VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

Project Report:
Deep learning and sampling-based path planning for autonomous robots

Lecturer: Lê Hồng Trang
Students: Tạ Trung Tín, Phạm Ngọc Long
January 2025

Contents

1. Problem

1.1. Problem introduction

Path planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination.

Image's source: https://www.theconstruct.ai/

1.1. Problem introduction

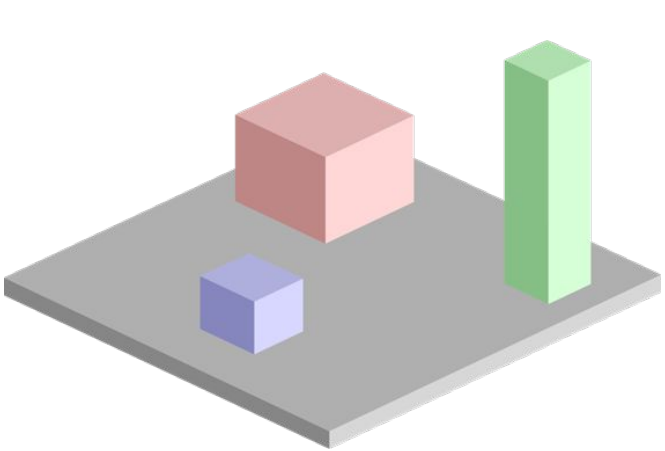
- With the development of robotics and automation technology, solving the path planning problem well will bring great benefits to many fields. This is the motivation for solving this problem.
- Finding the optimal path in real life is not easy, because there are many obstacles or random events that confuse the robot.

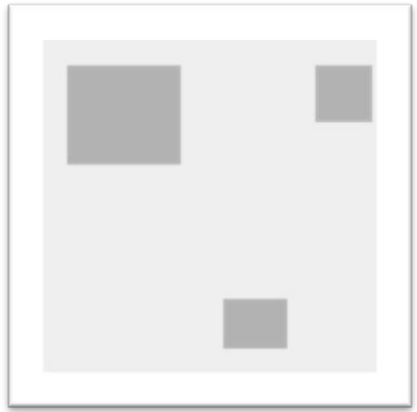
Image's source:
https://valutrack.com/hardware/autonomous-mobile-robots/

Image's source:
https://www.automate.org/robotics/blogs/food-delivery-robots-take-to-the-streets

1.2. Problem definition

- A basic path planning problem is to compute an optimal continuous path that connects a start configuration S and a goal configuration G, while avoiding collision with known obstacles.
- An example for configuration space:



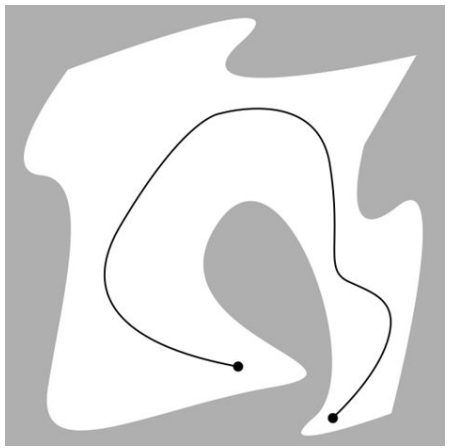


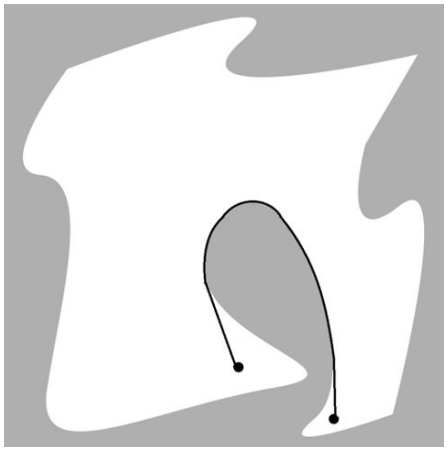
Configuration space of a point-sized robot.
White = C_{free}, gray = C_{obs}

Image's source: https://en.wikipedia.org/wiki/Motion_planning

1.2. Problem definition

- The optimal planning problem is defined as the search for the path that minimize the cost function on that path among all valid path. Cost function is usually the length of the path taken.





Image's source: https://en.wikipedia.org/wiki/Motion_planning

1.2. Problem definition

Formally, let $X \in \mathbb{R}^n$ be the configuration space. Obstacle space $X_{obs} \subsetneq X$ is a space that the robot can not move to. Let $X_{free} = X \setminus X_{obs}$ be the be the free space - the set of configurations that avoids collision with obstacles. Let $X_{start} \in X_{free}$ be the start configuration and $X_{goal} \in X_{free}$ be the goal configuration. Let a collision-free path τ be a continuous mapping in X such that $\tau : [0,1] \to X_{free}$, where $\tau(0) = X_{start}$, $\tau(1) = X_{goal}$. Let T be the set of collision-free paths. The optimal planning problem is then formally defined as the search for the path τ^* that minimize the cost function $c : T \to \mathbb{R}$.

$$\tau^* = \arg\min_{\tau \in T} c(\tau) .$$

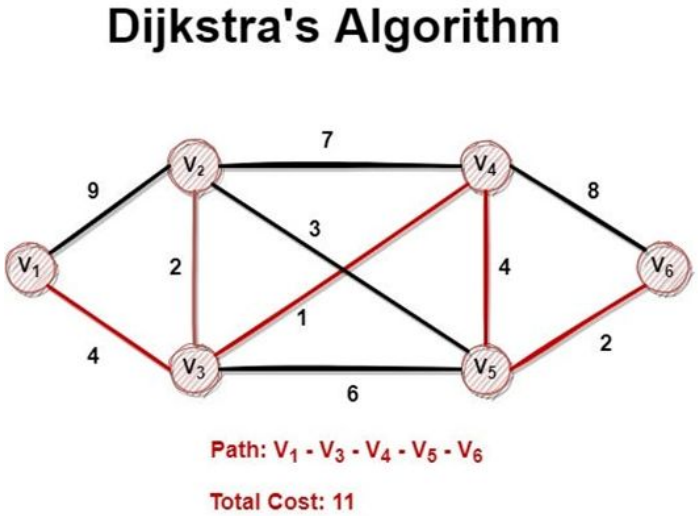
1.3. Problem variants

- According to different degrees of understanding of environmental information, there are two kinds of global path planning, that environmental information is fully known and local path planning where environmental information is partially unknown or completely unknown.
- According to whether the obstacle is moving in the environment, it can be divided into static planning with static environment unchanged and dynamic planning with obstacle movement.

2. Search-based algorithms

2.1. Definition

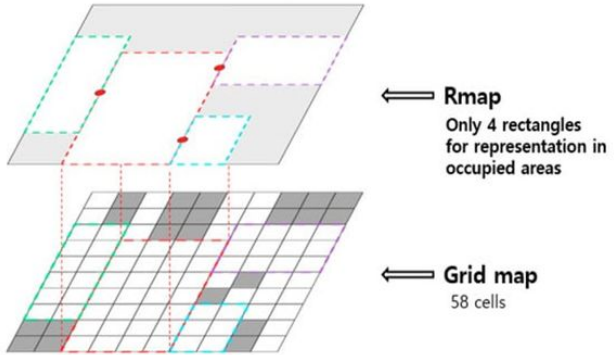
- Search-based, or map-based algorithm is the traditional path planning algorithm based on map (Dijkstra algorithm, A * algorithm, etc.).
- Specifically, the algorithm will process and search on the paths formed by nodes on the graph to find the optimal/ sub-optimal path. The solution of the problem will always be unique for each fixed input.

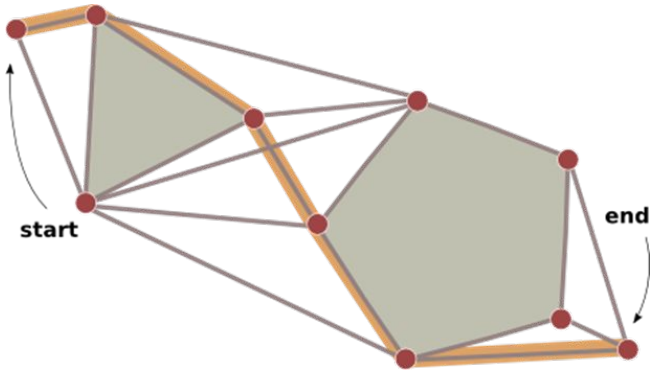


Image's source: Google

2.2. Map representing

- Grids: We can divide the map into R x C grid map, each cell can contain information about that area on a real map (i.e. free cell, occupied cell). Each cell is a node in the graph, two adjacent cells are connected.
- Non-grid graph: The most common is to use a polygonal representation. If the movement cost across large areas is uniform, and if your units can move in straight lines instead of following a grid.





Image's source:
https://www.redblobgames.com/pathfinding/a-star/introduction.html

2.3. Algorithms

- Breadth-first search: We suppose the cost of moving on each edge is equal, this algorithm starts on a start node and keeps expanding till it reaches the goal.
- Dijkstra algorithm: The algorithm traverses the most subnodes one by one through the greedy principle, and then uses the relaxation method to optimize the path selection.

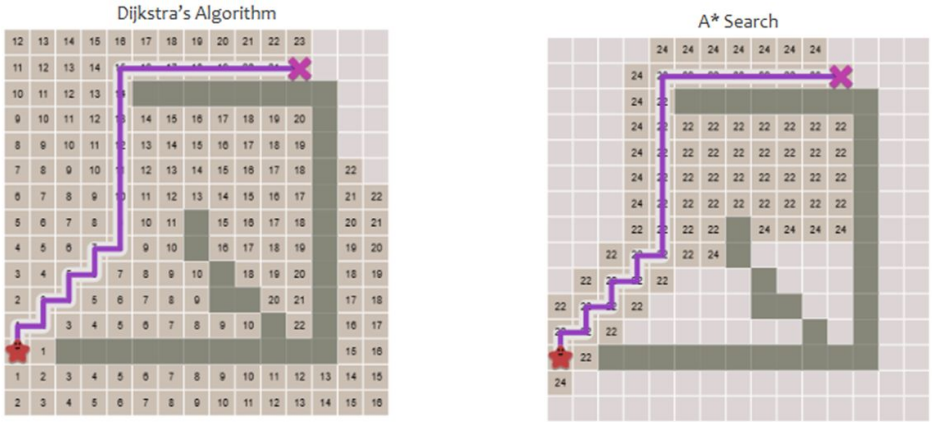
Algorithm 1: Dijkstra's algorithm

Input: Graph $G = (V, E)$
1 $(\forall x \neq s) dist[x] = +\infty$ //Initialize dist[]
2 $dist[s] = 0$
3 $S = \emptyset$
4 $Q = V$ // Keyed by $dist[]$.
5 **while** $Q \neq \emptyset$ **do**
6 $u = extract\_min(Q)$
7 $S = S \cup \{u\}$
8 **foreach** *vertex* $v \in Adj(u)$ **do**
9 $dist[v] = min(dist[v], dist[u] + w(u, v))$
10 //"*Relax*" operation.

2.3. Algorithms

- A * algorithm: A* algorithm is a heuristic search algorithm, as an improvement of Dijkstra algorithm. In the process of Dijkstra algorithm search, heuristic function is added. Heuristic function is a lower boundary for the cost from a given node to the goal.

