JC3504 Robot Technology

Lecture 10: Plan and Search

Dr Xiao Li xiao.li@abdn.ac.uk

Dr Junfeng Gao Junfeng.gao@abdn.ac.uk

Outline

- Map Representations
- Path Planning
    - Dijkstra’s algorithm
    - A* algorithm
    - Rapidly Exploring Random Trees

Map Representations

Map Representations

Map representations are crucial for enabling robots to navigate and understand their environment, typically involving the creation of spatial models like grid maps, feature-based maps, and semantic maps.

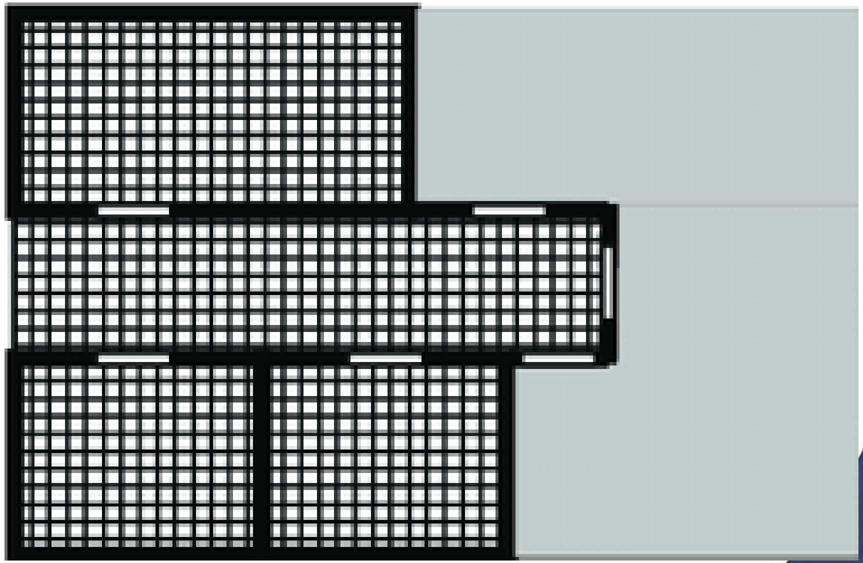
These representations vary in complexity and detail, from simple occupancy grids that denote space as free or occupied, to complex semantic maps that include labels and attributes of objects within the environment.

Different map representations are utilised for varied tasks, with each type optimally tailored to facilitate specific functionalities such as navigation, obstacle avoidance, object recognition, and environmental understanding in robotics.

Grid Maps

A two-dimensional representation where the environment is divided into a grid of cells, each indicating whether the space is occupied, free, or unknown.

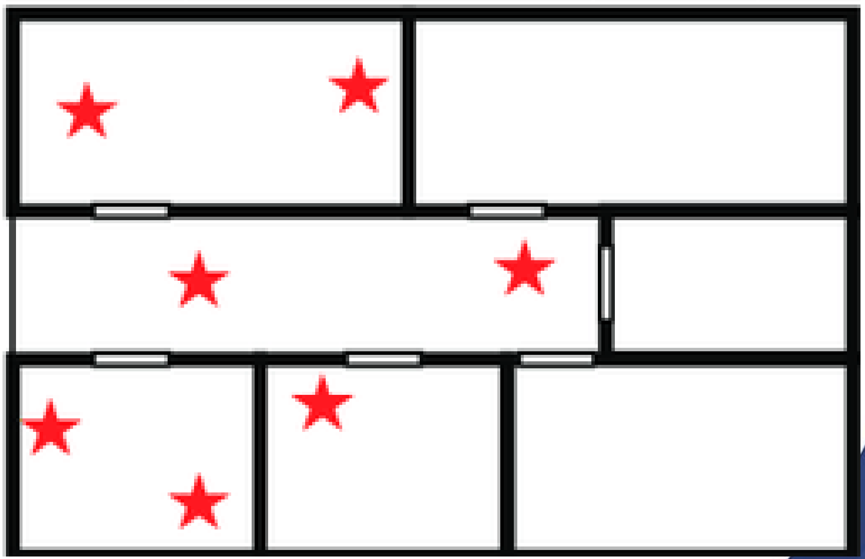
This type is particularly useful for navigation and obstacle avoidance in structured environments.



Feature-based Maps

These maps identify and use distinct environmental features (like edges, corners, or specific objects) as landmarks for navigation and localization.

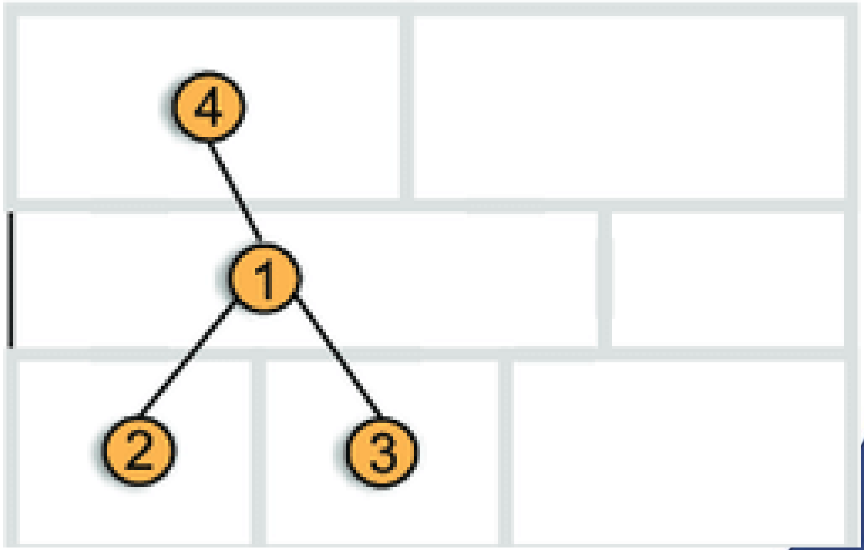
They are less memory-intensive than grid maps and are useful in environments where unique features can be reliably detected.



