Robotic Navigation and Exploration

Week 3: Motion Planning

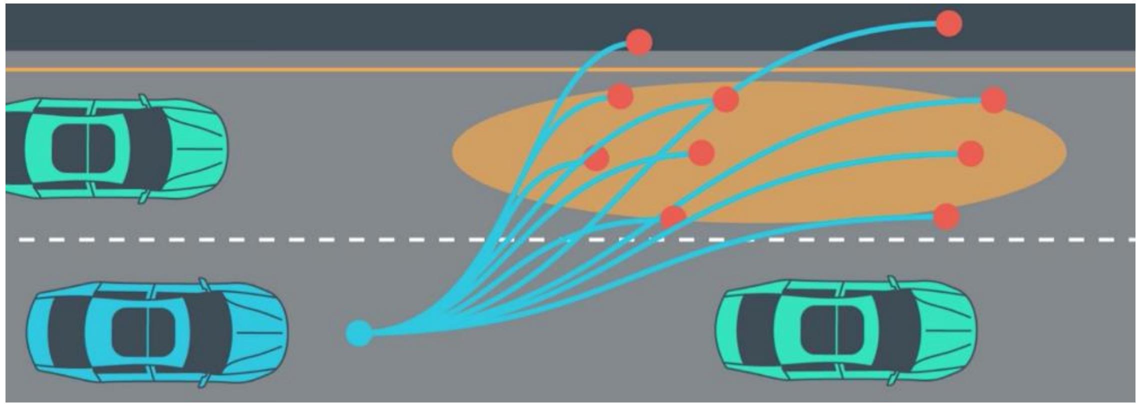
Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

Outline

- Introduction to Motion Planning
- Path Planning
  - Graph Search Based Methods
  - Sampling Based Methods
- Curve Interpolation
- Trajectory Planning
  - State Lattices

Motion Planning

- Motion planning aims to find the motion steps that moves the robot from the source to destination. 找出一连串的动作序列，将机器人循序渐进的移动到终点
- The planned motion sequence needs to satisfy the following requirements:
  - Avoiding collision with known obstacles 能够避开障碍物
  - Motion constraints (e.g. maximum speed) 运动限制
  - Smooth path 平滑路径
  - Minimal length 最短路径

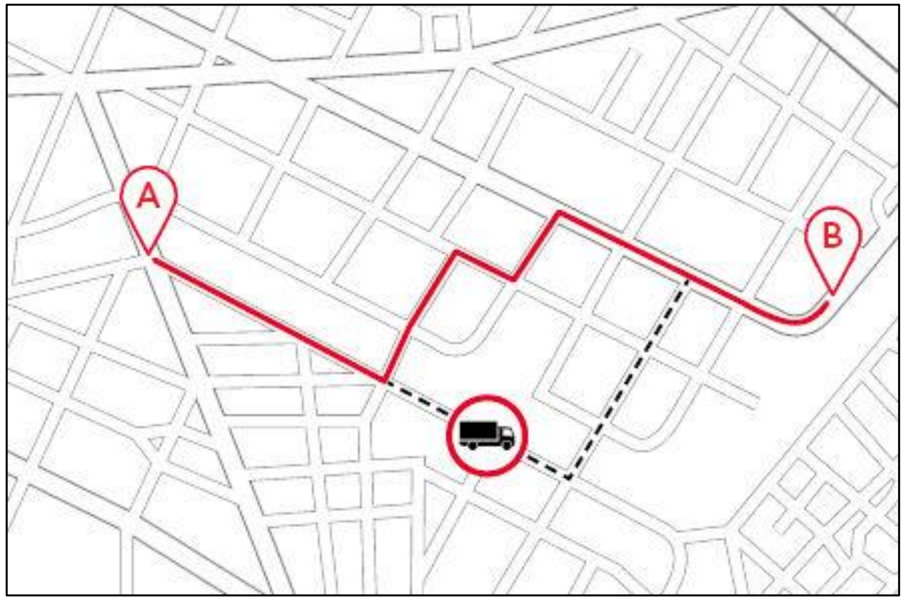


Path Planning

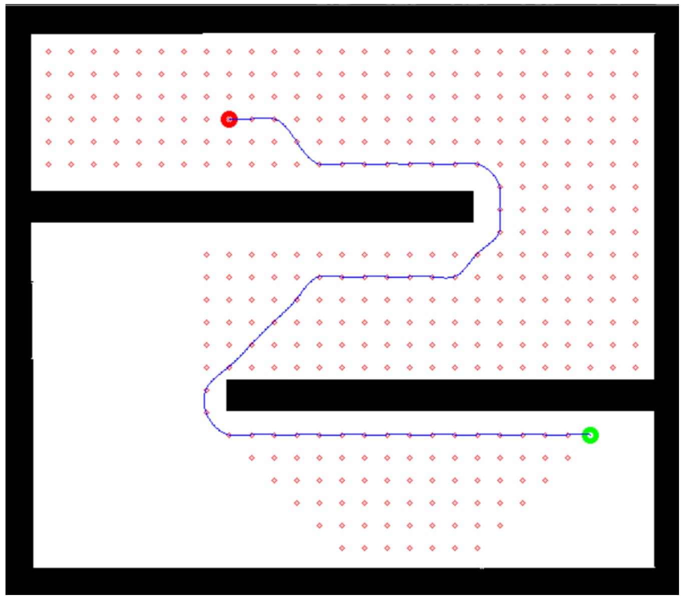
找出起点与中间的的离散中间点

- Find the way-points from the source position to the target position.