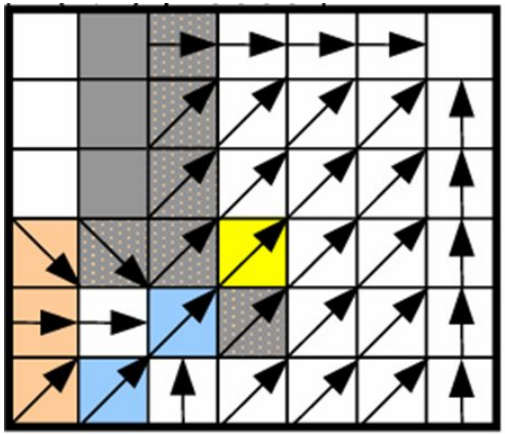
$$dist[v] = \min(dist[v], dist[u] + w(u,v) + heuristic(v, goal))$$

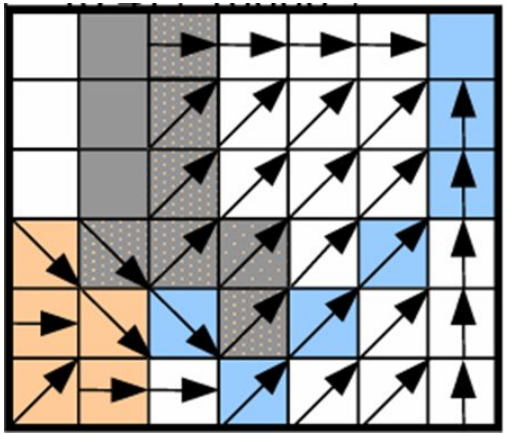


Image's source:
https://www.redblobgames.com/pathfinding/a-star/introduction.html

2.3. Algorithms

- D * algorithm: Stands for “Dynamic A* Search”. D* algorithm is a reverse incremental search algorithm. Reverse means that the algorithm searches gradually from the target point to the starting point. If the obstacle appear, we replan the path from the obstacle.





Image’s source: https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf

2.3. Algorithms

- LPA* algorithm: Life Planning A* algorithm is an incremental version of A*, which can adapt to changes in the graph without recalculating the entire graph, by updating the g-values (distance from start) from the previous search during the current search to correct them when necessary. When edge costs change, LPA* outperforms A* (assuming the latter is run from scratch) as only a fraction of nodes need to be expanded again.
- D * Lite algorithm: D* Lite is an incremental heuristic search algorithm which is based on LPA*. It has completely obsoleted D*.

2.3. Algorithms

Table 1. Comparison of performance and application of traditional algorithms

	Heuristic	Incremental	Search direction	Scope of application
Dijkstra	F	F	positive	Global information is known, static planning
A*	T	F	positive	Global information is known, static planning
D*	F	T	reverse	Some information is known, Dynamic programming
LPA*	F	T	positive	Part of the information is known, assuming the rest of the free path, dynamic programming
D*lite	T	T	reverse	Part of the information is known, assuming the rest of the free path, dynamic programming

2.4. Pros and cons of Search-based algorithm

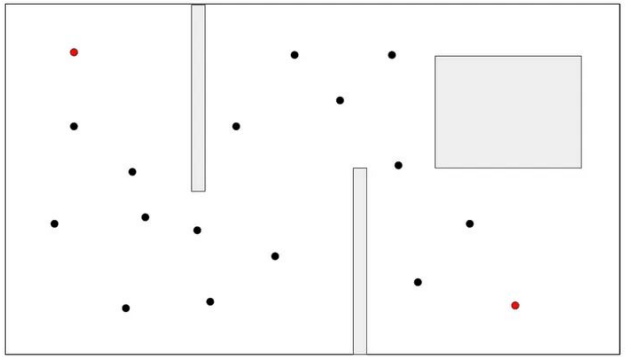
- Pros:
  - Optimal in terms of the path cost.
  - Fast search speed for small graph.
- Cons:
  - Performance depend on the map representing.
  - Takes up a lot of memory.
  - Complexity is exponential in the dimension of the robot's Configuration space.

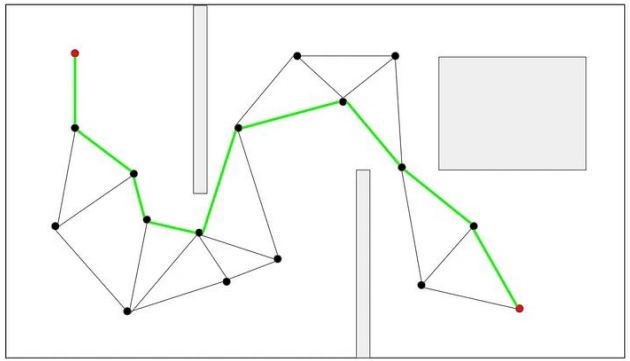
→ Sampling based algorithm.

3. Sampling-based path planning

3.1. Concepts

- Sampling-based planning: Search for collision-free path only by sampling points (randomly or strategically)
  → don't require exploring the full configuration space.
- Sampling method:
  - Unbiased method (uniform sampling): uniformly sampling over the whole configuration space.
  - Biased method: bias sampling toward interesting regions of the configuration space.



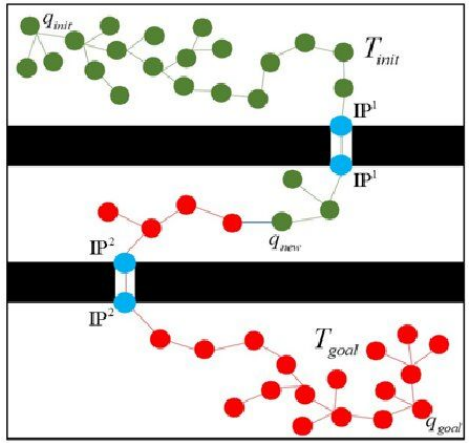


Source: What is Sampling-Based Motion Planning? - https://medium.com/@oelmofty/what-is-sampling-based-motion-planning-a693a534a2a8

3. Sampling-based path planning

3.1. Concepts

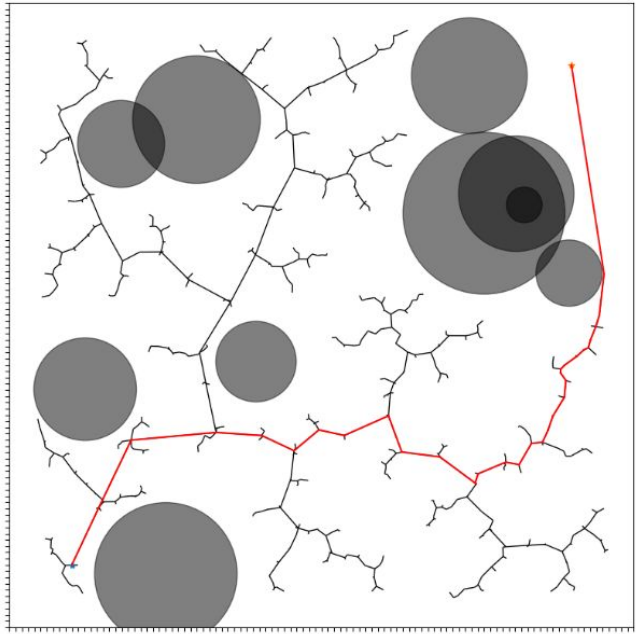
- Advantages:
  - *Reduce computational cost and storage space* → scalable.
  - *Probabilistic Completeness:* Given enough time they are guaranteed to find a path if one exists.
  - *Asymptotic Optimality* (RRT*, PRM*, …).
- Disadvantages:
  - *Dependence on Randomness:* not suitable for environments with dense obstacles or narrow passage.
  - *Suboptimal and unsmooth path:* prioritize finding a feasible path over optimization. RRT* address this but may require more computational time and resources.



Sampling-based path planning in case of narrow passage

3.2 Rapidly-exploring random trees [1]

- RRT is a *sampling-based* path planning algorithm.
- It incrementally builds a tree through random sampling and infer the path toward the goal using that tree.
- Features:
  - *Scalable*.
  - *Asymptotic Completeness*.
  - Usually create *suboptimal* and *non-smooth* path.
  - *May be slow convergence.*



Source: ChengeYang/RRT: Basic RRT algorithm implementations.

[1] LaValle, Steven M. (October 1998). "Rapidly-exploring random trees: A new tool for path planning" (PDF). Technical Report (TR 98–11). Computer Science Department, Iowa State University.

3.2 Rapidly-exploring random trees [1]

Pseudocode:

Algorithm 6 RRT Algorithm

```
Input:
    start_node ← The start node
    goal_node ← The goal node
    K ← The number of nodes in RRT
    δ ← Incremental distance
    D ← The planning domain
    O ← The set of obstacles within the planning domain
Algorithm:
Initialize tree T with start_node
for i ← 0 to K − 1 do
    random_node ← sample_from(D)
    nearest_node ← pick_nearest_vertex(random_node, T)
    new_node ← fit_incremental_distance(nearest_node, random_node, δ)
    if not_in_obstacle(O, new_node) then
        edge = create_edge(nearest_node, new_node)
        if not_in_obstacle(O, edge) then
            add_to_tree(T, edge)
        end if
    end if
end for
```

[1] LaValle, Steven M. (October 1998). "Rapidly-exploring random trees: A new tool for path planning" (PDF). Technical Report (TR 98–11). Computer Science Department, Iowa State University.

3.3 RRT* [2]

- RRT* is an *extension* of RRT that improves the algorithm by incorporating an *optimization phase*.
- Main *differences* between RRT and RRT*:
  - RRT* calculates the total cost to reach a new node via its neighbors and *chooses the minimum cost path* to connect.
  - *Rewire* the tree if possible to connect the lower-cost paths.
- Key features:
  - Guarantees *asymptotic optimality*.
  - *May be slower than RRT* because of additional computation in the optimization phase.

[2] Karaman, S. and Frazzoli, E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7):846–894, 2011a. DOI: 10. 1177/0278364911406761.

3.3 RRT* [2]

Pseudocode:

Algorithm 7 RRT* Algorithm

Input:
  $start\_node \leftarrow$ The start node
  $goal\_node \leftarrow$ The goal node
  $K \leftarrow$ The number of nodes in RRT
  $\Delta \leftarrow$ Incremental distance
  $D \leftarrow$ The planning domain
  $O \leftarrow$ Obstacles within the planning domain
  $search\_radius \leftarrow$ Search radius
Algorithm:
Initialize tree T with $start\_node$
for $i \leftarrow 0$ **to** $K - 1$ **do**
  $random\_node \leftarrow pick\_random\_node(D)$
  $new\_node \leftarrow fit\_incremental\_distance(nearest\_node, random\_node, \Delta)$
  if $not\_in\_obstacle(O, new\_node)$ **then**
    $near\_nodes \leftarrow find\_near\_nodes(T, new\_node, search\_radius)$

[2] Karaman, S. and Frazzoli, E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7):846–894, 2011a. DOI: 10. 1177/0278364911406761.

3.3 RRT* [2]

```
        for node in near_nodes do
            curr_cost = cost(new_node, node) + cost(node)
            if curr_cost < min_cost then
                edge = create_edge(new_node, node)
                if not_in_obstacle(O, edge) then
                    min_cost = curr_cost
                    link = edge
                end if
            end if
        end for
        add_to_tree(T, link)
        for node in near_nodes do
            if min_cost + cost(new_node, node) < cost(node) then
                edge = create_edge(new_node, node)
                if not_in_obstacle(O, edge) then
                    add_to_tree(T, edge)
                    parent(node) = new_node
                end if
            end if
        end for
        end if
    end for
```

[2] Karaman, S. and Frazzoli, E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7):846–894, 2011a. DOI: 10. 1177/0278364911406761.