Topological Maps

Simplifying the environment into nodes and edges, topological maps represent spatial relationships and connectivity between different locations.

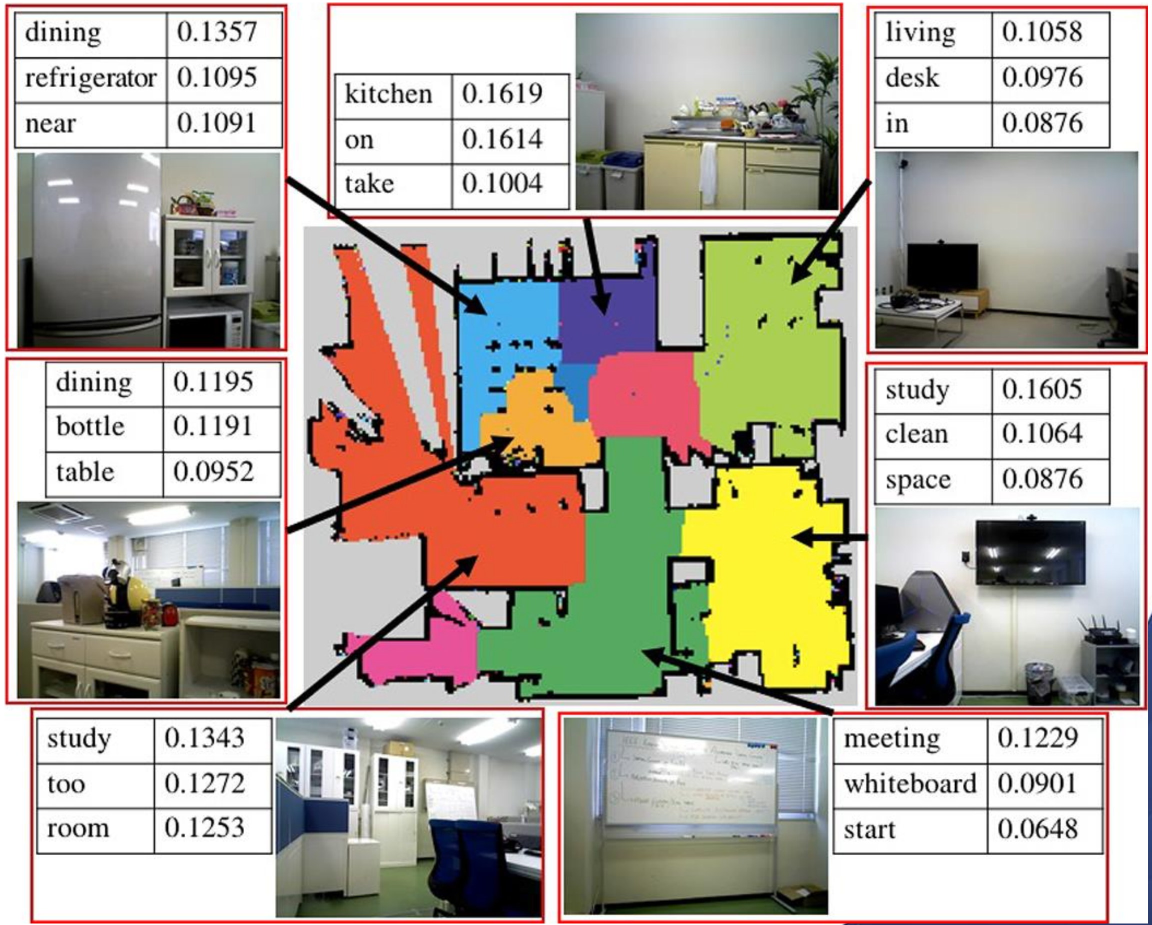
This abstract form of mapping is beneficial for planning routes through large or complex environments.



Semantic Maps

Going beyond physical space representation, semantic maps include information about the type and function of objects within the environment.

They enable robots to perform tasks that require understanding of the environment's context, such as fetching an item from a room.