地图形式



Route Planning

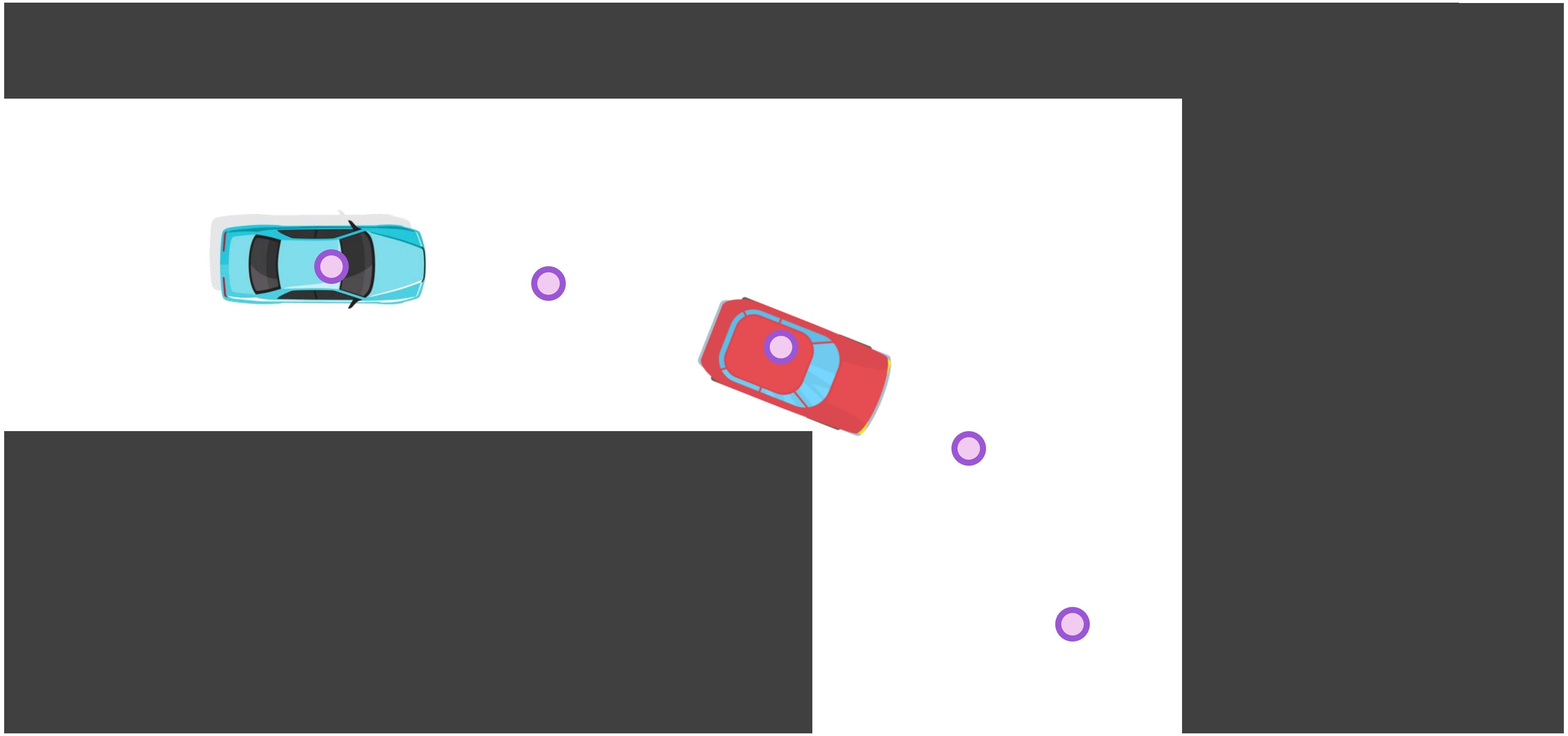


Planning in Grid Space

地图看成grif,十字路口看成note每条马路看成两个note的age

Path Planning

way-point太稀疏了车子可能找到相同点，或一次跳太多点，需要在离散点中间内插出连续点
稀疏的点构成的路径不够平顺造成转折过大，不利于行驶
路径演算法中需要知道路径的一阶微分信息（每个时间点当前方向）或二阶信微分息（每个时间点的曲率）

