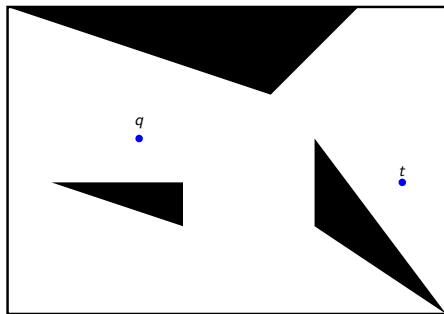
- find the shortest path along meshes



What's the *Polyanya*?

■ Given:

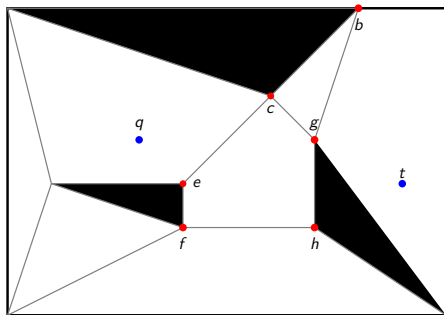
- a map with polygonal obstacles
 - q : query point
 - t : target
 - a precomputed navigation mesh
 - convex polygon: all inside points are visible
- find the shortest path along meshes



What's the *Polyanya*?

■ Given:

- a map with polygonal obstacles
- q : query point
- t : target
- a precomputed navigation mesh
- convex polygon: all inside points are visible
- find the shortest path along meshes

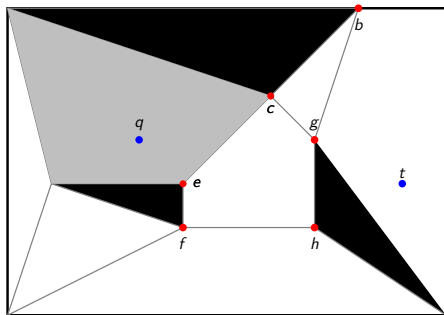


What's the *Polyanya*?

■ Given:

- a map with polygonal obstacles
- q : query point
- t : target
- a precomputed navigation mesh
- convex polygon: all inside points are visible

- find the shortest path along meshes

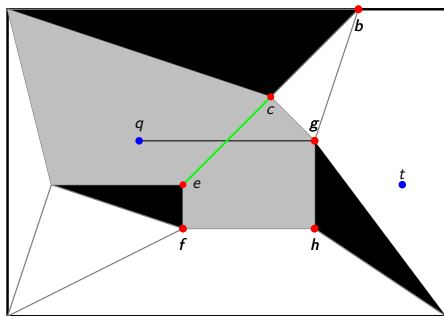


What's the *Polyanya*?

■ Given:

- a map with polygonal obstacles
- q : query point
- t : target
- a precomputed navigation mesh
- convex polygon: all inside points are visible

- find the shortest path along meshes

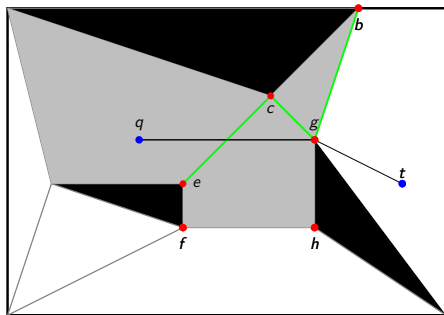


What's the *Polyanya*?

■ Given:

- a map with polygonal obstacles
- q : query point
- t : target
- a precomputed navigation mesh
- convex polygon: all inside points are visible

- find the shortest path along meshes



Polyanya: Overview

Polyanya is an A^* like algorithm, it has three components

- 1 Search Node
- 2 Successors
- 3 Evaluation Function



Polyanya: Overview

Polyanya is an A^* like algorithm, it has three components

- 1 Search Node
- 2 Successors
- 3 Evaluation Function



Polyanya: Overview

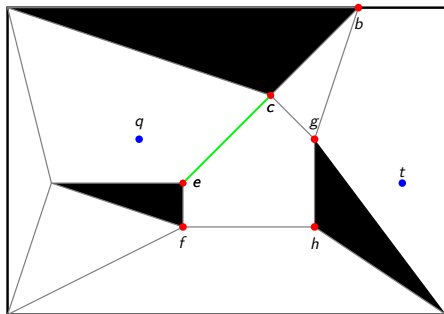
Polyanya is an A^* like algorithm, it has three components

- 1 Search Node
- 2 Successors
- 3 Evaluation Function



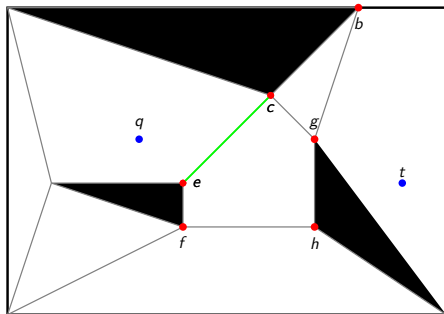
Polyanya: Search Node

- root r : $r \in (V \cup \{q\})$
- interval I : on an edge
- all points $\in I$: visible from r



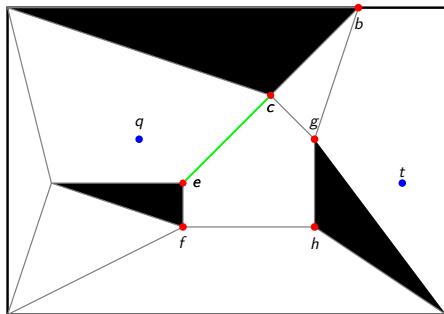
Polyanya: Search Node

- root r : $r \in (V \cup \{q\})$
- interval I : on an edge
- all points $\in I$: visible from r