3.4 Informed RRT* [3]

- Motivation: RRT and RRT* sample the entire space → (may) redundantly explore
- Informed RRT* (IRRT*) is an enhancement of RRT* that biases sampling within an *"informed" ellipsoidal* region.
  - *Reduce the number of unnecessary samples.*
  - *Improve the convergence rate* towards the optimal solution.
  - *Incrementally improve* the path and the "informed" region.

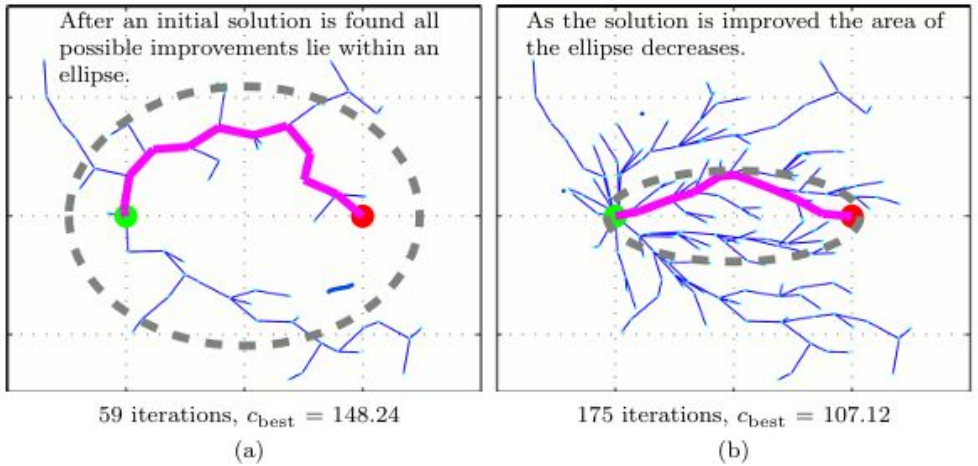
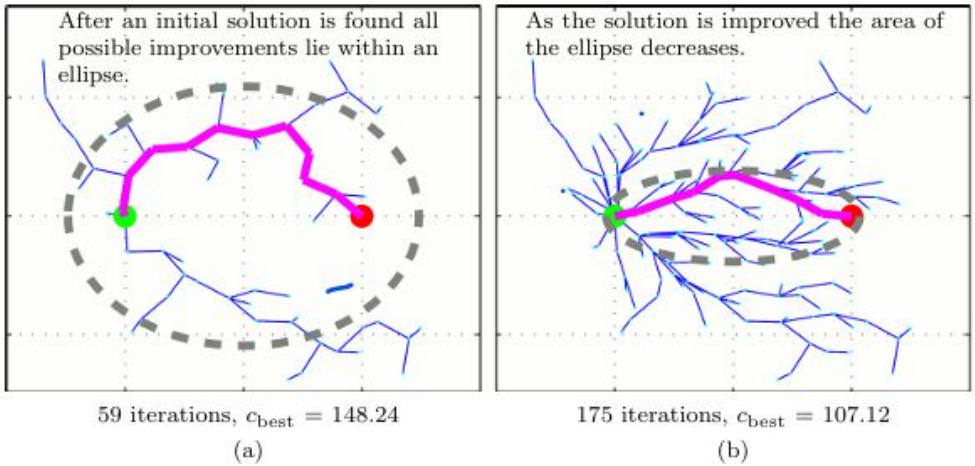


Illustration of IRRT*. [3]

[3] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," doi: 10.1109/IROS.2014.6942976.

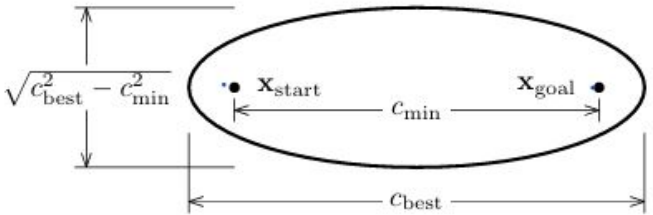
3.4 Informed RRT* [3]

- Informed RRT* includes **two** phases:
  - At first, IRRT* behaves like RRT*.
  - Once the initial solution is found, IRRT* focuses the search on the "ellipsoidal informed subset".



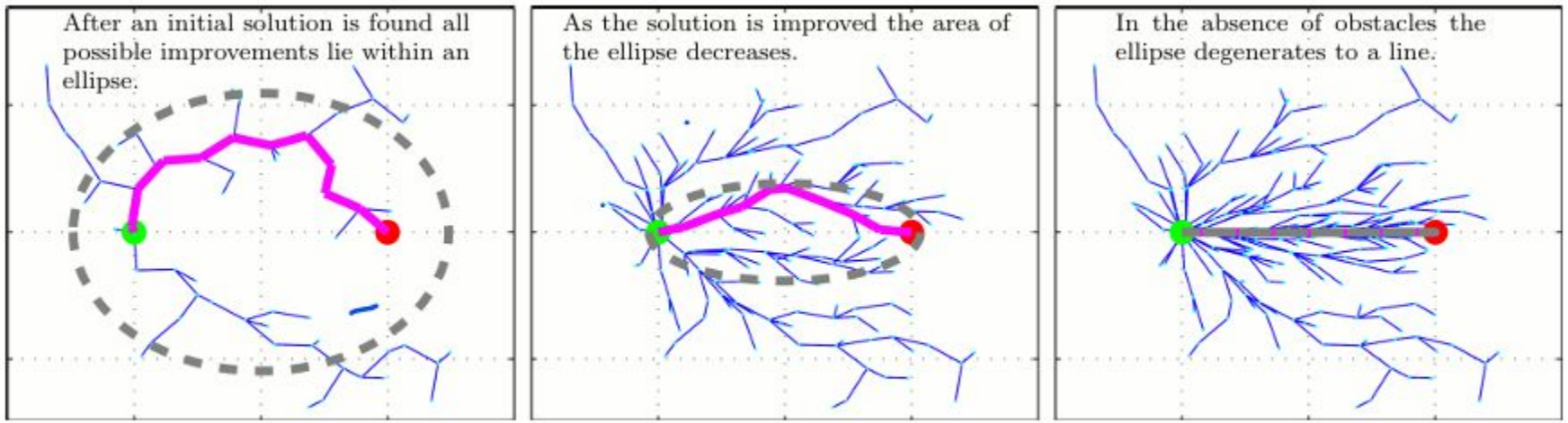
- **Ellipsoidal informed subset** is determined by:
  - Current best solution cost (c_best)
  - Theoretical minimum cost (c_min)

$$X_f = \{x \in X \mid \|x_{\text{start}} - x\|_2 + \|x - x_{\text{goal}}\|_2 \leq c_{\text{best}}\}$$



[3] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," doi: 10.1109/IROS.2014.6942976.

3.4 Informed RRT* [3]



Source: [3]

[3] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," doi: 10.1109/IROS.2014.6942976.

3.4 Informed RRT* [3]

- Advantages:
  - Faster Convergence
  - More effective than RRT* without additional user-tuned parameters
- Disadvantages:
  - Reliance on having an initial solution → still not suitable for narrow passage, …

[3] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," doi: 10.1109/IROS.2014.6942976.

4. Learning-based path planning
4.1. Neural RRT* [4]

- Motivation:
  - IRRT* relies on the initial path.
  - The quality of the *initial* path is *not guaranteed* and it takes *much time* to converge to the optimal path.

→ Neural RRT*: Propose CNN model that can predict the *probability distribution o*f the optimal path on the map → bias the sampling process.

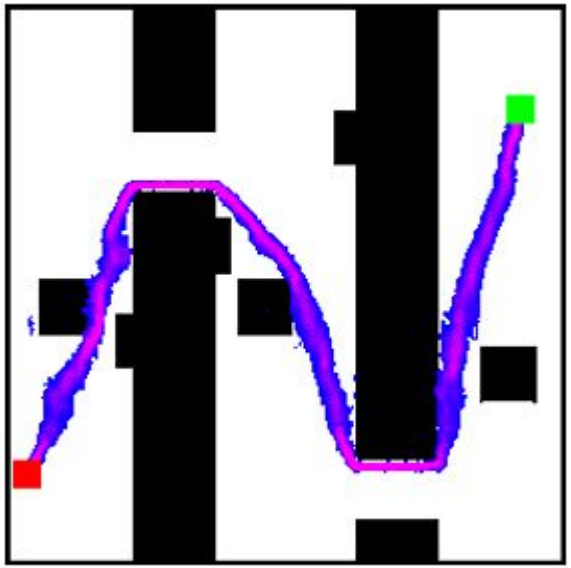


Figure: Predicted probability distribution of the optimal path

[4] Wang, Jiankun & Chi, Gloria & Li, Chenming & Wang, Chaoqun & Meng, Max. (2020). Neural RRT*: Learning-Based Optimal Path Planning. IEEE Transactions on Automation Science and Engineering. PP. 1-11. 10.1109/TASE.2020.2976560.

4. Learning-based path planning
4.1. Neural RRT* [4]

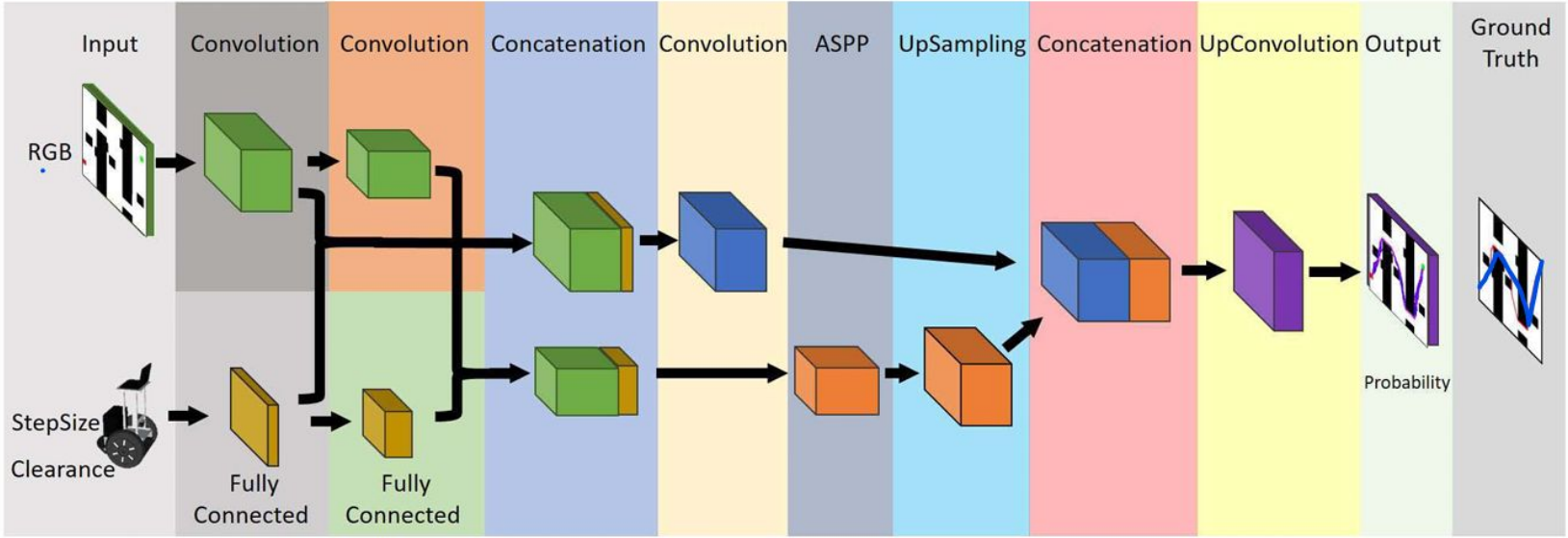


Fig. 3. Illustration of the proposed CNN model.