dining	0.1357
refrigerator	0.1095
near	0.1091

kitchen	0.1619
on	0.1614
take	0.1004

living	0.1058
desk	0.0976
in	0.0876

dining	0.1195
bottle	0.1191
table	0.0952

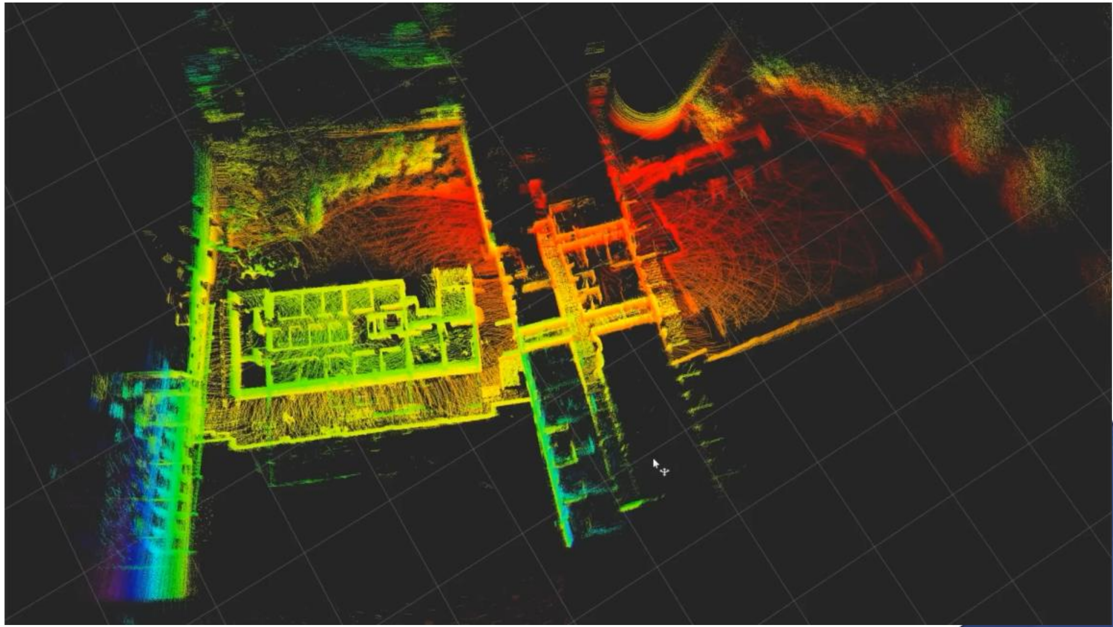
study	0.1605
clean	0.1064
space	0.0876

study	0.1343
too	0.1272
room	0.1253

meeting	0.1229
whiteboard	0.0901
start	0.0648

3D Maps and Point Clouds

With the advent of advanced sensors, 3D mapping has become more prevalent. These maps offer a three-dimensional view of the environment, providing detailed information about the shapes and locations of objects and surfaces, which is crucial for navigation and manipulation tasks in complex environments.



Path Planning

Path Planning

Path planning allows autonomous mobile robots and manipulators to find a path to move between two points.

Depending on the choice of the planning algorithm, a path could satisfy various degrees of optimality with respect to some criteria such as minimising path length, minimising turns, or minimising the amount of braking.

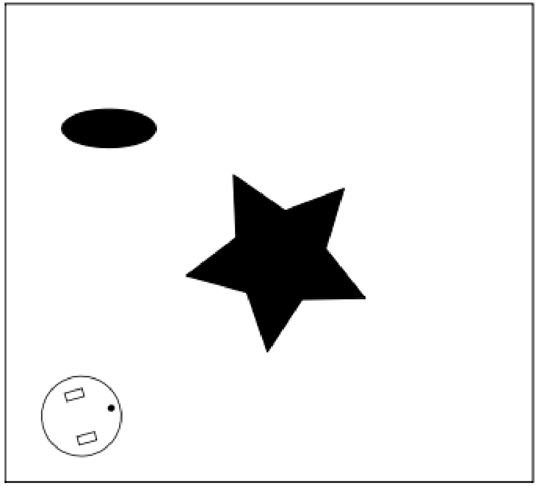
Algorithms to find a shortest path are important not only for robotics applications, but also in network routing, video games, and understanding protein folding.

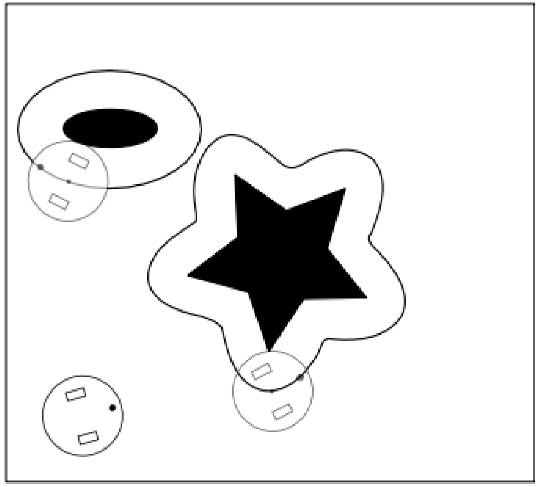
Path planning requires a suitable representation of the environment such as a map we just introduced, and a perceptual understanding of the robot’s location with respect to such representation. We will assume for now that the robot is able to localize itself, and is equipped with a map.

The Configuration Space

In the vast majority of path-planning algorithms, the robot is treated as a point-mass element with no volume.

It is possible for the robot to be reduced to a point-mass while growing all obstacles by its radius.



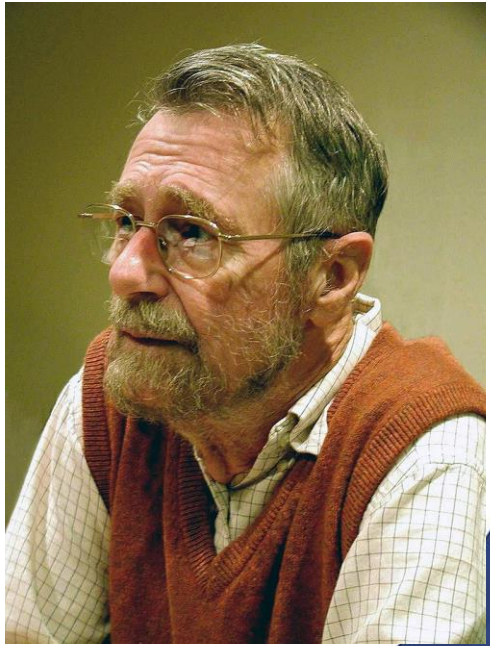


Dijkstra’s Algorithm

Dijkstra's algorithm, developed by Dutch computer scientist Edsger W. Dijkstra in 1956, is a foundational technique used in computing to find the shortest paths between nodes in a graph, which may represent.

The algorithm works by iteratively selecting the node with the smallest known distance from the start node, calculating the distance through it to each of its neighbours, and updating these distances if smaller than previously known. The process is repeated until the shortest paths from the start node to all other nodes in the graph are established.

Dijkstra's algorithm assumes that all weights are non-negative, as it relies on the fact that the shortest path between any two nodes does not contain any cycles.


Edsger W. Dijkstra

Dijkstra’s Algorithm

< Video: 10 - How Dijkstra's Algorithm Works.mp4>

A* Algorithm

The A* algorithm is a search and pathfinding algorithm renowned for its efficiency and accuracy. The essence of A* lies in its ability to combine features of Dijkstra's algorithm (focusing on the shortest path) and the Greedy Best-First-Search (prioritising proximity to the goal).