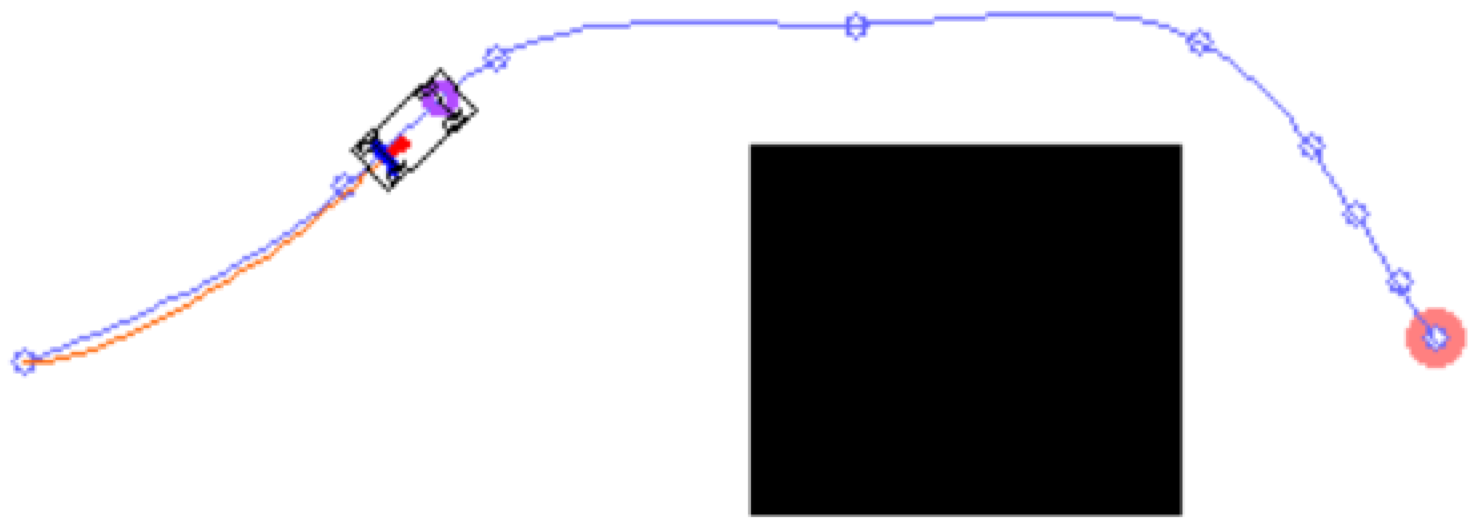


曲线内插

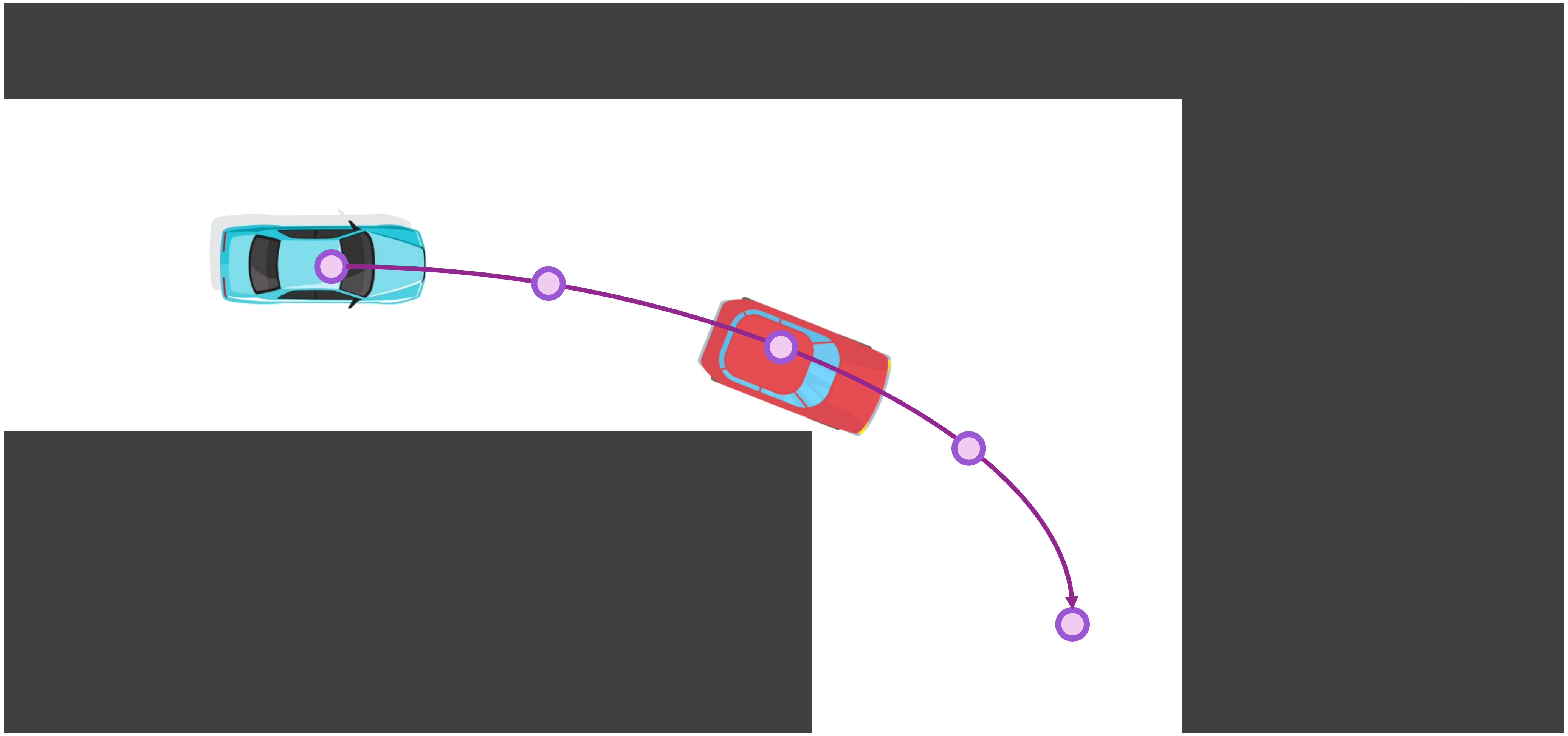
Curve Interpolation

- Given the way points and a curve function, interpolate the middle point to make the path smooth.
- We can also utilize the curve function to compute the derivative information of the path.