[4] Wang, Jiankun & Chi, Gloria & Li, Chenming & Wang, Chaoqun & Meng, Max. (2020). Neural RRT*: Learning-Based Optimal Path Planning. IEEE Transactions on Automation Science and Engineering. PP. 1-11. 10.1109/TASE.2020.2976560.

4. Learning-based path planning
4.1. Neural RRT* [4]

- Input of the CNN model:
  - 2-D images to represent the 2-D state space (0: free, 1: obstacles, 2: start, 3: goal)
  - Step size: maximum step length of the robot
  - Clearance: minimum required distance of the path from obstacles
- Output: **2D probability map**, where each pixel shows the **likelihood** of being on the optimal path.
- Ground truth: is generated using the A* algorithm to create optimal paths on the map.
- Cross-entropy loss function:

$$L = \sum \mathrm{ce}(\mathcal{O}, \mathcal{G}) + \lambda \mathrm{ce}(\mathcal{I}, \mathcal{R})$$

- I: input map
- O: predicted probability image
- G: ground truth
- R: Reconstructed map

[4] Wang, Jiankun & Chi, Gloria & Li, Chenming & Wang, Chaoqun & Meng, Max. (2020). Neural RRT*: Learning-Based Optimal Path Planning. IEEE Transactions on Automation Science and Engineering. PP. 1-11. 10.1109/TASE.2020.2976560.

4. Learning-based path planning
4.1. Neural RRT* [4]

Algorithm 10 NRRT*

Input: $x_{\text{init}}, G(x_{\text{goal}}), Map, S, C$
Output: T
$V \leftarrow x_{\text{init}}, E \leftarrow \emptyset, T \leftarrow (V, E)$
$\mathcal{O} \leftarrow \text{NeuralModel}(Map, S, C)$
for $i = 1 \ldots N$ **do**
  if $\text{Rand}() > 0.5$ **then**
    $x_{\text{rand}} \leftarrow \text{NonuniformSample}(\mathcal{O})$
  else
    $x_{\text{rand}} \leftarrow \text{UniformSample}()$
  end if
  $x_{\text{nearest}} \leftarrow \text{Nearest}(T, x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
  if $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $T \leftarrow \text{Extend}(T, x_{\text{new}})$
    $\text{Rewire}(T)$
    if $x_{\text{new}} \in G(x_{\text{goal}})$ **then**
      return T
    end if
  end if
end for
return failure

[4] Wang, Jiankun & Chi, Gloria & Li, Chenming & Wang, Chaoqun & Meng, Max. (2020). Neural RRT*: Learning-Based Optimal Path Planning. IEEE Transactions on Automation Science and Engineering. PP. 1-11. 10.1109/TASE.2020.2976560.

4. Learning-based path planning
4.1. Neural RRT* [4]

- Experiment results with different maps, clearance and step size:
  - Faster convergence
  - The initial path is closer to the optimal path
  - Rely on the quality of the training dataset.

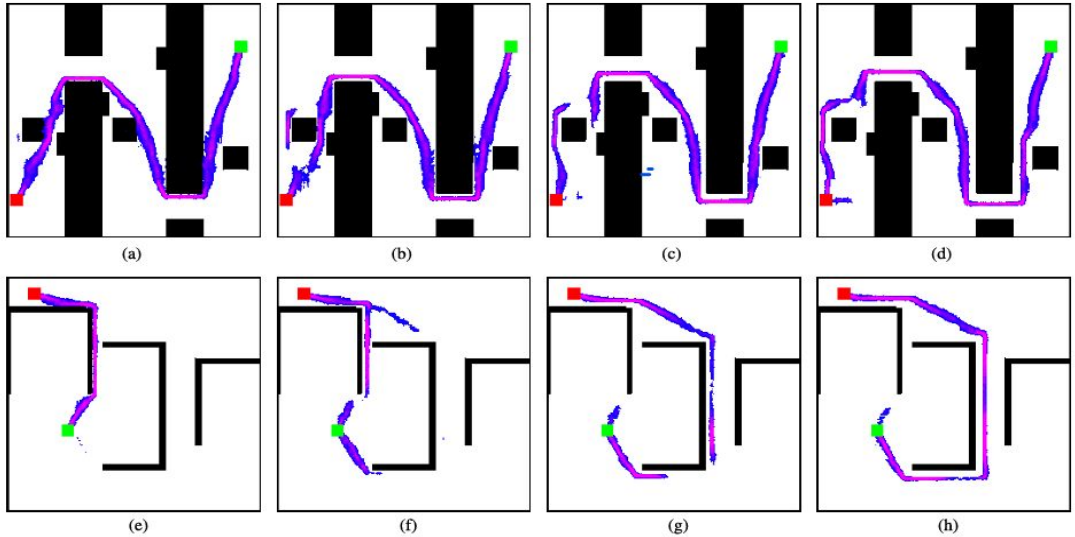


Figure: Predicted sampling distribution under different clearance settings.

[4] Wang, Jiankun & Chi, Gloria & Li, Chenming & Wang, Chaoqun & Meng, Max. (2020). Neural RRT*: Learning-Based Optimal Path Planning. IEEE Transactions on Automation Science and Engineering. PP. 1-11. 10.1109/TASE.2020.2976560.

4.2 Neural Informed RRT* [6]

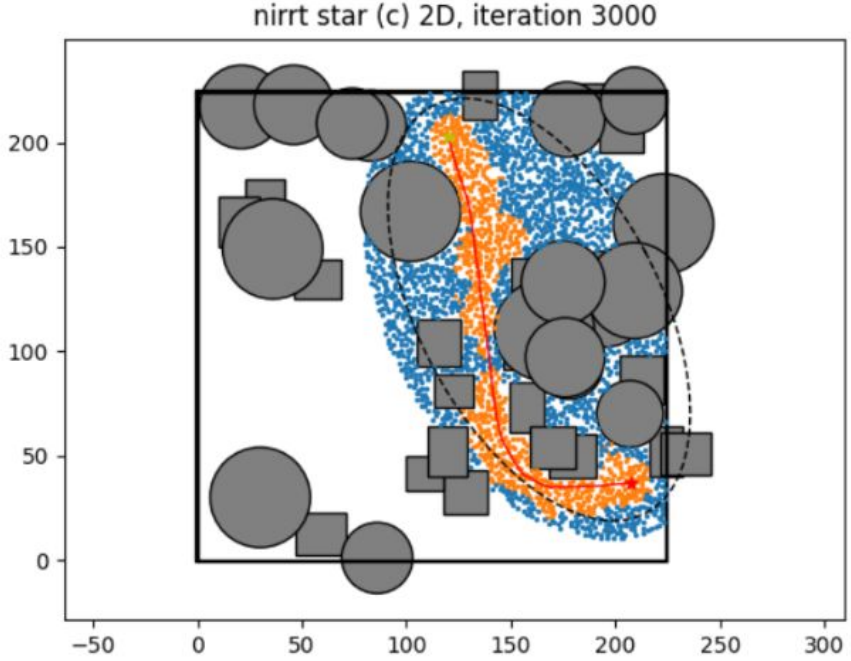
Motivation:

- Learning-based methods encode whole state space to generate guidance states without iterative improvement.
- Learning-based methods do not consider connectivity of the guidance state set, which severely affects the convergence rate in complex planning problems.
- Rule-based informed sampling does not favor topologically critical states in the ellipsoidal subset.

4.2 Neural Informed RRT* [6]

Main contribution:

- Use a Point-based Network (PointNet++) to directly take free states as point cloud input to generate multiple guidance states in one run.
- Present Neural Informed RRT*, by introducing Neural Focus to integrate Point-based Network and Informed RRT*.
- Propose Neural Connect to address the connectivity issue of inferred guidance state set.


nirrt star (c) 2D, iteration 3000
200
150
100
50
0
−50
0
50
100
150
200
250
300

4.2 Neural Informed RRT* [6]

Neural Informed RRT*: (red part is NIRRT* contribution)

Algorithm 1 Neural Informed RRT*

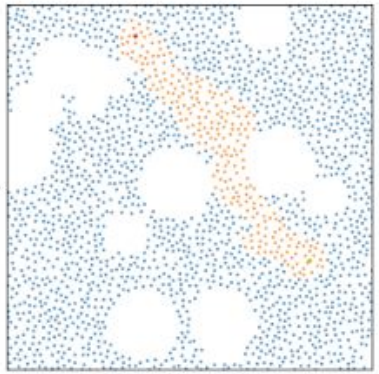
$V \leftarrow \{x_{\text{start}}\}; E \leftarrow \varnothing;$
$X_{\text{soln}} \leftarrow \varnothing;$
$c^0_{\text{best}} \leftarrow \infty;$
$c_{\text{update}} \leftarrow c^0_{\text{best}};$
$X_{\text{guide}} \leftarrow \texttt{PointNetGuide}(x_{\text{start}}, x_{\text{goal}}, c^0_{\text{best}}, X_{\text{free}});$
for $i = 1$ to n **do**
  $c^i_{\text{best}} \leftarrow \min_{x_{\text{soln}} \in X_{\text{soln}}} \{\texttt{Cost}(x_{\text{soln}})\};$
  $x_{\text{rand}}, X_{\text{guide}}, c_{\text{update}} \leftarrow \texttt{PointNetGuidedSampling}(X_{\text{guide}}, x_{\text{start}}, x_{\text{goal}}, c_{\text{update}}, c^i_{\text{best}}, X_{\text{free}});$
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}});$
  if $\texttt{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, r_{\text{RRT*}});$
    $V \leftarrow V \cup \{x_{\text{new}}\};$
    $x_{\text{min}} \leftarrow x_{\text{nearest}};$
    $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$
    for all $x_{\text{near}} \in X_{\text{near}}$ **do**
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}};$
        $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}));$
      end if
    end for
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$
    for all $x_{\text{near}} \in X_{\text{near}}$ **do**
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then**
        $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$
        $E \leftarrow (E \backslash \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\};$
      end if
    end for
    if $\texttt{InGoalRegion}(x_{\text{new}})$ **then**
      $X_{\text{soln}} \leftarrow X_{\text{soln}} \cup \{x_{\text{new}}\};$
    end if
  end if
end for
return G=(V, E);

4.2 Neural Informed RRT* [6]

PointNetGuide use PointNet++ as Point-based network.

What does PointNetGuide do?

→ It takes the starting state, goal state, the length of current best path, free state and output the set of guidance states (X_guide). I.e. state that should have higher probability to be chosen when doing sampling.



(d) guidance state inference

→ PointNet++ takes the point cloud and returns a probability p of each point, then we choose points that have p > 0,5 as guidance points.

→ PointNetGuide also use PointCloudSampling to create a sets of point to feed into PointNet++, and also apply the mechanism called Neural Connect to ensure the connectivity.

4.2 Neural Informed RRT* [6]

PointNetGuidedSampling: Updates X_guide if needed and returns an sampling point.