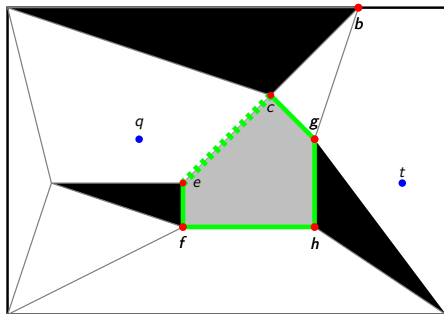
Polyanya: Search Node

- root r : $r \in (V \cup \{q\})$
- interval I : on an edge
- all points $\in I$: visible from r



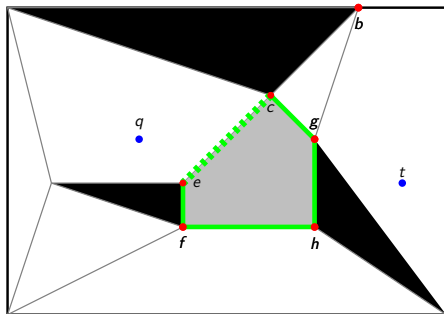
Polyanya: Successors

- Successors are also search nodes
- Generated by pushing the parent node away to adjacent mesh.



Polyanya: Successors

- Successors are also search nodes
- Generated by pushing the parent node away to adjacent mesh.



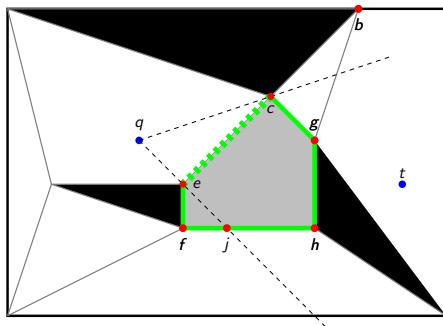
Polyanya: Successors

■ Observable successors

- root: parent's root

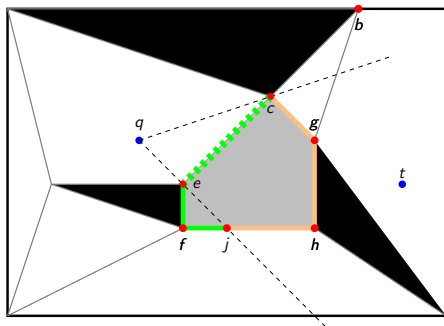
■ Non-observable successors

- root: an end point of l



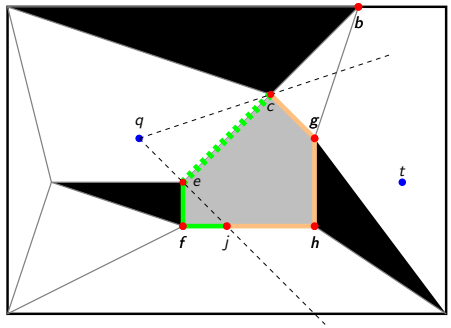
Polyanya: Successors

- **Observable successors**
 - root: parent's root
- **Non-observable successors**
 - root: an end point of l



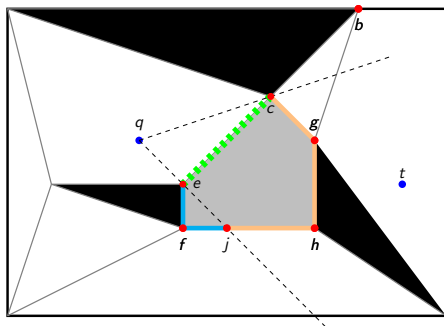
Polyanya: Successors

- **Observable successors**
 - root: parent's root
- **Non-observable successors**
 - root: an end point of l



Polyanya: Successors

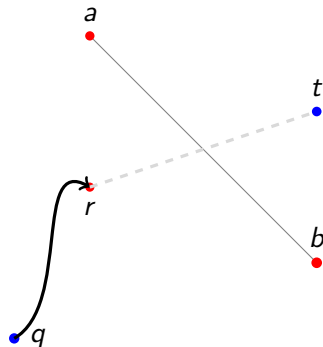
- **Observable successors**
 - root: parent's root
- **Non-observable successors**
 - root: an end point of l



Polyanya: Evaluation Function

Evaluation function of a search node (r, l) has:

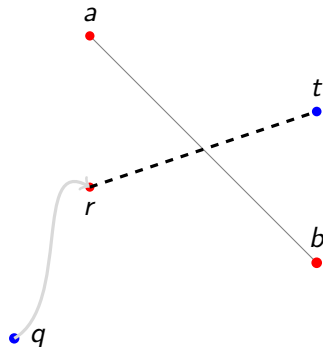
- *g-value*:
 $|shortestPath(q, r)|$
 (certain)
- *h-value*: r to t cross l
 (underestimation)
- *f-value*:
 $g\text{-value} + h\text{-value}$
 (underestimation of
 $|shortestPath(q, t)|$)



Polyanya: Evaluation Function

Evaluation function of a search node (r, l) has:

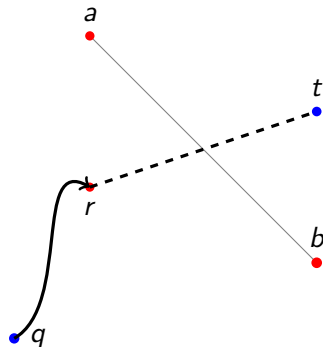
- *g-value*:
 $|shortestPath(q, r)|$
 (certain)
- *h-value*: r to t cross l
 (underestimation)
- *f-value*:
 $g\text{-value} + h\text{-value}$
 (underestimation of
 $|shortestPath(q, t)|$)



Polyanya: Evaluation Function

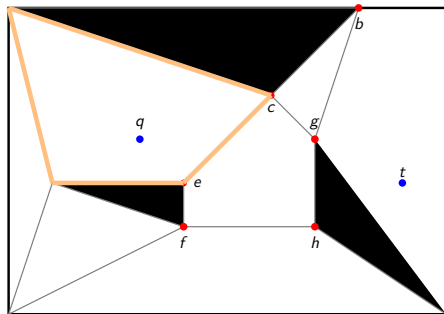
Evaluation function of a search node (r, l) has:

- *g-value*:
 $|shortestPath(q, r)|$
 (certain)
- *h-value*: r to t cross l
 (underestimation)
- *f-value*:
 $g\text{-value} + h\text{-value}$
 (underestimation of $|shortestPath(q, t)|$)



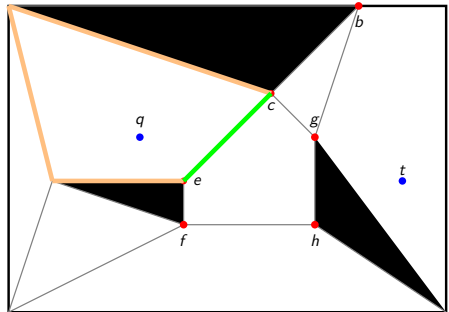
Polyanya: Example

Initial Search Nodes are edges of mesh that contains the q .