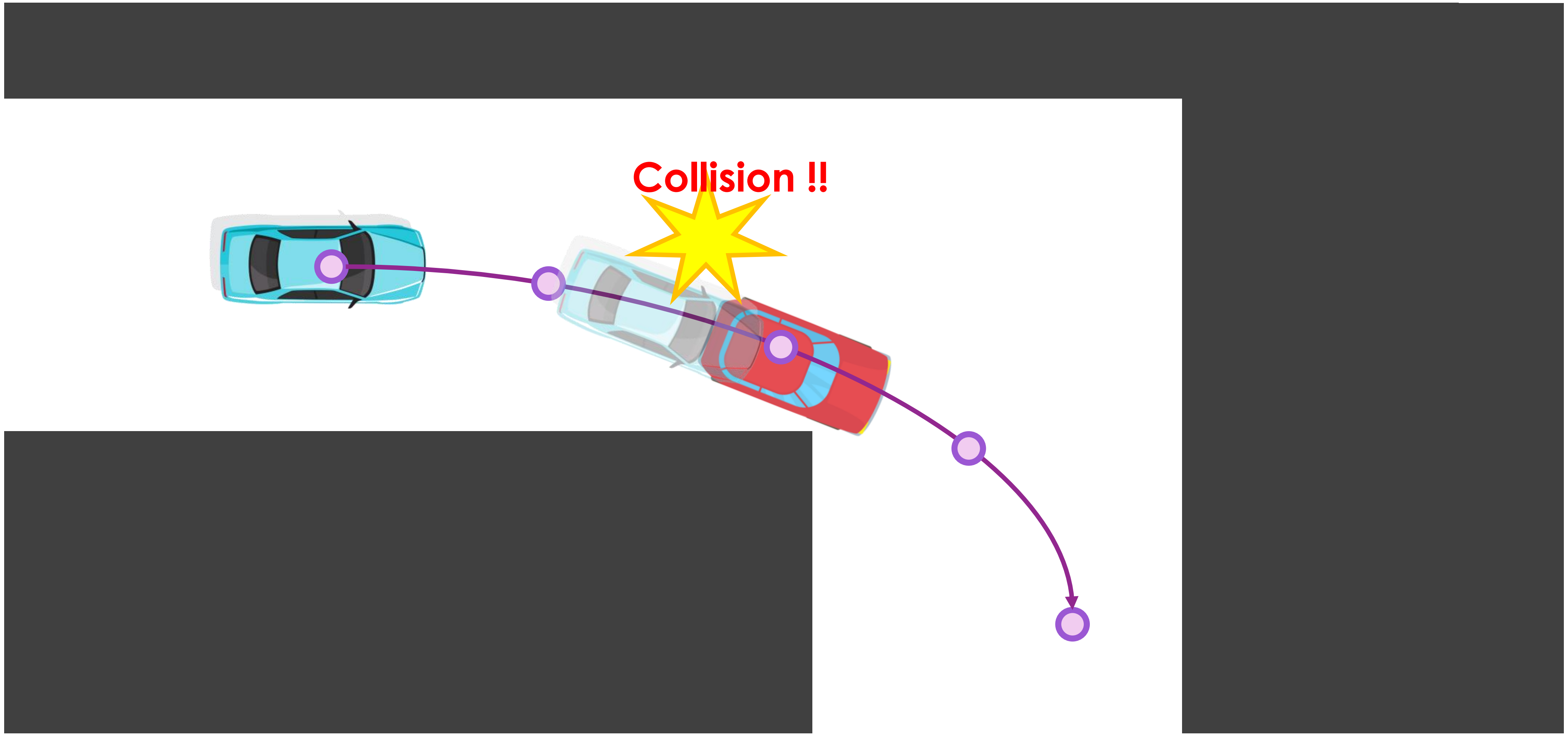
使用二次微分函数，限制每个时间点的前后一阶微分都要一致



Curve Interpolation



Dynamic Environment Problem

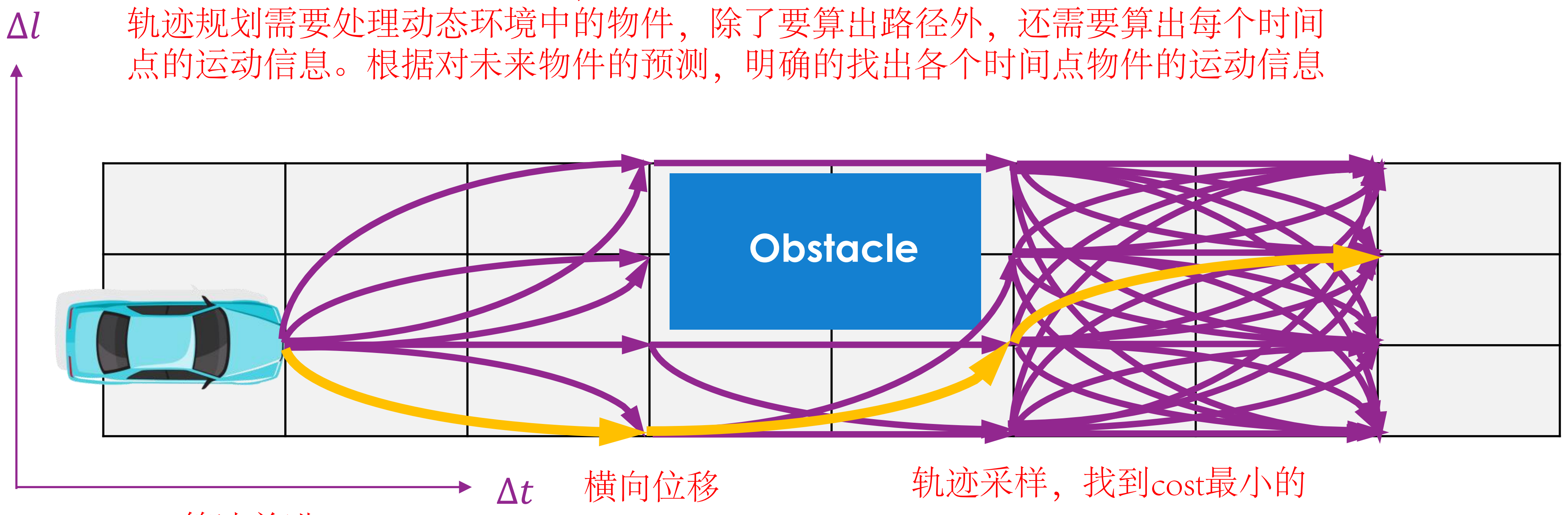


Trajectory Planning

轨迹规划

- Plan the motion sequence that contains speed and acceleration information to handle the dynamic environment.

路径规划只需要找到一条路径，不需要关心在什么时间点走到路径上的什么位置
轨迹规划需要处理动态环境中的物件，除了要算出路径外，还需要算出每个时间点的运动信息。根据对未来物件的预测，明确的找出各个时间点物件的运动信息