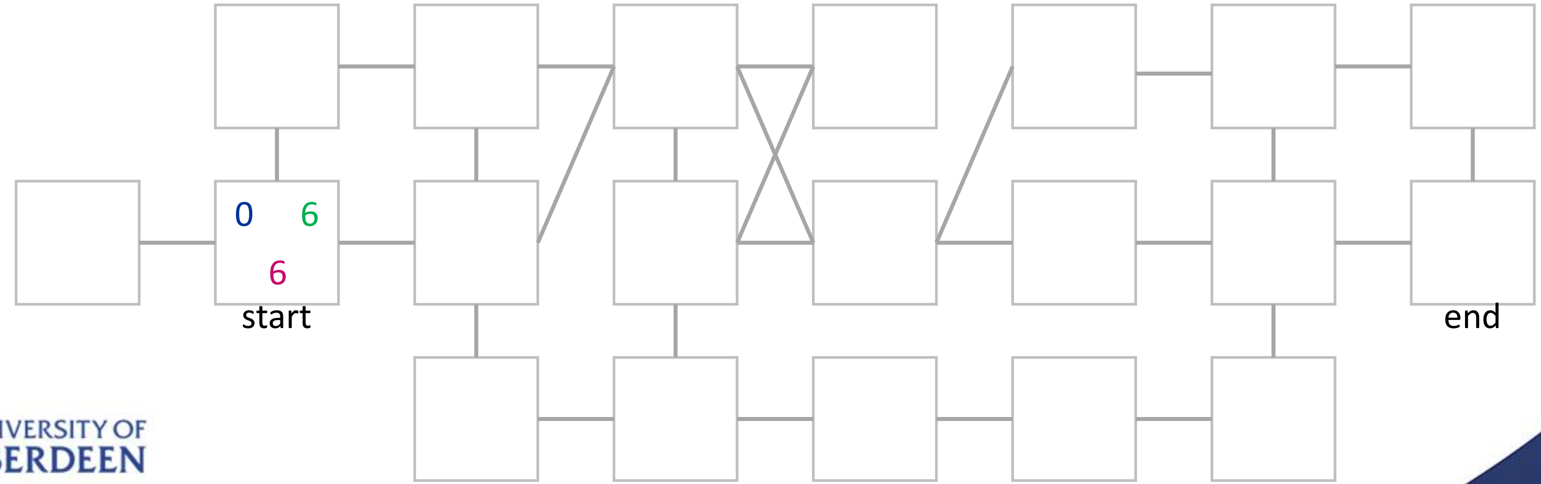
A* operates by maintaining a priority queue of paths based on the cost function

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic estimate of the cost to get from n to the goal. The algorithm proceeds by exploring paths in the order of their estimated total cost, expanding paths to their neighbouring nodes until the goal is reached.

A* Algorithm

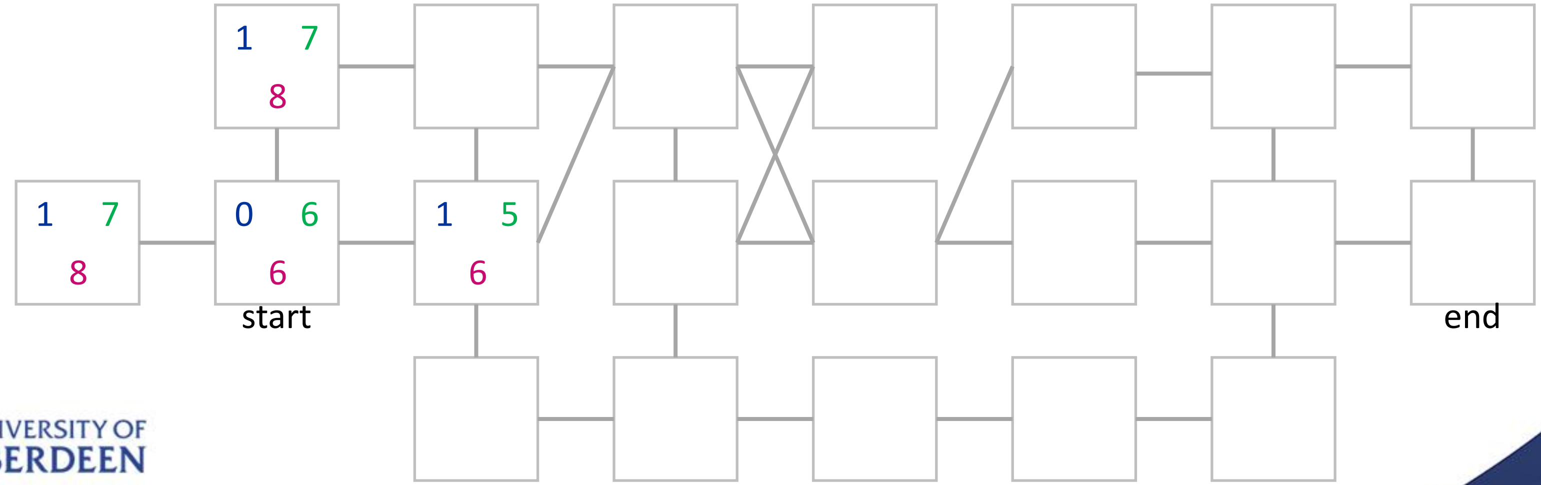
Initially, start by placing the start node on the open list, which contains nodes to be evaluated. The start node's cost: $f(start) = g(start) + h(start) = 0 + dis(start, end)$. Usually, we use the distance between two points as an estimate of $h(start)$.



A* Algorithm

Initially, start by placing the start node on the open list, which contains nodes to be evaluated. The start node's cost: $f(start) = g(start) + h(start) = 0 + dis(start, end)$. Usually, we use the distance between two points as an estimate of $h(start)$.