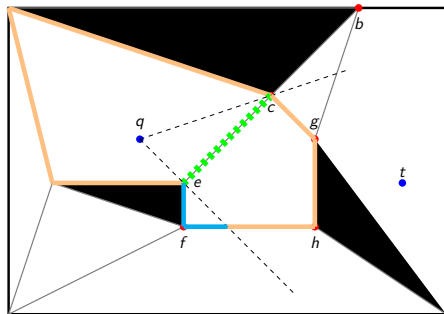
Polyanya: Example

Search Node $(q, [e, c])$ has the best estimation, so popped out



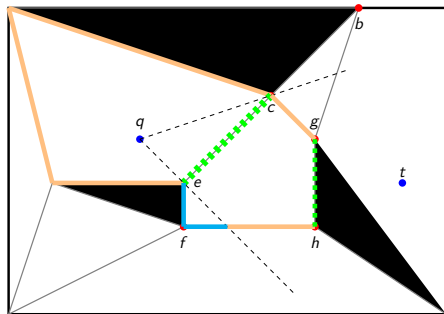
Polyanya: Example

Expand successors in adjacent mesh.

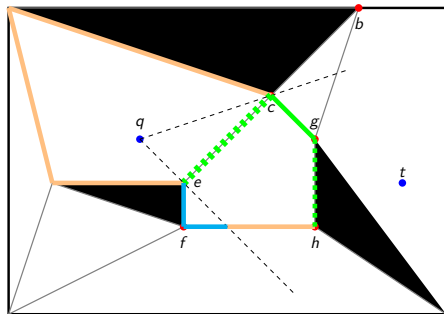


Polyanya: Example

Pop $(q, [g, h])$,
adjacent to obstacle,
so we discard it.

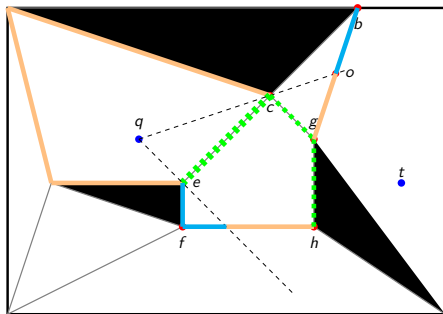


Polyanya: Example

 $\text{Pop}(q, [c, g]).$ 

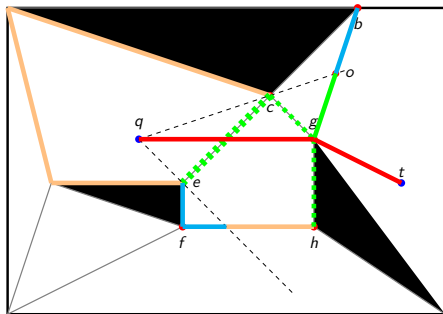
Polyanya: Example

Expand successors.



Polyanya: Example

Pop ($q, [g, o]$),
the adjacent mesh contains t .
We've found the shortest path!



Outline

- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research**
- 6 Experiments
- 7 Conclusion and future work
- 8 End



My Research

- *Polyanya* only work for single pair shortest path
- My research:
 - multi-targets search based on framework of *Polyanya*
 - with good scalability



My Research

- *Polyanya* only work for single pair shortest path
- My research:
 - multi-targets search based on framework of *Polyanya*
 - with good scalability



My Research

- *Polyanya* only work for single pair shortest path
- My research:
 - multi-targets search based on framework of *Polyanya*
 - with good scalability



My Research

- *Polyanya* only work for single pair shortest path
- My research:
 - multi-targets search based on framework of *Polyanya*
 - with good scalability



Proposed algorithm 1: brute-force *Polyanya*

A naive solution is calling *Polyanya* for each target:

```
for t in targets:  
    polyanya.run(q, t)
```

- Drawback: inefficient when targets many.



Proposed algorithm 1: brute-force *Polyanya*

A naive solution is calling *Polyanya* for each target:

```
for t in targets:  
    polyanya.run(q, t)
```

- Drawback: inefficient when targets many.



Proposed algorithm 1: brute-force *Polyanya*

A naive solution is calling *Polyanya* for each target:

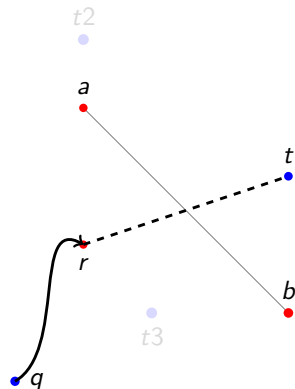
```
for t in targets:  
    polyanya.run(q, t)
```

- Drawback: inefficient when targets many.



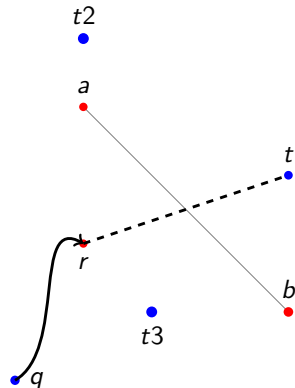
Proposed algorithm 2: interval heuristic

- Let's review the evaluation function in *Polyanya*
- When there are multiple targets...
 - *h-value* shouldn't be affected by a specific target
- How about remove t from *h-value*?