横向位移

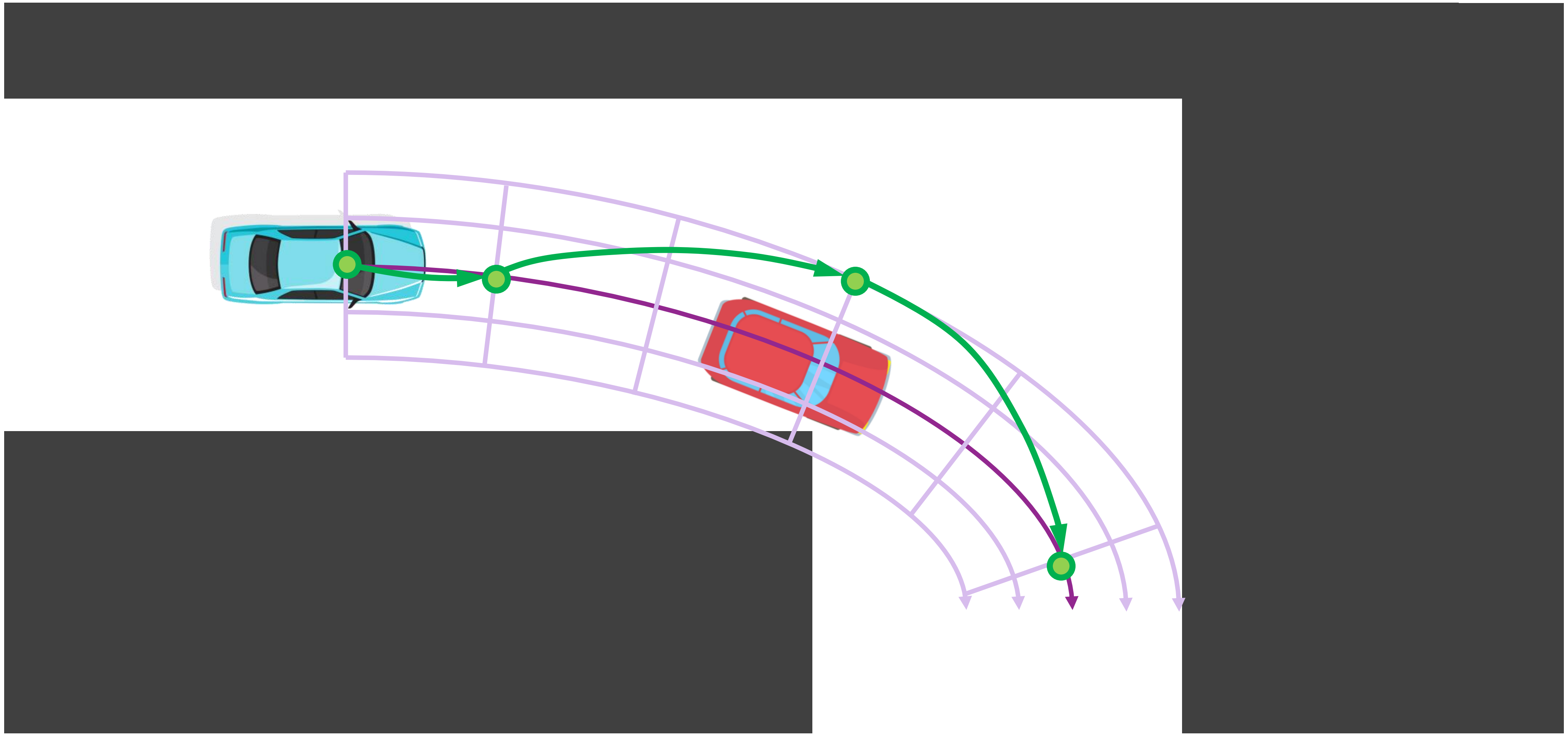
轨迹采样，找到cost最小的

等速前进

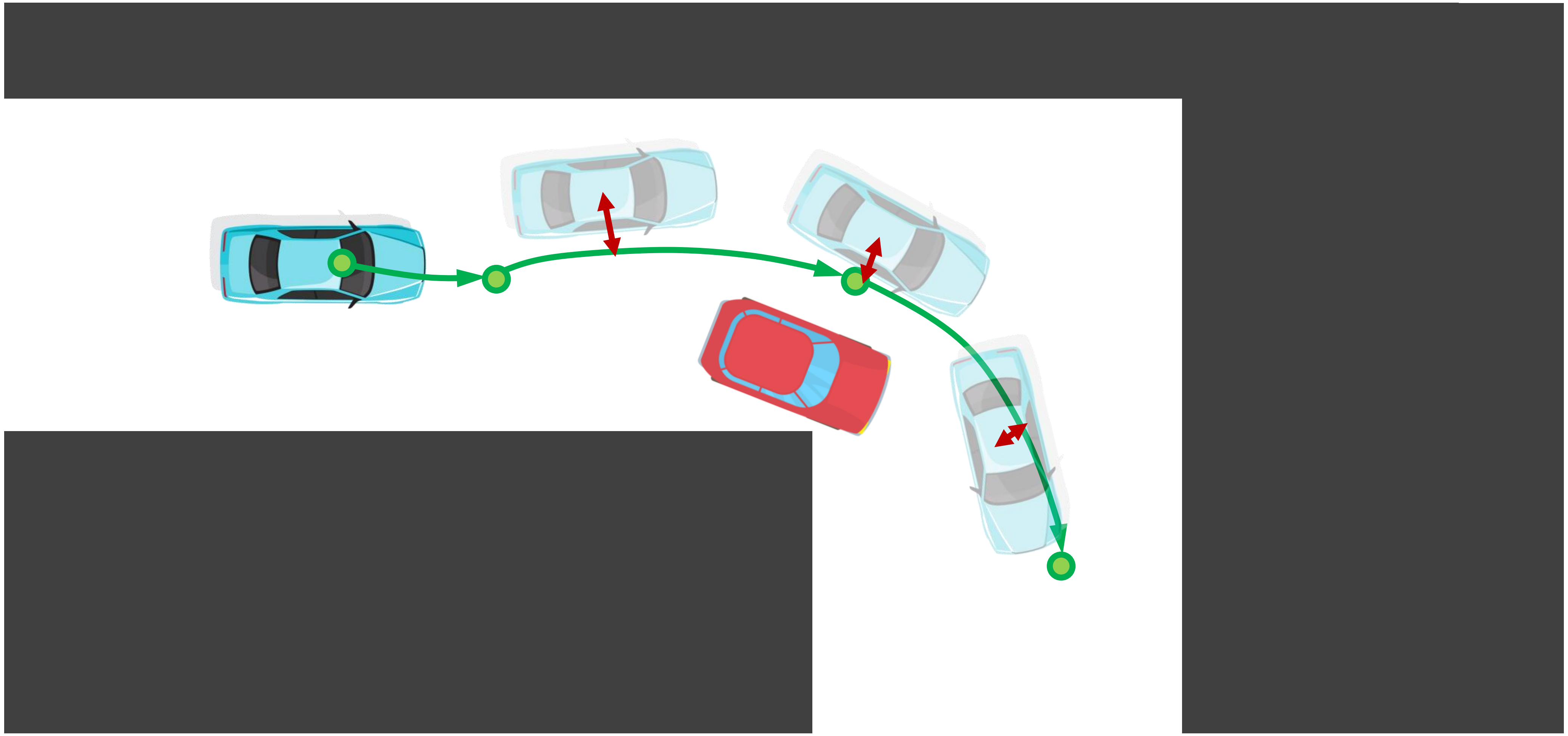
State Lattices Planning

用图像检测找到障碍物的bondingbox。最对它过去的轨迹进行回归，就可以得到未来轨迹的预测

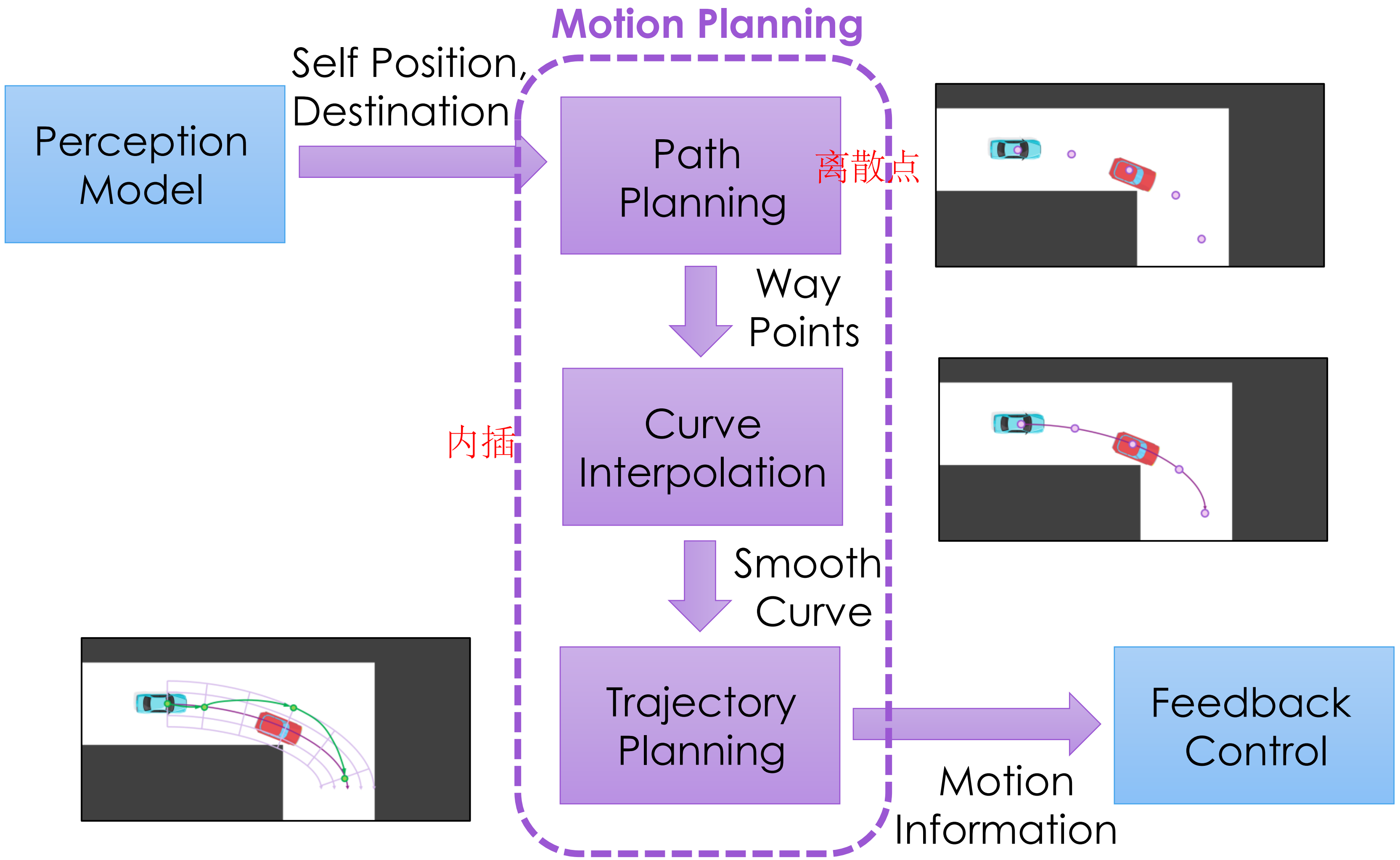