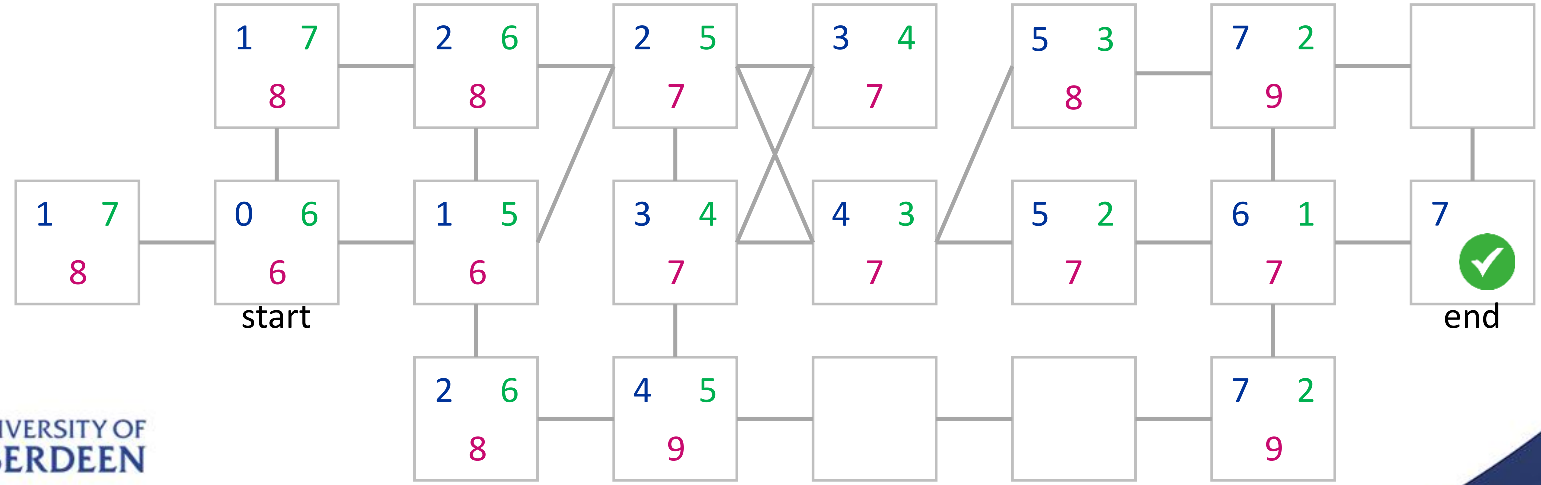
Then, we calculate the cost of the neighbouring nodes.



A* Algorithm

Then, we only explore the node with the lowest cost i.e. calculate the cost of its neighbouring nodes.

We repeat the process.



Dijkstra’s VS. A*

Dijkstra's algorithm is a pathfinding method that guarantees the shortest path between two points in a graph by systematically exploring all possible routes, prioritising those with the lowest cumulative cost without considering the direction of the goal.

In contrast, A* algorithm enhances Dijkstra's approach by incorporating a heuristic estimate of the distance to the goal, allowing it to focus the search direction towards the destination and often find the shortest path more efficiently.

Dijkstra’s VS. A*

< Video: 10 - Dijkstra's vs. A＊ (A-Star).mp4>

Rapidly-exploring Random Trees (RRT)

Rapidly-exploring Random Trees (RRT) is a pathfinding and motion planning algorithm that excels in efficiently navigating complex and high-dimensional spaces.

By iteratively expanding a tree from the start point towards random locations within the search space, RRT quickly covers the area, making it highly effective in environments with obstacles and challenging terrain.

Rapidly-exploring Random Trees (RRT)

Rapidly-exploring Random Trees (RRT) is a pathfinding and motion planning algorithm that excels in efficiently navigating complex and high-dimensional spaces.

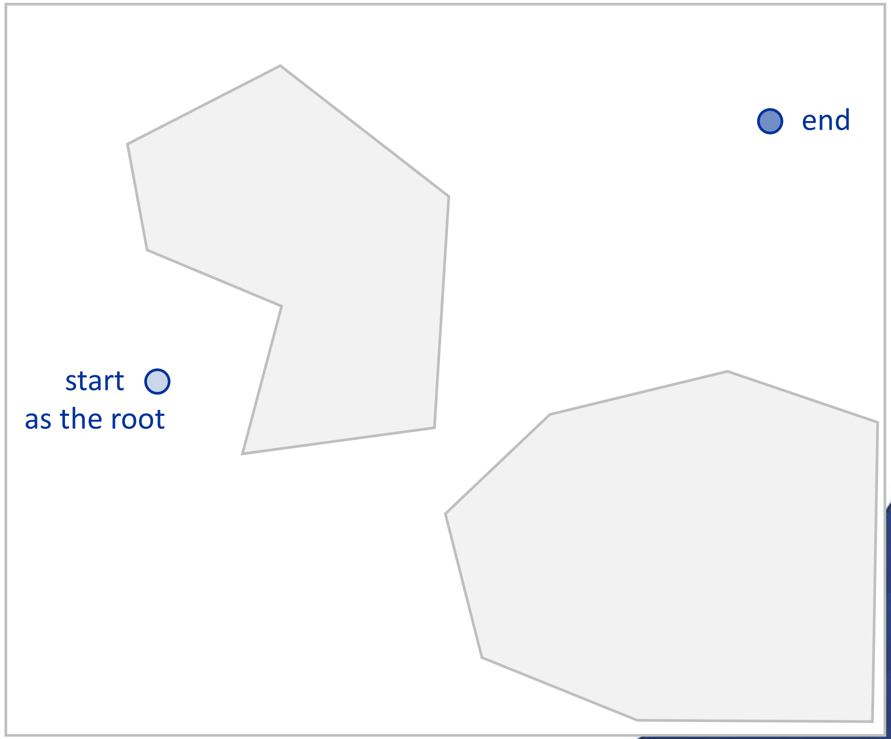
By iteratively expanding a tree from the start point towards random locations within the search space, RRT quickly covers the area, making it highly effective in environments with obstacles and challenging terrain.

A main feature of the RRT algorithm is its ability to quickly explore large spatial areas, which is suitable for scenarios where the quality of the solution is not very high.