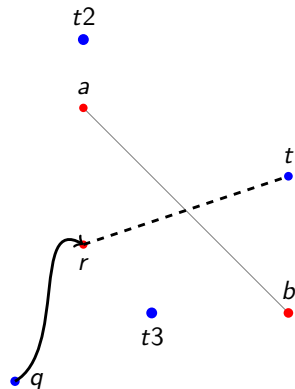
Proposed algorithm 2: interval heuristic

- Let's review the evaluation function in *Polyanya*
- When there are multiple targets...
 - *h-value* shouldn't be affected by a specific target
 - How about remove *t* from *h-value*?



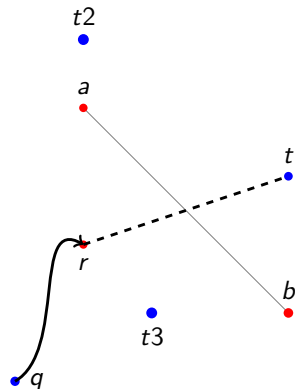
Proposed algorithm 2: interval heuristic

- Let's review the evaluation function in *Polyanya*
- When there are multiple targets...
 - h -value shouldn't be affected by a specific target
- How about remove t from h -value?



Proposed algorithm 2: interval heuristic

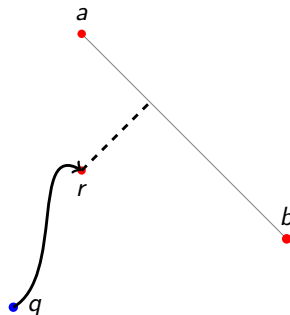
- Let's review the evaluation function in *Polyanya*
- When there are multiple targets...
 - h -value shouldn't be affected by a specific target
- How about removing t from h -value?



Proposed algorithm 2: interval heuristic

Then we get: Interval heuristic

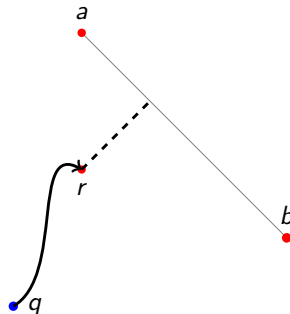
- g -value is same
- h -value: distance from r to l



Proposed algorithm 2: interval heuristic

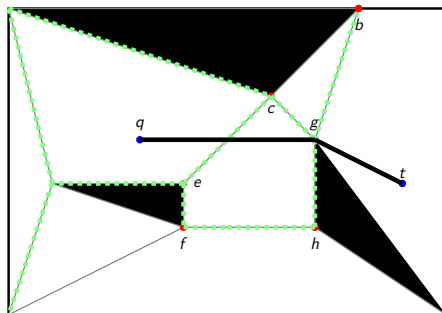
Then we get: Interval heuristic

- g -value is same
- h -value: distance from r to l



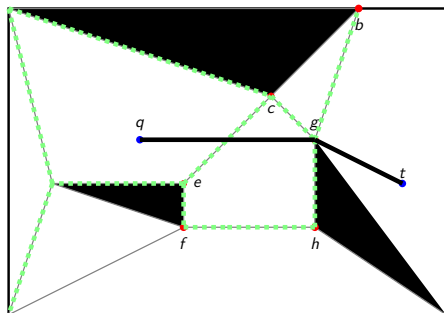
Interval heuristic: drawback

- *interval heuristic* causes redundant expansions
- especially in sparse targets scenario
- e.g.: query is "nearest storage locations where capacity ≥ 100 ".
- motivation: we may need t in h -value to make search smarter



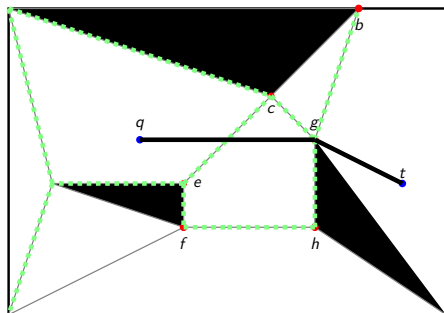
Interval heuristic: drawback

- *interval heuristic* causes redundant expansions
- especially in sparse targets scenario
- e.g.: query is "nearest storage locations where capacity ≥ 100 ".
- motivation: we may need t in h -value to make search smarter



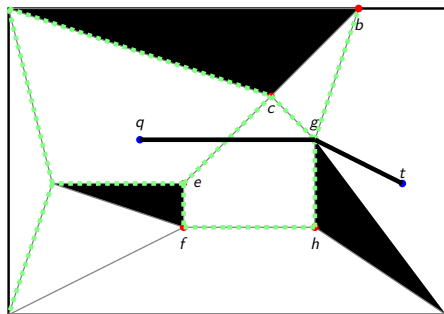
Interval heuristic: drawback

- *interval heuristic* causes redundant expansions
- especially in sparse targets scenario
- e.g.: query is "nearest storage locations where capacity ≥ 100 ".
- motivation: we may need t in h -value to make search smarter



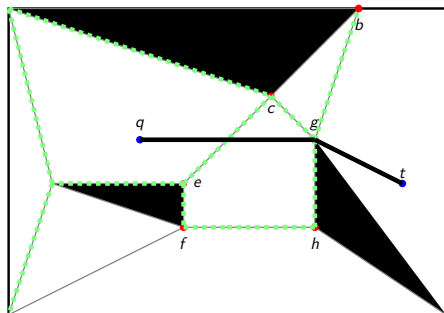
Interval heuristic: drawback

- *interval heuristic* causes redundant expansions
- especially in sparse targets scenario
- e.g.: query is "nearest storage locations where capacity ≥ 100 ".
- motivation: we may need t in h -value to make search smarter