- When the current best path cost is less than the path cost improvement ratio $\alpha \leq 1$, then we use PointNetGuide to update the X_guide.
- There is a 50% chance that it will uniformly random a point in the ellipsoid. And a 50% chance that it'll uniformly random a point from X_guide.

→ Point in X_guide has higher probability to be chosen.

4.2 Neural Informed RRT* [6]

PointNetGuide:

Algorithm 3 $\texttt{PointNetGuide}(x_{\text{start}}, x_{\text{goal}}, c_{\text{curr}}, X_{\text{free}})$

$x^1_{\text{start}} \leftarrow x_{\text{start}}$;
$x^1_{\text{goal}} \leftarrow x_{\text{goal}}$;
$X_{\text{guide}} \leftarrow \varnothing$;
if $c_{\text{curr}} < \infty$ **then**
    $X_{\text{focus}} \leftarrow \texttt{InformedSubset}(x_{\text{start}}, x_{\text{goal}}, c_{\text{curr}})$;
    $X_{\text{input}} \leftarrow \texttt{PointCloudSampling}(X_{\text{focus}} \cap X_{\text{free}})$;
else
    $X_{\text{input}} \leftarrow \texttt{PointCloudSampling}(X_{\text{free}})$;
end if
for $j = 1$ to n_{guide} **do**
    $\bar{X}_{\text{input}} \leftarrow \texttt{AddOneHotFeatures}(X_{\text{input}}, x^j_{\text{start}}, x^j_{\text{goal}})$;
    $\bar{X}_{\text{input}} \leftarrow \texttt{NormalizeCoordinates}(\bar{X}_{\text{input}})$;
    $X_{\text{guide}} \leftarrow X_{\text{guide}} \cup \text{PointBasedNetwork}(\bar{X}_{\text{input}})$;
    connectivity, $x^{j+1}_{\text{start}} \leftarrow \texttt{BFS}(X_{\text{guide}}, x_{\text{start}}, x_{\text{goal}}, \eta)$;
    if connectivity **then**
        return X_{guide};
    end if
    connectivity, $x^{j+1}_{\text{goal}} \leftarrow \texttt{BFS}(X_{\text{guide}}, x_{\text{goal}}, x_{\text{start}}, \eta)$;
    if connectivity **then**
        return X_{guide};
    end if
end for
return X_{guide};

4.2 Neural Informed RRT* [6]

PointNetGuidedSampling

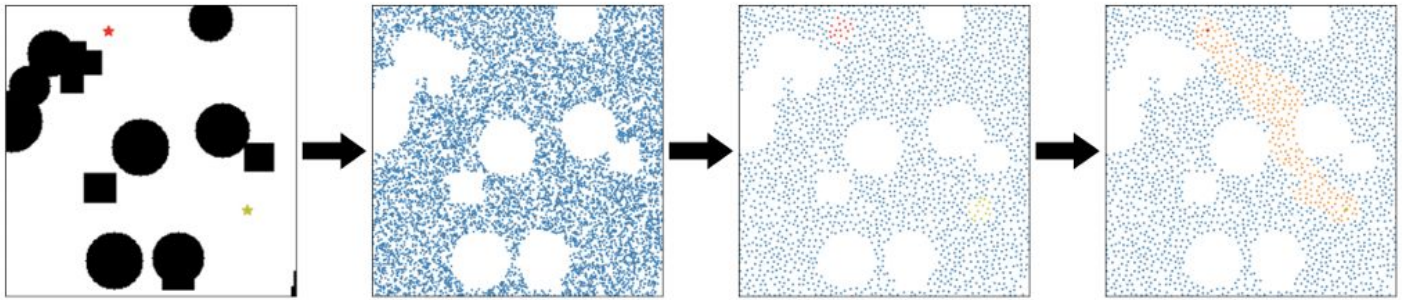
Algorithm 2 PointNetGuidedSampling($X_{\text{guide}}, x_{\text{start}}, x_{\text{goal}}, c_{\text{update}}, c_{\text{curr}}, X_{\text{free}}$)

if $c_{\text{curr}} < \alpha\, c_{\text{update}}$ **then**
    $X_{\text{guide}} \leftarrow$ PointNetGuide($x_{\text{start}}, x_{\text{goal}}, c_{\text{curr}}, X_{\text{free}}$);
    $c_{\text{update}} \leftarrow c_{\text{curr}}$;
end if
if Rand() < 0.5 **then**
    if $c_{\text{curr}} < \infty$ **then**
        $x_{\text{rand}} \leftarrow$ InformedSampling($x_{\text{start}}, x_{\text{goal}}, c_{\text{curr}}$);
    else
        $x_{\text{rand}} \leftarrow$ UniformSampling(X_{free});
    end if
else
    $x_{\text{rand}} \leftarrow$ UniformSampling(X_{guide});
end if
return $x_{\text{rand}}, X_{\text{guide}}, c_{\text{update}}$;

4.2 Neural Informed RRT* [6]

PointNetGuide: *Point-based Network* (NIRRT* use PointNet++):

- Sampling point cloud as input for the network: We oversample points uniformly from X_free. And perform minimum distance downsampling to obtain the point cloud with even distribution. Create one-hot vector for each point, indicating whether the point is within radius η of x_start or x_goal, concatenate the one-hot vectors with normalized point coordinates.



4.2 Neural Informed RRT* [6]

PointNetGuide: *Point-based Network* (NIRRT* use PointNet++):

$$\{p_1, p_2, \ldots, p_N\} = f(\bar{X}_{\text{input}}), \; X_{\text{guide}} = \{x_i | p_i > 0.5\}.$$

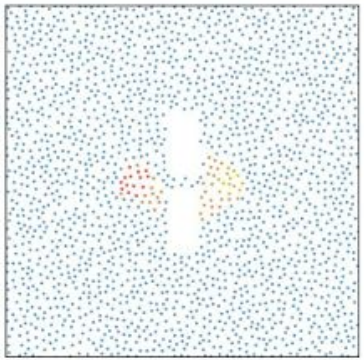
- Prepare data to train the network: 4,000 2D random worlds as the training dataset.
  - Run A* to generate the pixel-wise optimal path.
  - Generate a point cloud of number N = 2048.
  - Generate guidance state labels by checking whether each point is around any point of the pixel-wise optimal path in radius of η.

4.2 Neural Informed RRT* [6]

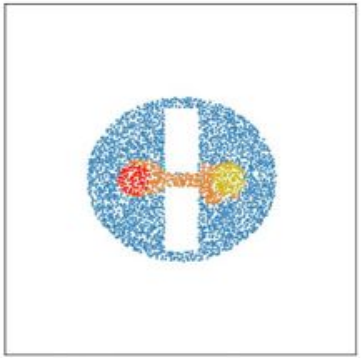
PointNetGuide: *Neural focus*: Neural Focus is to constrain the point cloud input to the point-based network inside the X_focus.

$$X_{\text{focus}} = \{x \in X \,|\, ||x - x_{\text{start}}||_2 + ||x - x_{\text{goal}}||_2 \leq c_{\text{curr}}\}$$

Since we normalize point coordinates when processing point cloud inputs, the trained point-based network can handle point clouds sampled from domains at different scales.



(a) unfocused inference



(b) focused inference

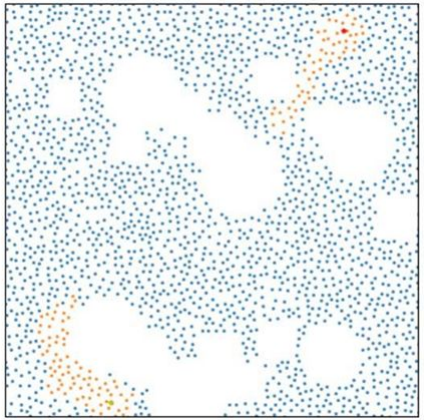
4.2 Neural Informed RRT* [6]

PointNetGuide: *Neural Connect*:

When the distance between x_start and x_goal gets longer, the guidance state set X_guide is more likely to be separated into disconnected “blobs”.

To address this, we run Breadth First Search (BFS) from x_start to x_goal through the guidance states in X_guide. The neighbor radius of BFS is set as η, and no collision check is performed.

After BFS is finished, if the connectivity is still not confirmed.

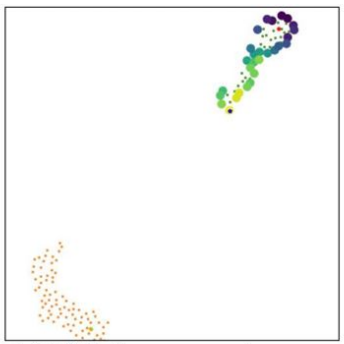


4.2 Neural Informed RRT* [6]

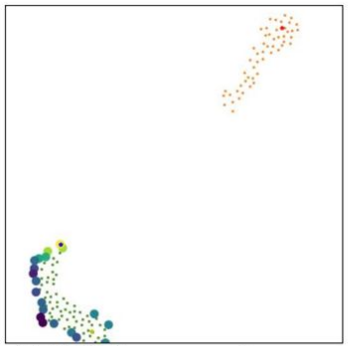
PointNetGuide: *Neural Connect*:

After BFS is finished, if the connectivity is still not confirmed, we find the boundary points X_bound (All points that is close to the visited guidance points). Choose a point from X_bound that is heuristically the furthest from x_start and to reach x_goal with minimum total heuristic cost. Update x_start with this point.

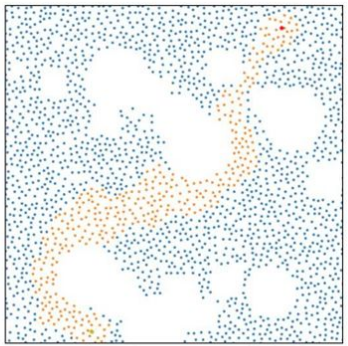
Then we continue to BFS, but with the start of BFS as x_goal, and the goal of BFS as x_start. Doing the same to update x_goal. Then we use PointNet++ to infer new guidance states with new x_start and x_goal. Then doing BFS like above till connectivity is built or the limit of iteration is reached.