Rapidly-exploring Random Trees (RRT)

1. Initialise

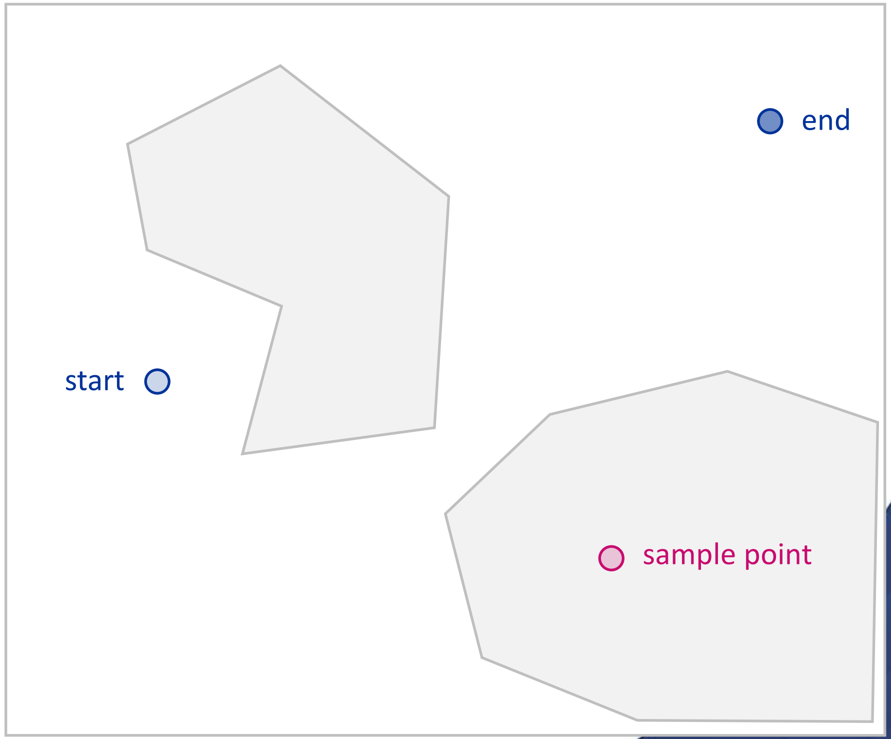
Given a map, we start by initialising the tree with the starting point as its root node.



Rapidly-exploring Random Trees (RRT)

2. Random Sampling

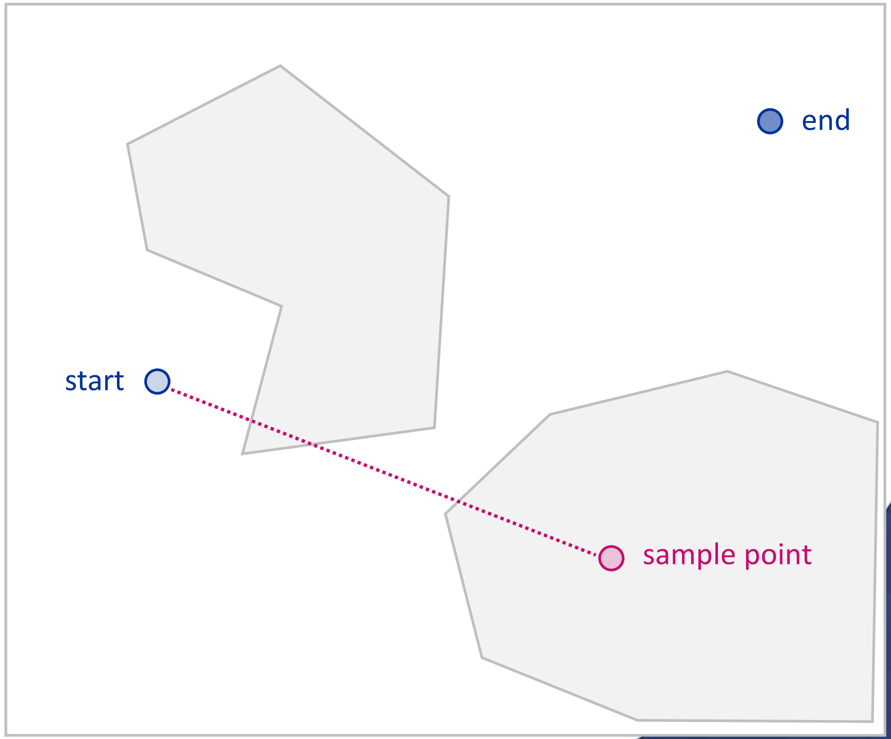
Randomly select a point in the space as a sample point. This selection can be entirely random or guided by some strategy to improve search efficiency.



Rapidly-exploring Random Trees (RRT)

3. Find the Nearest Node

Identify the node in the tree that is closest to the sample point, using the nearest node. The distance calculation often relies on the Euclidean distance or other metrics suitable for the specific problem.