Interval heuristic: drawback

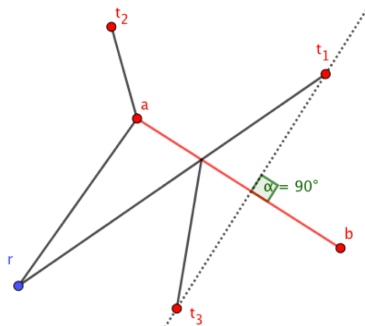
- *interval heuristic* causes redundant expansions
- especially in sparse targets scenario
- e.g.: query is "nearest storage locations where capacity ≥ 100 ".
- motivation: we may need t in h -value to make search smarter



Proposed algorithm 3: target heuristic

Let me introduce the detail of *h-value* in *Polyanya*,
 $h_p(\text{node}, t)$ equals:

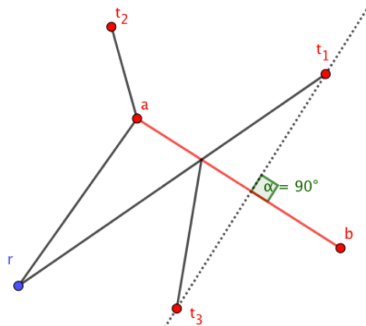
- Case 1: $d_e(r, t_1)$
- Case 2: $d_e(r, a) + d_e(a, t_2)$
- Case 3: when r and t_3 at same side, compute mirror point of t_3 , and go to Case 1 or Case 2



Proposed algorithm 3: target heuristic

Let me introduce the detail of *h-value* in *Polyanya*,
 $h_p(\text{node}, t)$ equals:

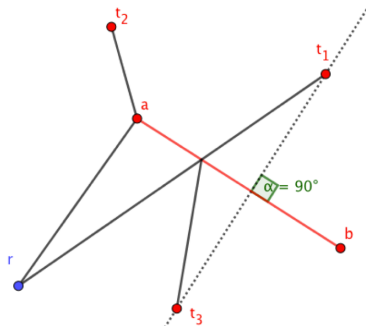
- Case 1: $d_e(r, t_1)$
- Case 2: $d_e(r, a) + d_e(a, t_2)$
- Case 3: when r and t_3 at same side, compute mirror point of t_3 , and go to Case 1 or Case 2



Proposed algorithm 3: target heuristic

Let me introduce the detail of *h-value* in *Polyanya*,
 $h_p(\text{node}, t)$ equals:

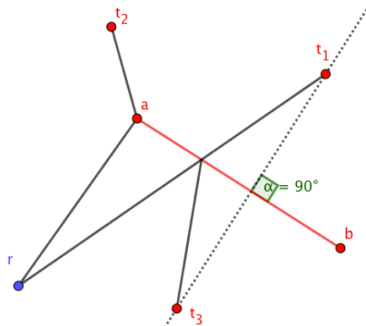
- Case 1: $d_e(r, t_1)$
- Case 2: $d_e(r, a) + d_e(a, t_2)$
- Case 3: when r and t_3 at same side, compute mirror point of t_3 , and go to Case 1 or Case 2



Proposed algorithm 3: target heuristic

Let me introduce the detail of *h-value* in *Polyanya*,
 $h_p(\text{node}, t)$ equals:

- Case 1: $d_e(r, t_1)$
- Case 2: $d_e(r, a) + d_e(a, t_2)$
- Case 3: when r and t_3 at same side, compute mirror point of t_3 , and go to Case 1 or Case 2



Proposed algorithm 3: target heuristic

When there are multiple targets ...



Proposed algorithm 3: target heuristic

When there are multiple targets ...

Definition

closest target of search node is a target t that $h_p(\text{node}, t)$ is minimal.



Proposed algorithm 3: target heuristic

When there are multiple targets ...

Definition

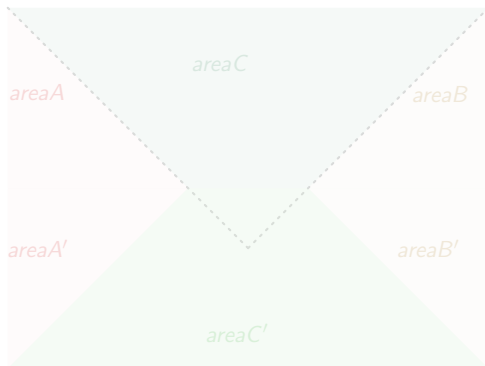
closest target of search node is a target t that $h_p(\text{node}, t)$ is minimal.

How to find the closest target for a search node?



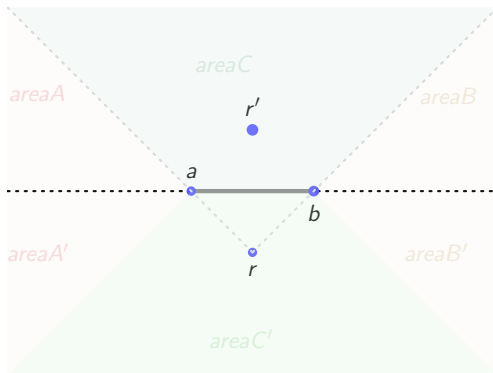
Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.



Proposed algorithm 3: target heuristic

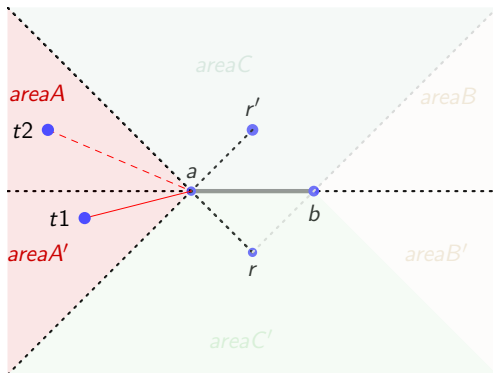
- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.



Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.