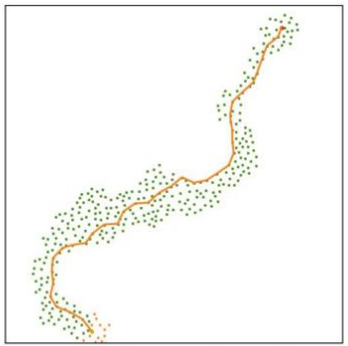
(e) BFS from x_{start} to x_{goal}



(f) BFS from x_{goal} to x_{start}



(g) second point network inference



(h) connectivity confirmed by BFS

4.2 Neural Informed RRT* [6]

PointNet: *Architecture*

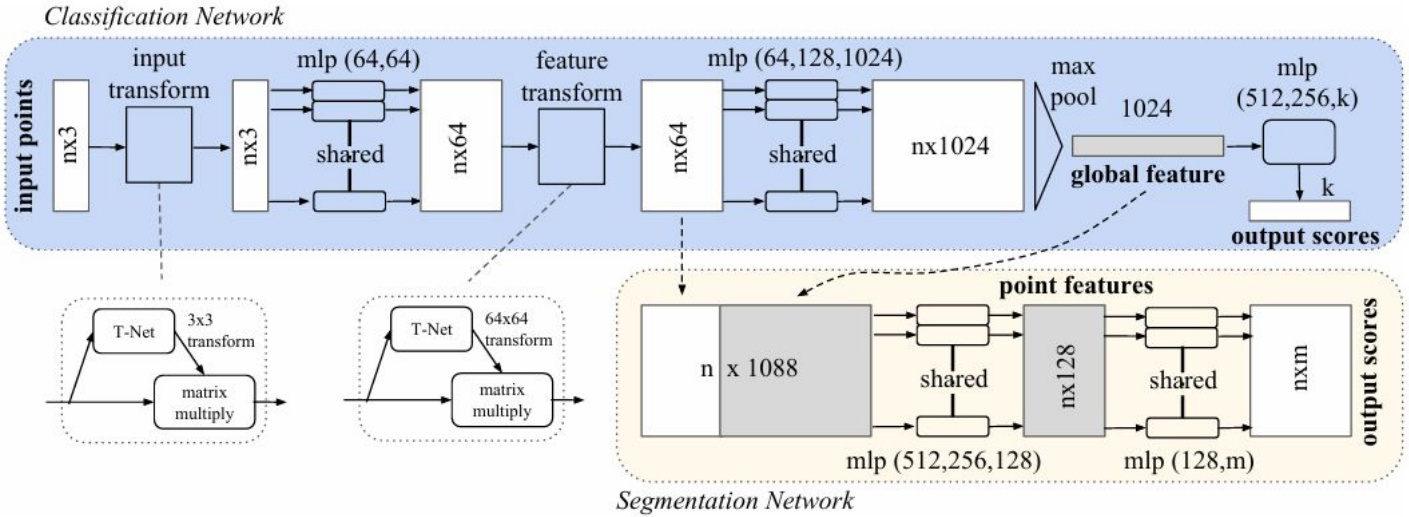


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

4.2 Neural Informed RRT* [6]

PointNet++:

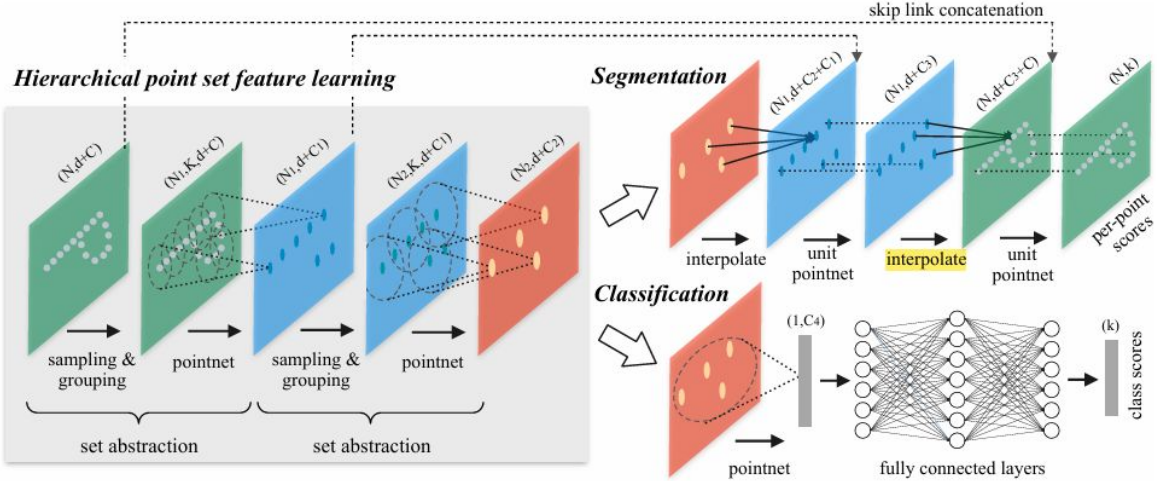


Figure 2: Illustration of our hierarchical feature learning architecture and its application for set segmentation and classification using points in 2D Euclidean space as an example. Single scale point grouping is visualized here. For details on density adaptive grouping, see Fig. 3.

EVALUATION AND SUGGESTION

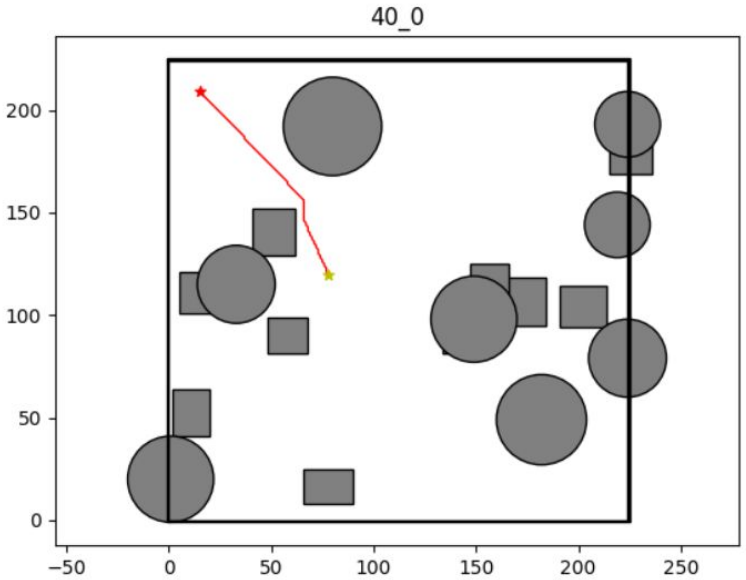
Dataset for training NIRRT*

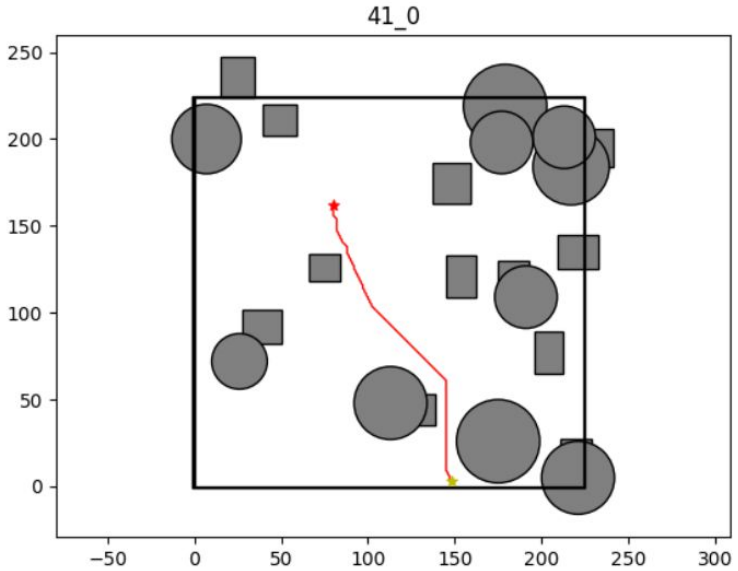
The dataset consists of 500 random 2d worlds. Each world has some rectangles and circles as obstacles, and has 4 (start, goal) pairs in free space. The main config is as follows:

```
…
env_height: 224
env_width: 224
rectangle_width_range: [16, 24]
circle_radius_range: [8, 12]
num_rectangles_range: [8, 12]
num_circles_range: [8, 12]
…
```

Is this enough for real-life application?

Dataset for training NIRRT*

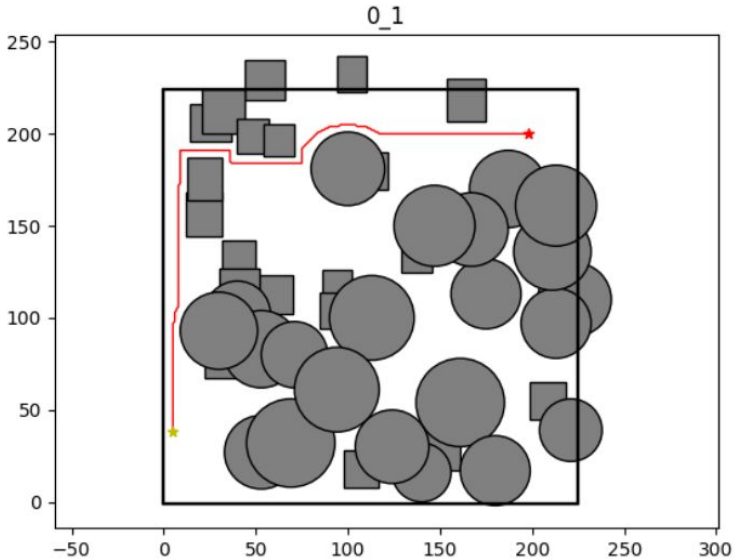




Deep evaluation for NIRRT*

We’re gonna generate some harder data to evaluate NIRRT*.

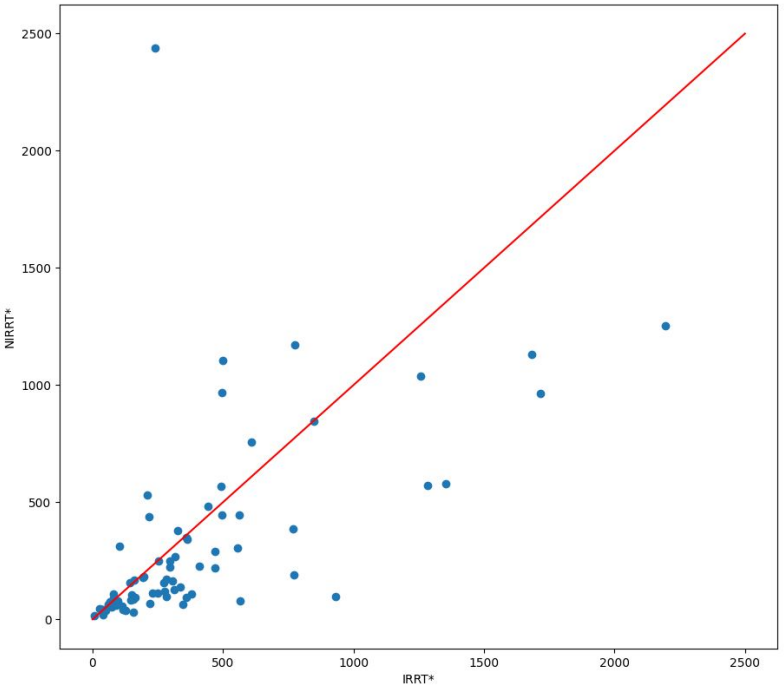
```
…
circle_radius_range: [16, 24]
num_rectangles_range: [18, 22]
num_circles_range: [18, 22]
…
```



Deep evaluation for NIRRT*

Result: The following chart shows the comparison of the number of iterations IRRT* and NIRRT* to find an initial solution for each random world problem.