Trajectory Planning



Feedback Control



Motion Planning Flow



Path vs. Trajectory

- **Motion Planning = Path Planning + Trajectory Planning**

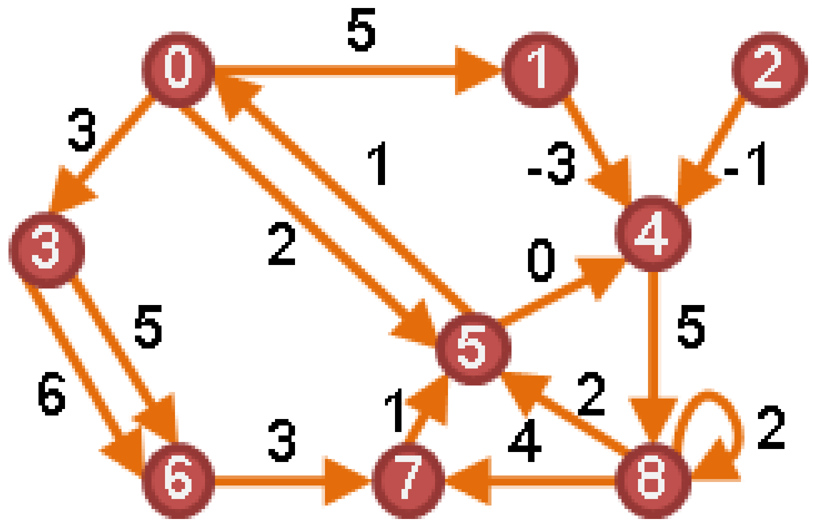
- Path Planning (Only consider the spatial information) 只有空间信息
  - **Input:** Start/End Position $\{[x_s, y_s], [x_e, y_e]\}$
  - **Output:** Way Points $\{[x_s, y_s], [x_1, y_1], [x_2, y_2], \dots, [x_e, y_e]\}$ 离散点