■ $NN_e(areaA \cup areaA', a)$ or

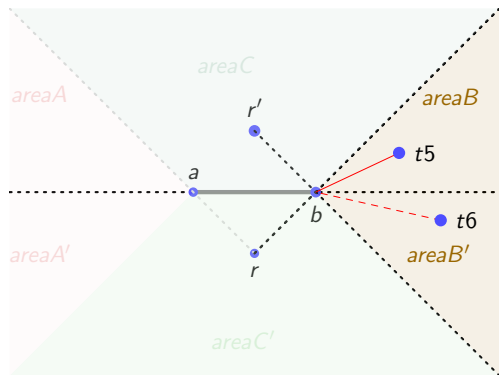


Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.

■ $NN_e(areaA \cup areaA', a)$ or

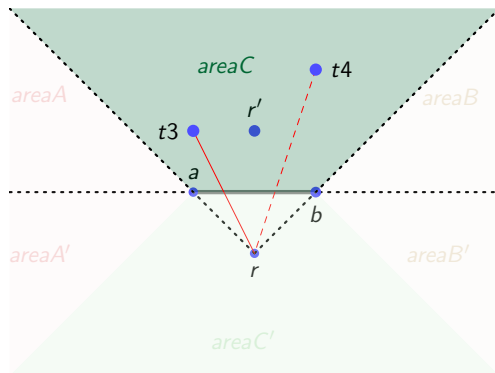
■ $NN_e(areaB \cup areaB', b)$ or



Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.

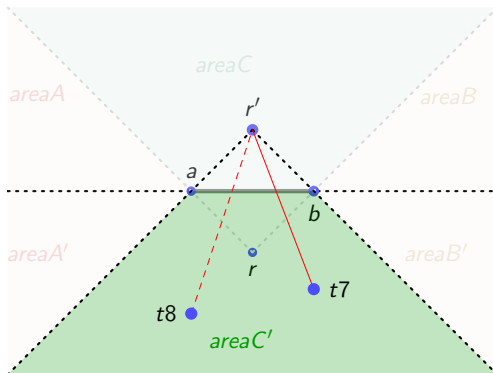
- $NN_e(areaA \cup areaA', a)$ or
- $NN_e(areaB \cup areaB', b)$ or
- $NN_e(areaC, r)$ or



Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.

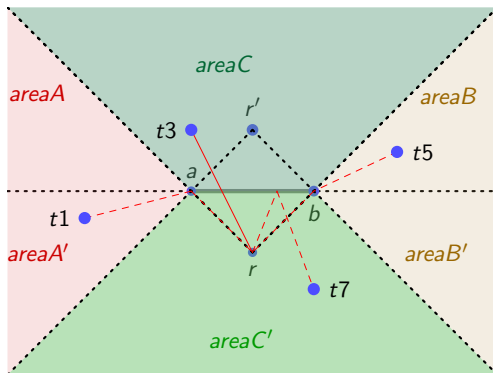
- $NN_e(\text{area}A \cup \text{area}A', a)$ or
- $NN_e(\text{area}B \cup \text{area}B', b)$ or
- $NN_e(\text{area}C, r)$ or
- $NN_e(\text{area}C', r')$



Proposed algorithm 3: target heuristic

- In Case 3, instead of flipping targets, we can flip the r
- Let $NN_e(area, p)$: traditional nearest neighbor of p in $area$.

- $NN_e(areaA \cup areaA', a)$ or
- $NN_e(areaB \cup areaB', b)$ or
- $NN_e(areaC, r)$ or
- $NN_e(areaC', r')$
- Choose the best



Proposed algorithm 3: target heuristic

- For each successor, assign the closest target to it

- Correctness:

Proposed algorithm 3 is correct because it maintains the monotonicity property between the closest target of a search node and its parent node. Suppose node n is the parent of node m . Then, the closest target of node n is closer to node n than the closest target of node m is to node m .



Proposed algorithm 3: target heuristic

- For each successor, assign the closest target to it
- Correctness:

Lemma

Non-decreasing property: Whenever the closest target of a search node changes, the h -value never decrease.



Proposed algorithm 3: target heuristic

- For each successor, assign the closest target to it
- Correctness:

Lemma

Non-decreasing property: Whenever the closest target of a search node changes, the h -value never decrease.



Proposed algorithm 3: target heuristic

- Four *R-tree* queries for each search node is expensive



Proposed algorithm 3: target heuristic

- Four *R-tree* queries for each search node is expensive
- So we are looking for further refinements...



Proposed algorithm 3: target heuristic refinements

- Lazy query



Proposed algorithm 3: target heuristic refinements

■ Lazy query

Definition

In expansion, instead of finding a new target, successors can inherit the closest target from their parent if the *h-value* doesn't change.



Proposed algorithm 3: target heuristic refinements

■ Lazy query

Definition

In expansion, instead of finding a new target, successors can inherit the closest target from their parent if the *h-value* doesn't change.

■ Correctness



Proposed algorithm 3: target heuristic refinements

■ Lazy query

Definition

In expansion, instead of finding a new target, successors can inherit the closest target from their parent if the *h-value* doesn't change.

■ Correctness

Lemma

*In this case, it is impossible to find a target with less *h-value*.*



Proposed algorithm 3: target heuristic refinements

■ Reassignment



Proposed algorithm 3: target heuristic refinements

■ Reassignment

Definition

Once t be retrieved, we must reassign another target to those search nodes who are regarding t as their closest target



Proposed algorithm 3: target heuristic refinements

■ Reassignment

Definition

Once t be retrieved, we must reassign another target to those search nodes who are regarding t as their closest target

■ Lazy reassignment



Proposed algorithm 3: target heuristic refinements

■ Reassignment

Definition

Once t be retrieved, we must reassign another target to those search nodes who are regarding t as their closest target

■ Lazy reassignment

Definition

Instead of exploring the entire open list, we can do reassignment when such search node pop out



Proposed algorithm 3: target heuristic refinements

■ Reassignment

Definition

Once t be retrieved, we must reassign another target to those search nodes who are regarding t as their closest target

■ Lazy reassignment

Definition

Instead of exploring the entire open list, we can do reassignment when such search node pop out

■ Correctness



Proposed algorithm 3: target heuristic refinements

■ Reassignment

Definition

Once t be retrieved, we must reassign another target to those search nodes who are regarding t as their closest target

■ Lazy reassignment

Definition

Instead of exploring the entire open list, we can do reassignment when such search node pop out

■ Correctness

Lemma

Lazy reassignment doesn't change relative expansion order.