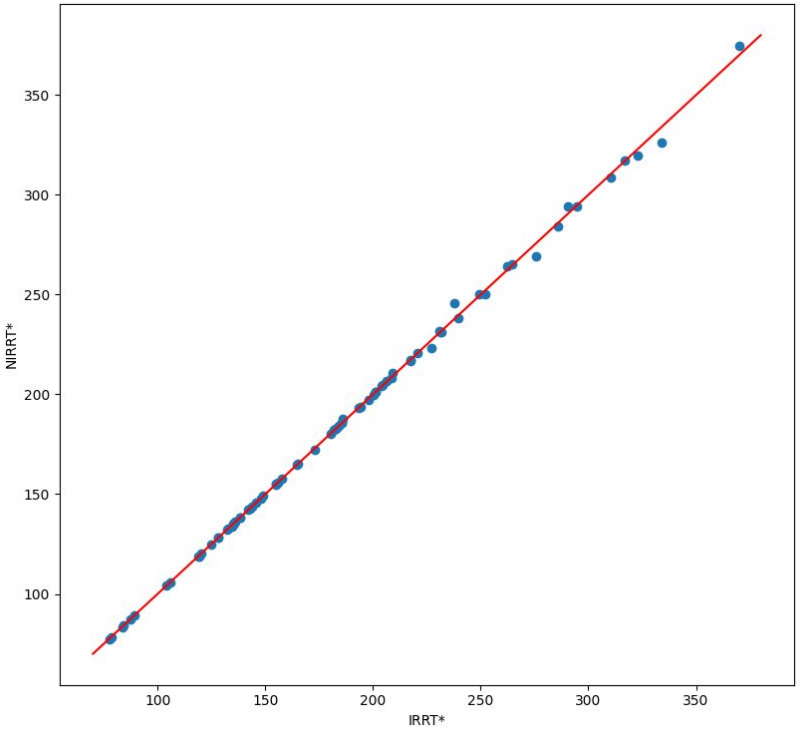
As we can see, IRRT* spends a little more iteration for finding the initial solution than NIRRT*, this proves the strength of applying deep neural network for guidance in NIRRT*



Deep evaluation for NIRRT*

Result: The following chart is about path cost.

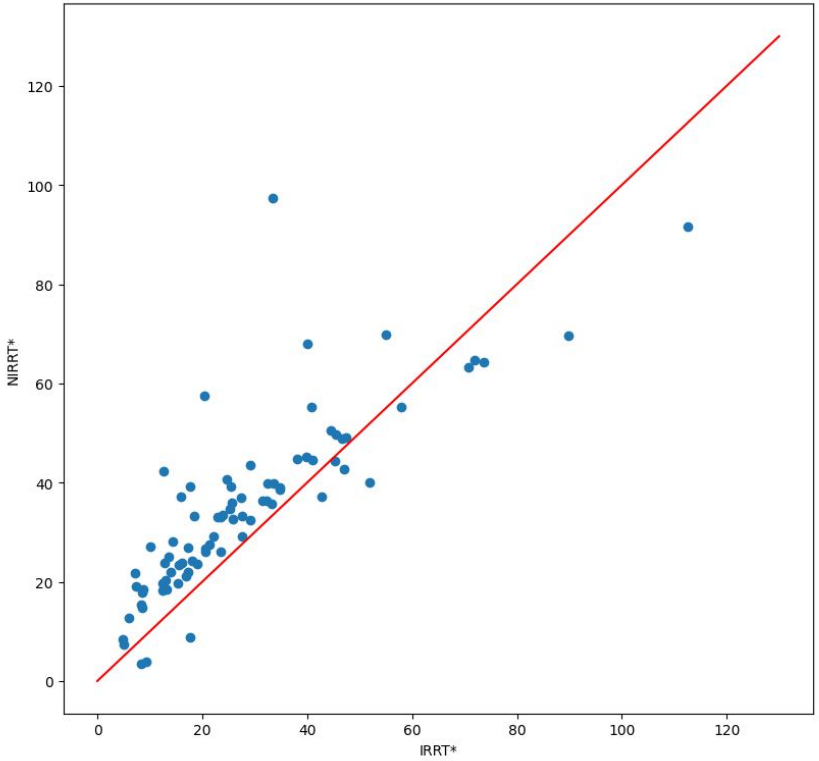
As we can see, IRRT* and NIRRT* are not so different optimizing path cost.



Deep evaluation for NIRRT*

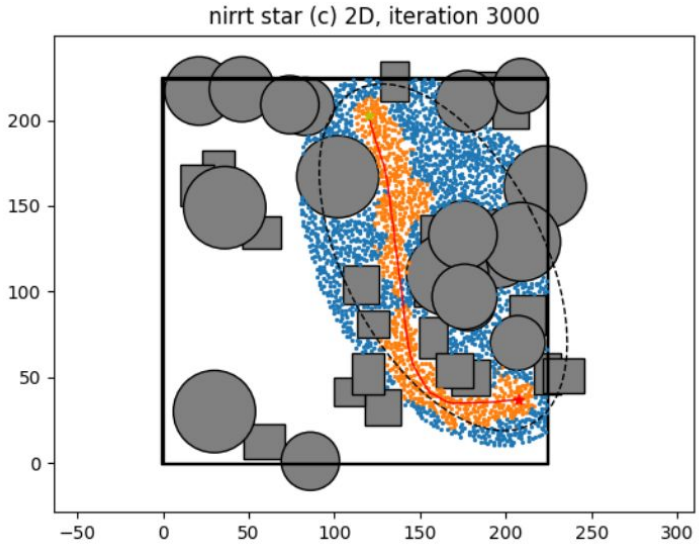
Result: The following chart shows the comparison of the time for IRRT* and NIRRT* to finish 3000 iteration.

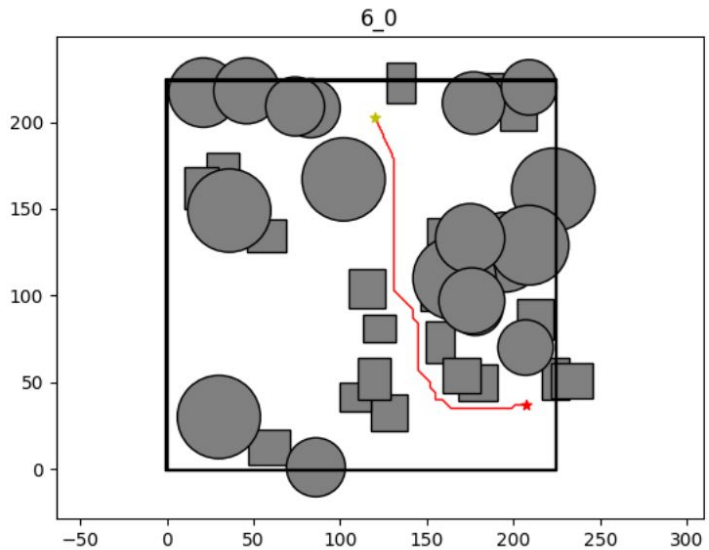
As we can see, IRRT* is faster here in almost all case.



Deep evaluation for NIRRT*

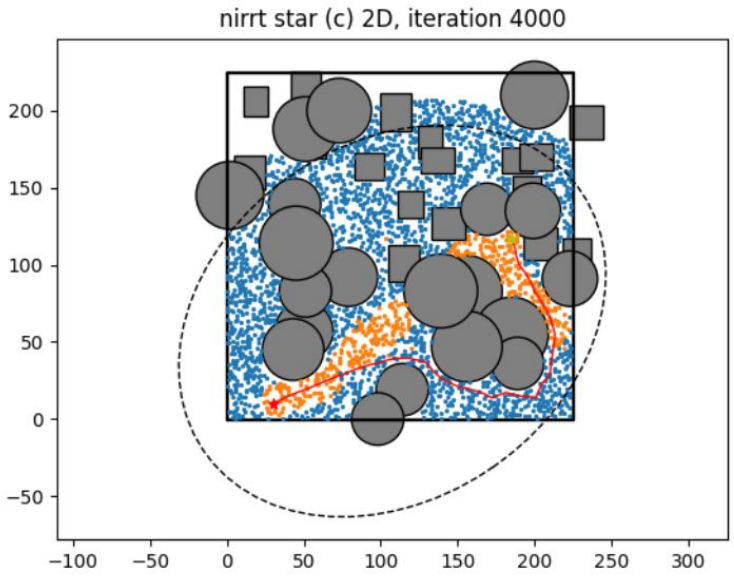
Visualization:

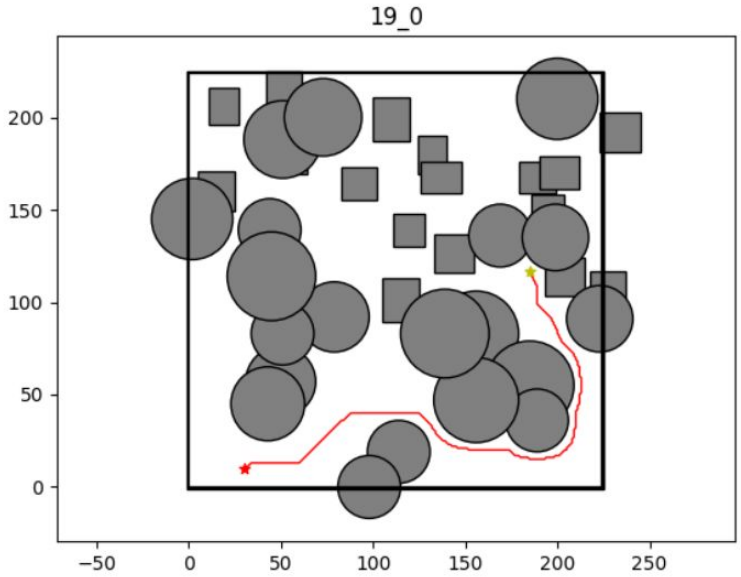




Deep evaluation for NIRRT*

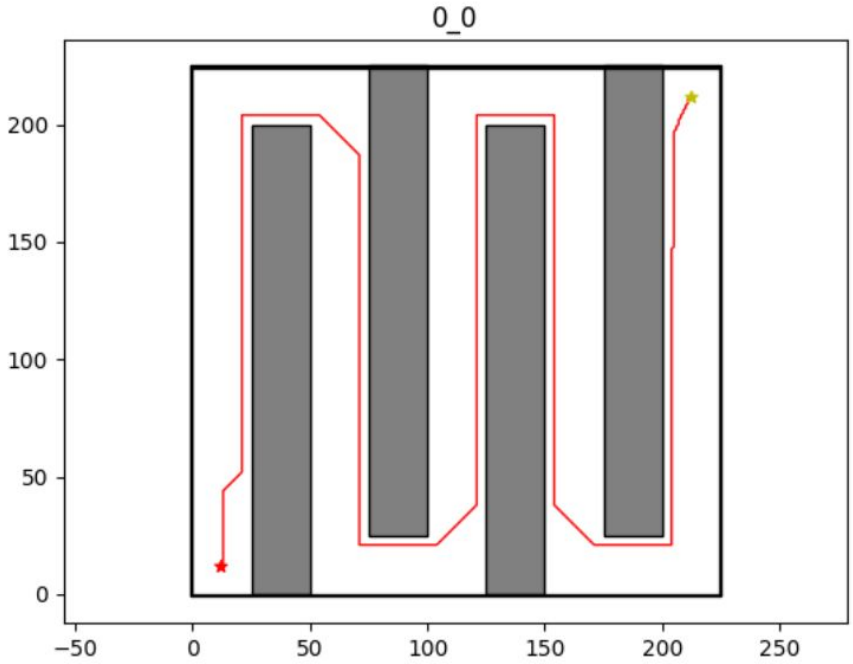
Visualization:





Deep evaluation for NIRRT*

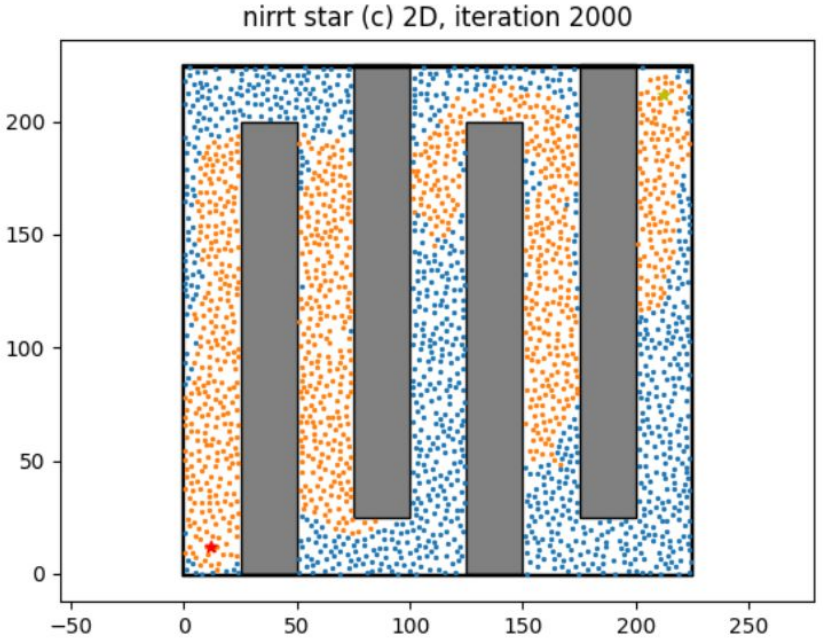
Self created map:



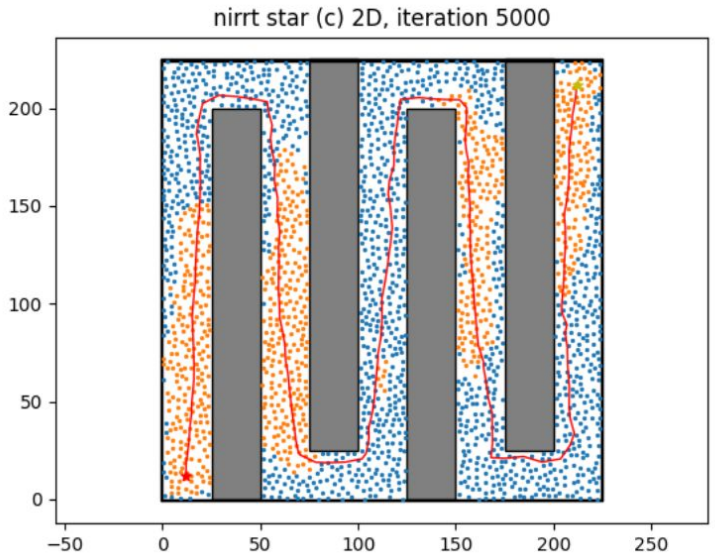
Deep evaluation for NIRRT*

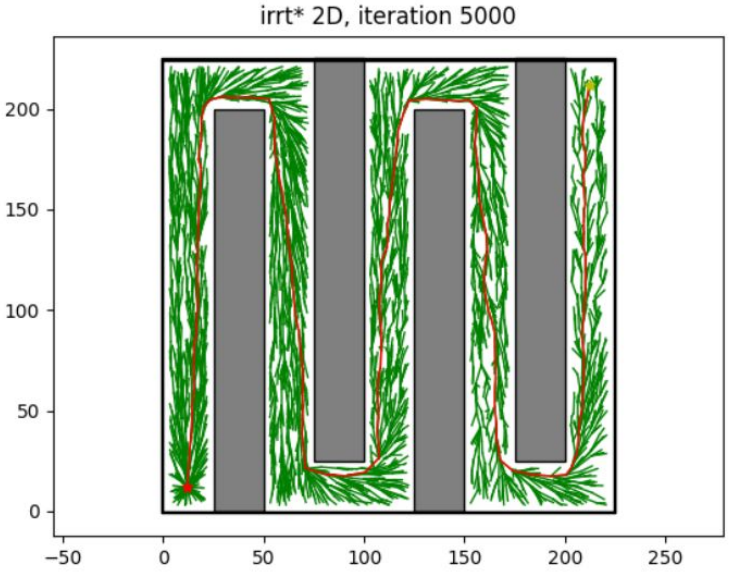
Problem with NIRRT*:

The guardian states are not connected, and this slow down the process of finding optimal path.



Deep evaluation for NIRRT*





Deep evaluation for NIRRT*

Comparison between NIRRT* and IRRT*: test 10 times on above map, calculate average number of iteration for finding first solution, total time of 400 iteration, and the final path cost.

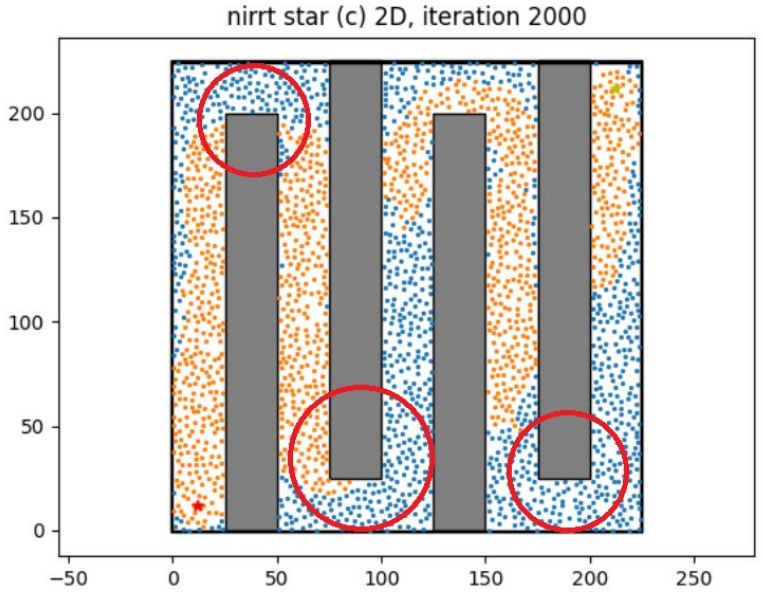
Case 0_0

	Exp No.	1	2	3	4	5	6	7	8	9	10	Avg	stdev
Number of iter for first solution	NIRRT*	3859	4197	2576	4200	4250	4071	3199	3335	4231	4221	**3813.9**	580.5843
	IRRT*	2673	2422	3006	3280	2752	3114	3105	3430	3217	2458	**2945.7**	349.8003
Time	NIRRT*	14.41301	15.45936	27.54571	20	15	14.53096	20.97187	19.93451	14	21.61263	**18.3468**	4.420895
	IRRT*	18.89009	26.52911	18.35028	15.74283	19.85023	26.08952	19.93951	15.85062	18.08596	33.54751	**21.28757**	5.673058
Path cost	NIRRT*	1099.249	1110.2	1096.902	1099	1113	1108.832	1087.411	1086.728	1102	1140	**1104.332**	15.32305
	IRRT*	1079.62	1072.128	1081.334	1084.568	1078.494	1082.258	1075.488	1076.879	1088.45	1075.589	**1079.481**	4.834843

Deep evaluation for NIRRT*

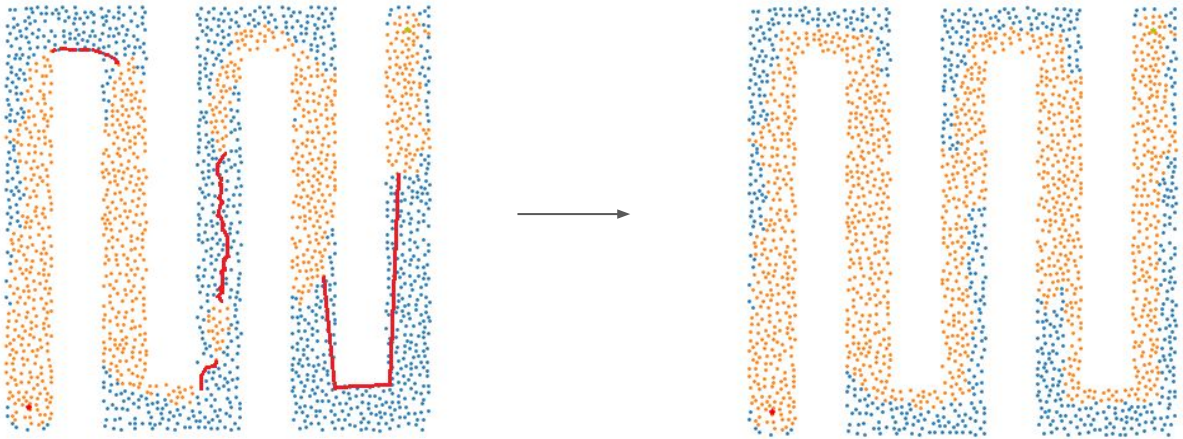
Result: The number of iterations for IRRT* to find the first solution is less than NIRRT*. And because of that, IRRT* has more time to optimize the path and achieve better path cost than NIRRT*.

Why this map is hard?



Suggestion

Improve Neural-connect:



We will try to connect all ‘blobs’. First, use a search-base algorithm to find the path connecting this blob to the next until all blob are connected into one.

Then, we will choose all points close to the red path to be guardian states.

Suggestion

After making guardian states connected: the number of iterations for finding the first solution is decreases, but the time used is higher.

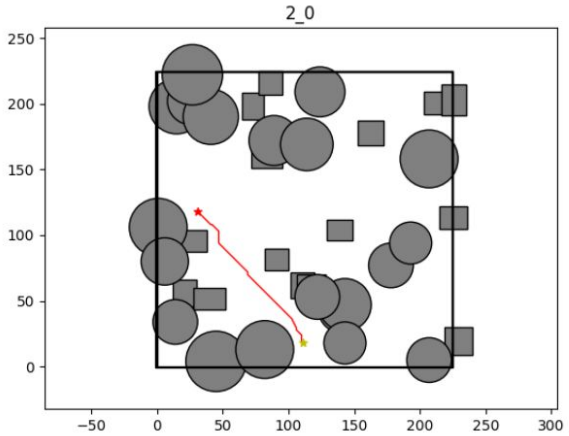
	Exp No.	1	2	3	4	5	6	7	8	9	10	Avg	stdev
Number of iter for first solution	NIRRT*	3859	4197	2576	4200	4250	4071	3199	3335	4231	4221	**3813.9**	580.5843
	NIRRT* - impv_nc	3430	3907	2795	3315	4165	2797	3933	2911	3147	2935	**3333.5**	509.8207

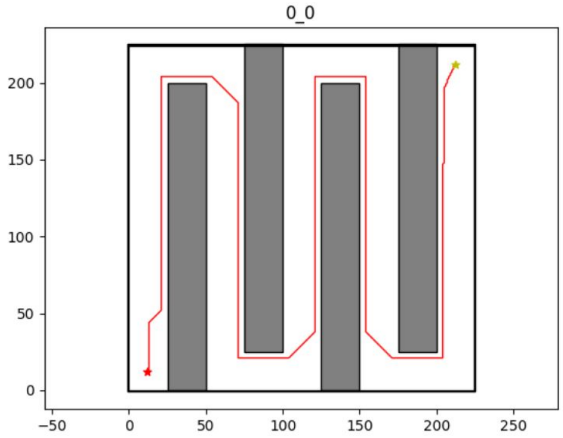
Suggestion

Improve the dataset:

- Generate more complicated dataset.
- Collect real-world dataset.

Note: A complicated map is not just the map that has many obstacles. We should consider the map complicated when the topology on it is hard to infer the guardian states, or the optimal path is longer and harder to find, ...





THE END