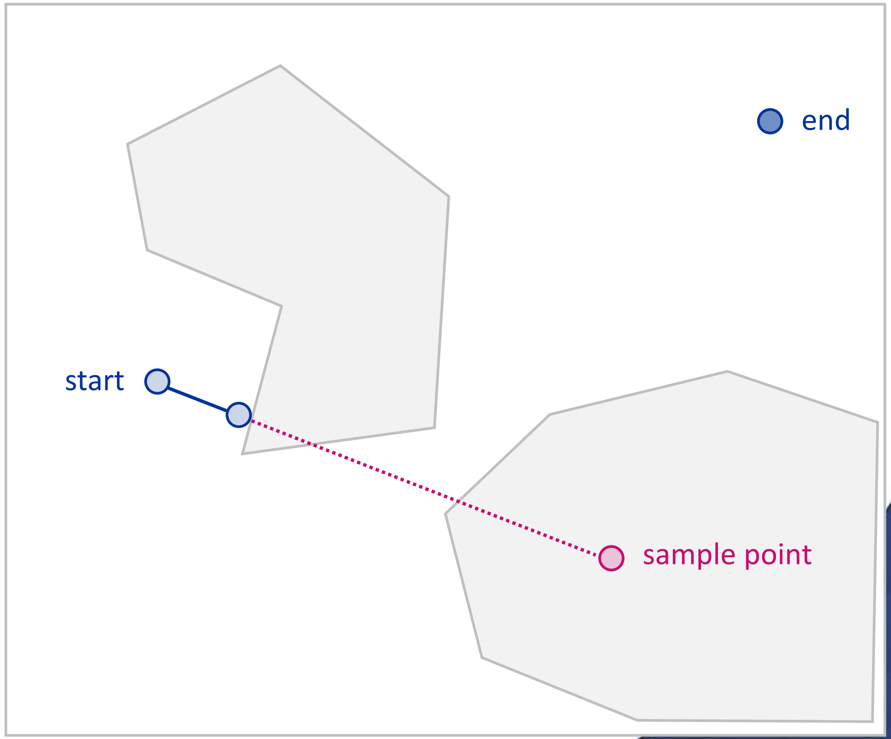
Rapidly-exploring Random Trees (RRT)

4. Extend a New Node

Extend from the nearest node towards the sample point to create a new node.

The extension's length may be fixed or dynamically adjusted based on the environment.

The key here is to ensure the new node's position is viable, meaning it doesn't cross any obstacles.



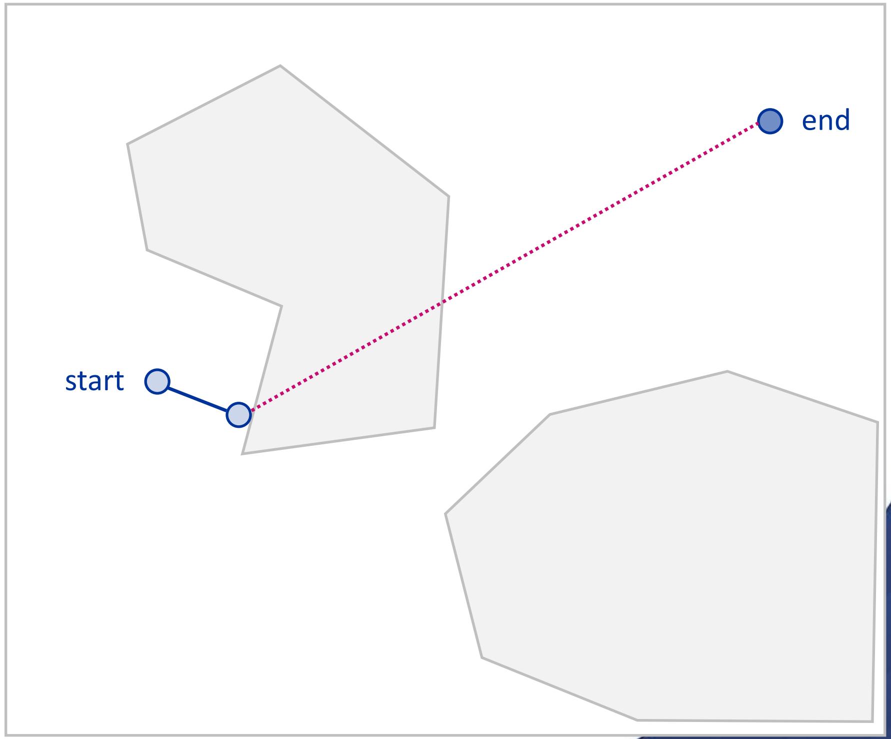
Rapidly-exploring Random Trees (RRT)

5. Check for Goal

Determine whether the newly added node is close to the goal.

If so, attempt to directly connect the new node with the goal point and check if the path is feasible.

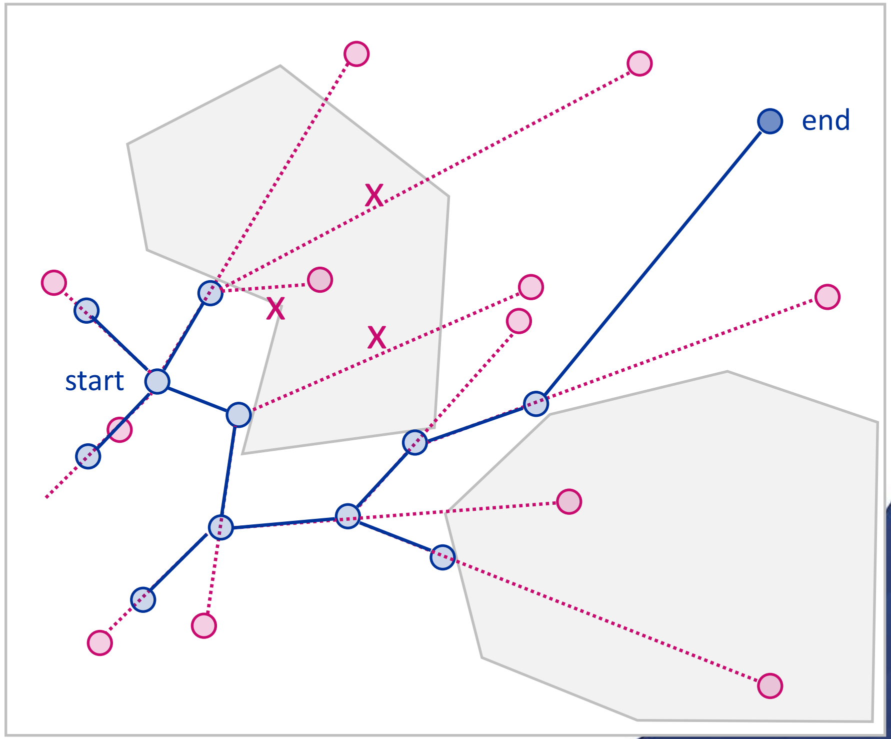
If the path is viable, the algorithm terminates and returns the path.