Outline

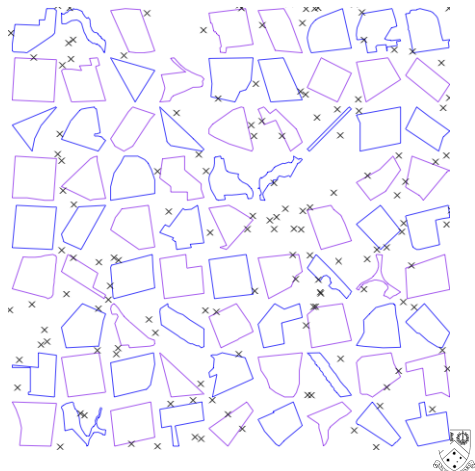
- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research
- 6 Experiments**
- 7 Conclusion and future work
- 8 End



Benchmark Problem

Dataset in *Zhang, EDBT 2004*: no longer available, so we generate new benchmark problems:

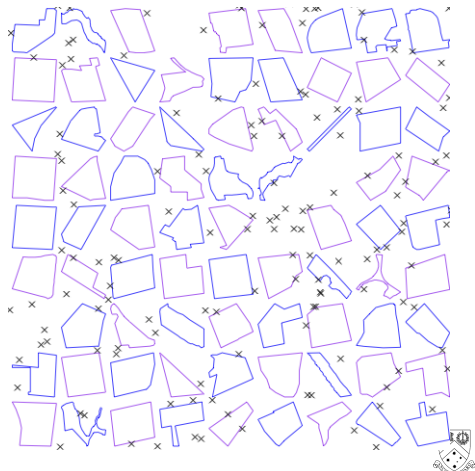
- All parks (≈ 9000) in Australia from *OpenStreetMap*
- Use them as polygonal obstacles



Benchmark Problem

Dataset in *Zhang, EDBT 2004*: no longer available, so we generate new benchmark problems:

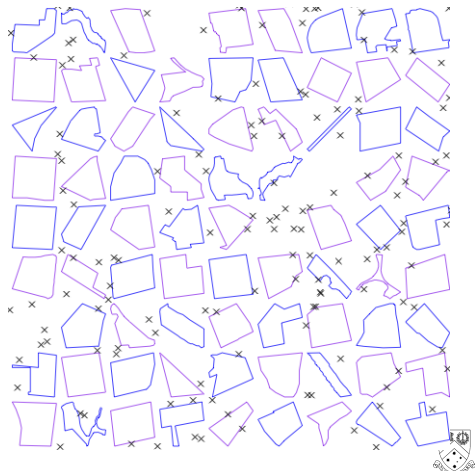
- All parks (≈ 9000) in Australia from *OpenStreetMap*
- Use them as polygonal obstacles



Benchmark Problem

Dataset in *Zhang, EDBT 2004*: no longer available, so we generate new benchmark problems:

- All parks (≈ 9000) in Australia from *OpenStreetMap*
- Use them as polygonal obstacles



Competitors

There are two types of test case:

- Dense targets: $|T| \approx |O|, |O| \approx 9000$
- Sparse targets: $|T| \leq 10, |O| \approx 9000$

In dense targets experiments, we compare between:

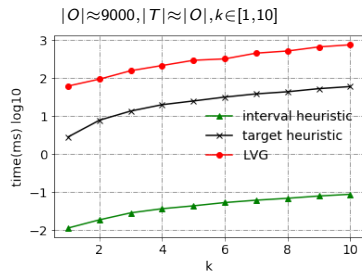
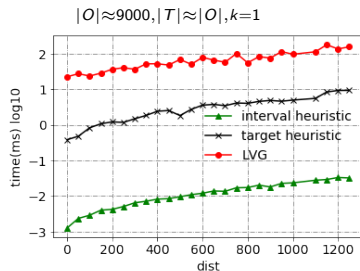
- *LVG* (from *Zhang, EDBT 2004*)
- Interval heuristic
- Target heuristic

In sparse targets experiments, we compare between:

- brute-force Polyanya
- Interval heuristic
- Target heuristic



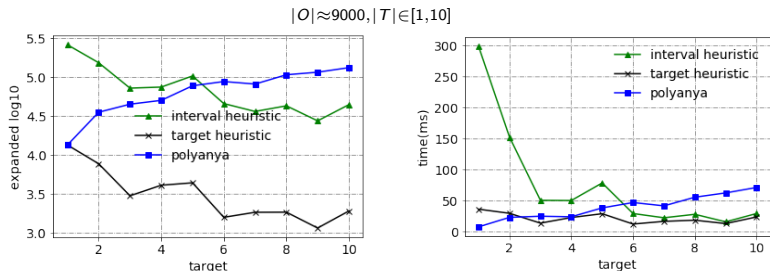
Dense targets



- *Interval heuristic* is three order of magnitude faster than *LVG*, in all aspects.



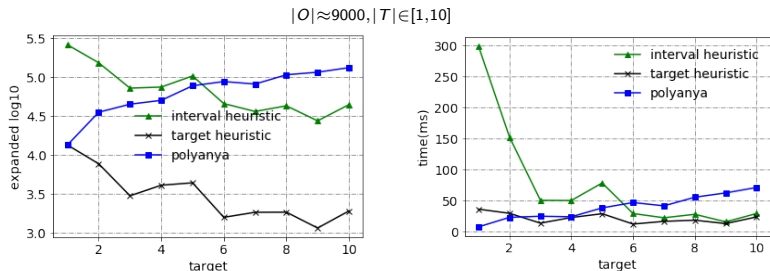
Sparse targets: fix $k = 1$



- *Target heuristic* always has smaller search space. (left)
- It gradually lose such advantage when $|T|$ increase. (right)
- Reason: the costly heuristic function.



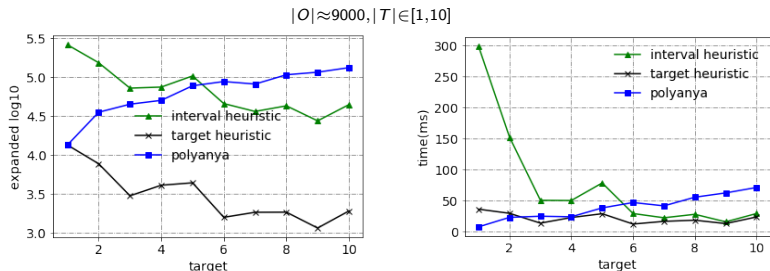
Sparse targets: fix $k = 1$



- *Target heuristic* always has smaller search space. (left)
- It gradually lose such advantage when $|T|$ increase. (right)
- Reason: the costly heuristic function.



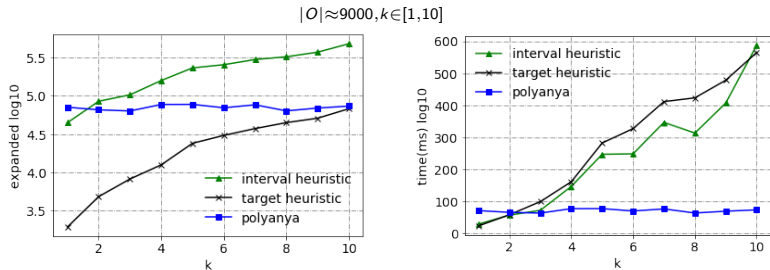
Sparse targets: fix $k = 1$



- *Target heuristic* always has smaller search space. (left)
- It gradually lose such advantage when $|T|$ increase. (right)
- Reason: the costly heuristic function.



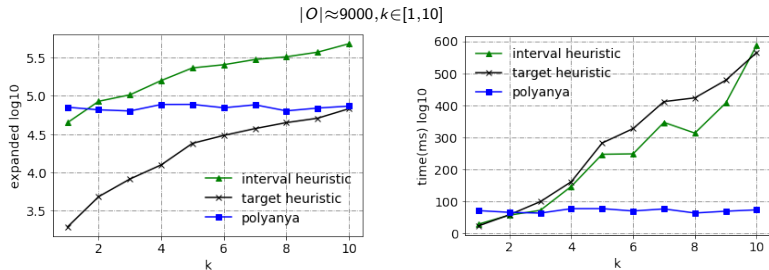
Sparse targets: fix $|T| = 10$



- *Target heuristic* always has small search space. (left)
- It's outperformed by *brute-force Polyanya* when $k \geq 2$. (right)
- Reason: lazy reassignment becomes more frequent.



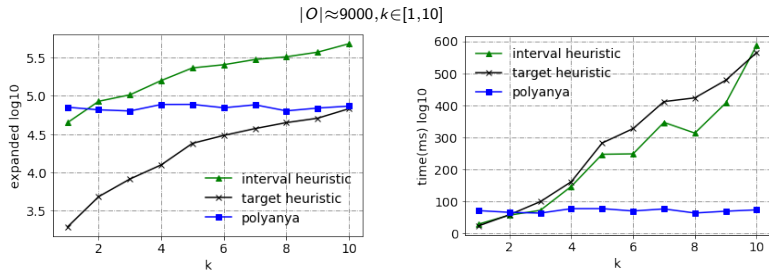
Sparse targets: fix $|T| = 10$



- *Target heuristic* always has small search space. (left)
- It's outperformed by *brute-force Polyanya* when $k \geq 2$. (right)
- Reason: lazy reassignment becomes more frequent.



Sparse targets: fix $|T| = 10$



- *Target heuristic* always has small search space. (left)
- It's outperformed by *brute-force Polyanya* when $k \geq 2$. (right)
- Reason: lazy reassignment becomes more frequent.



Outline

- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research
- 6 Experiments
- 7 Conclusion and future work**
- 8 End



Conclusion

- Proposed algorithms outperform *LVG* in all cases
- *Interval heuristic* works well when targets are many
- *Target heuristic* works well when targets are few and k is also small



Conclusion

- Proposed algorithms outperform *LVG* in all cases
- *Interval heuristic* works well when targets are many
- *Target heuristic* works well when targets are few and k is also small



Conclusion

- Proposed algorithms outperform *LVG* in all cases
- *Interval heuristic* works well when targets are many
- *Target heuristic* works well when targets are few and k is also small



Conclusion

Shizhe Zhao, David Taniar, Daniel Harabor, "Fast k-Nearest Neighbor On A Navigation Mesh", Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS'2018), colocated with IJCAI/ECAI'2018, July 2018 (accepted for publication)



Future works 1: improve other query processing

- Proposed algorithms can be used to speed up other types of spatial query which need to compute obstacle distance, e.g. *Obstacle Reverse Nearest Neighbor*.



Future works 2: improve *target heuristic*

- *Target heuristic* cost $\approx 80\%$ of total run time in *R-tree* query.
- Improve it by combining four queries into one, or using more suitable datastructure.



Future works 2: improve *target heuristic*

- *Target heuristic* cost $\approx 80\%$ of total run time in *R-tree* query.
- Improve it by combining four queries into one, or using more suitable datastructure.



Future works 3: improve *brute-force Polyanya*

- We notice that *brute-force Polyanya* sometimes outperforms other proposed algorithms in sparse scenario.
- Instead of considering every target, maybe a smart pruning strategy can make it work in general scenario.



Future works 3: improve *brute-force Polyanya*

- We notice that *brute-force Polyanya* sometimes outperforms other proposed algorithms in sparse scenario.
- Instead of considering every target, maybe a smart pruning strategy can make it work in general scenario.



Outline

- 1 Introduction
- 2 Related works
- 3 Challenges
- 4 New Framework
- 5 My research
- 6 Experiments
- 7 Conclusion and future work
- 8 End**



End

Q & A



End

Thank you!