Rapidly-exploring Random Trees (RRT)

6. Repeat Steps

Repeat steps 2-5 until a feasible path from the start to the goal is found, or a preset iteration count or time limit is reached.

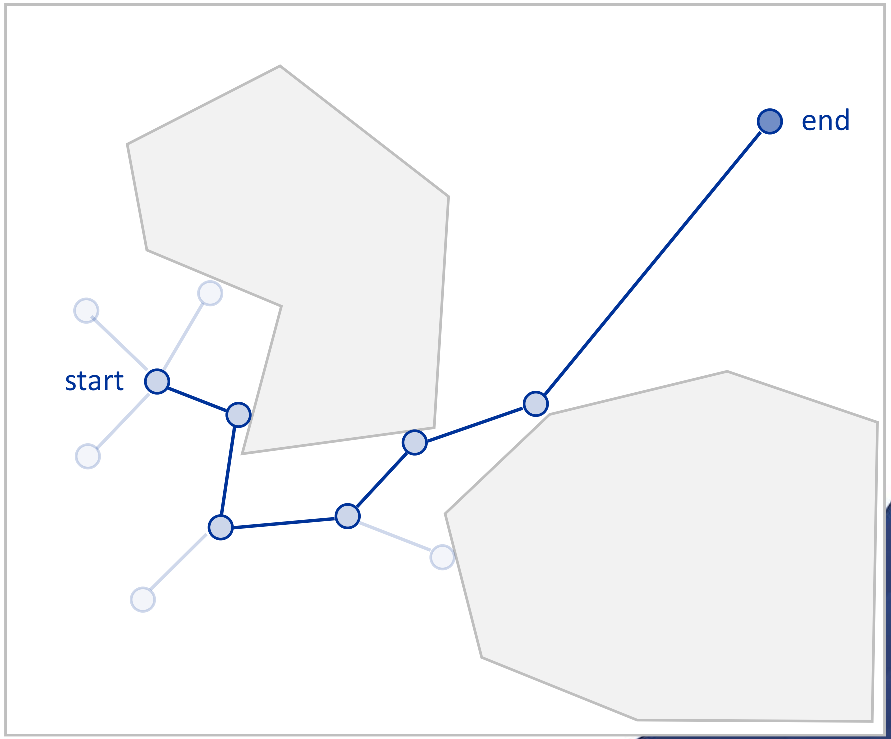


UNIVERSITY OF ABERDEEN

Rapidly-exploring Random Trees (RRT)

6. Repeat Steps

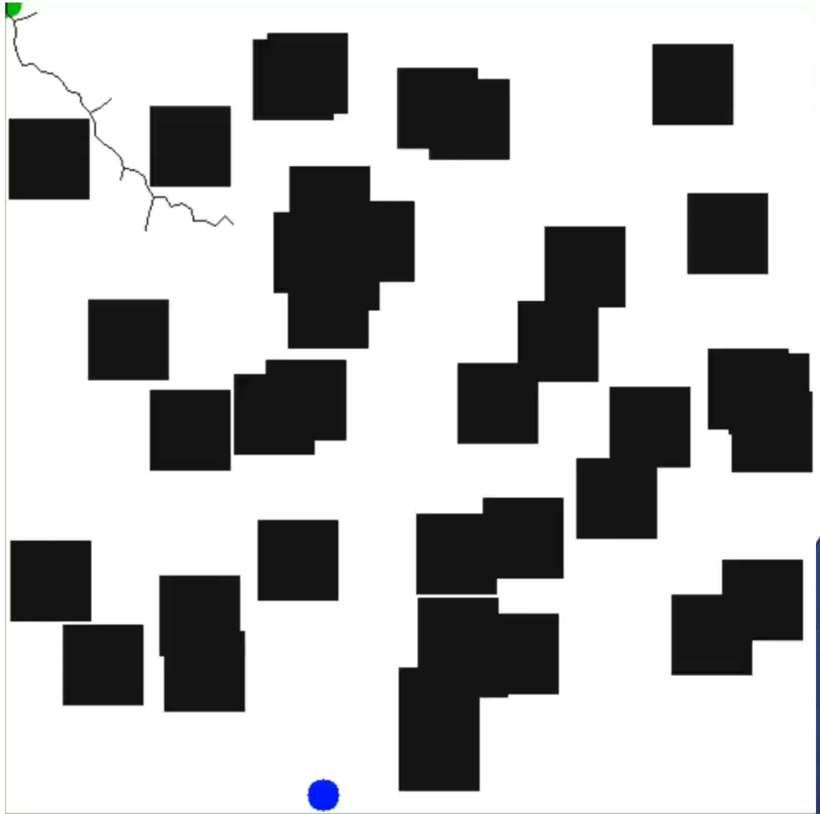
Repeat steps 2-5 until a feasible path from the start to the goal is found, or a preset iteration count or time limit is reached.



Rapidly-exploring Random Trees (RRT)

A sampling-based planning algorithm finds paths by sampling random points in the environment. Heuristics are used to maximize the exploration of space and bias the direction of search. This makes these algorithms fast, but neither optimal nor complete.

As the resulting paths are random, multiple trials might lead to entirely different results.



UNIVERSITY OF ABERDEEN

Conclusion

Map representations are crucial for enabling robots to navigate and understand their environment, typically involving the creation of spatial models like grid maps, feature-based maps, and semantic maps.

The first step in path planning is choosing a map representation that is appropriate to the application.

The second step is to reduce the robot to a point mass, which allows planning in the configuration space (or C-space).

This allows the application of general-purpose shortest path graph-based algorithms, which have applications in a large variety of domains and that are not limited to robotics.

There is no one-size-fits-all algorithm for a path planning algorithm and care must be taken to select the right paradigm (e.g. single-query vs. multi-query), heuristics, and parameters.