- Trajectory Planning (Consider both the spatial and temporal information)
  - **Inputs:** Start/End Position $\{[x_s, y_s], [x_e, y_e]\}$, Curve Function $f(x)$, Time Density Δt 时间区间
  - **Outputs:** Motion information with time
    $\{[x_0, y_0, v_0, a_0, t_0], [x_1, y_1, v_1, a_1, t_1], [x_2, y_2, v_2, a_2, t_2], \dots, [x_N, y_N, v_N, a_N, t_N]\}$

Path Planning

Graph Representation

- A graph G Consists of two sets, V and E
  - G = (V, E)
  - V : a set of Vertices
  - E : a set of Edges (pair of vertices)
- Direction
  - Undirected : (v1, v2) = (v2, v1)
  - Directed : <v1, v2> ≠ <v2, v1>
- Weight 代表cost
  - Graph may have weight on edges



Directed Graph with weight

有向图

Shortest Path Problem

- Find a path between two vertices such that
  - Sum of the weights of its constituent edges is minimized
- Single-source shortest paths
  - Dijkstra : Non-negative weights graphs 保证可以找到最佳解
  - Best-First Search (BFS) : Heuristic Greedy Search 速度快，但不一定能找到最佳解
  - A* : BFS with Dijkstra

三种解法

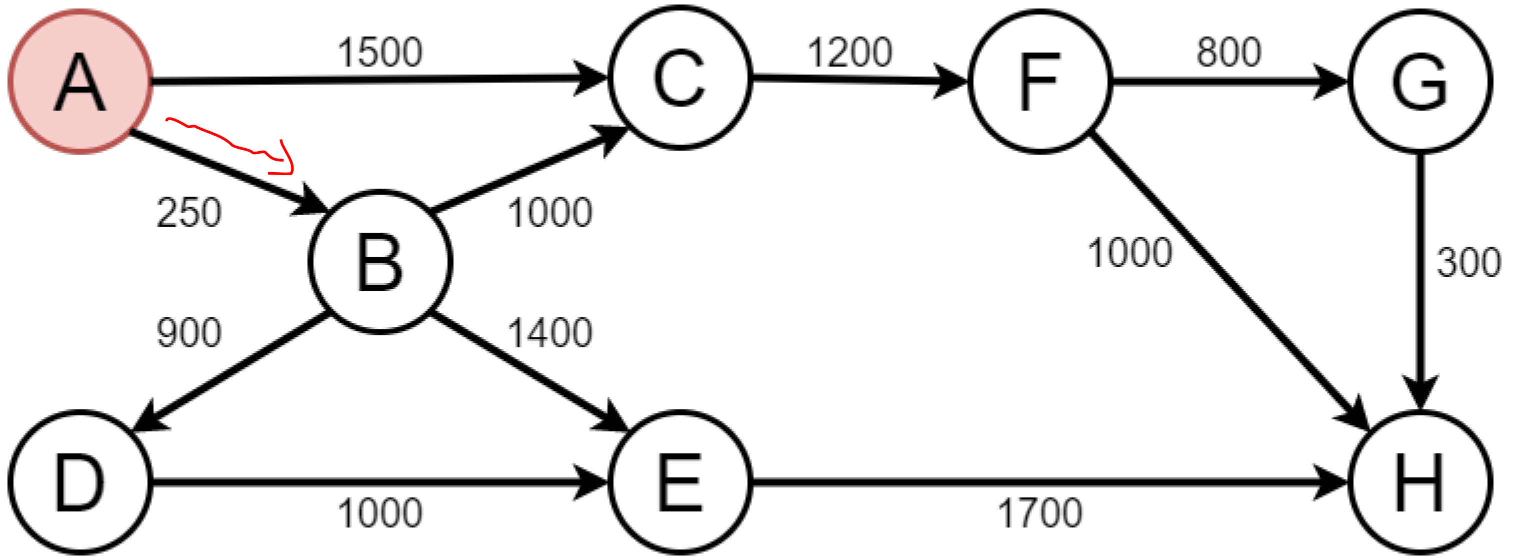
Dijkstra’s Algorithm

- Solves the single-source shortest path problem
  - Only with non-negative edge weight

每一条最短路径都是边数更少，权重更小的　　从边数最少，权重最小的开始建立

- From starting point v
  - Select a minimal distance and unfinished vertex u
  - Update other unfinished vertices v’ using the equation:
    - $d(v, v') = \min(\underline{d(v, u) + \langle u, v' \rangle}, \underline{d(v, v')})$

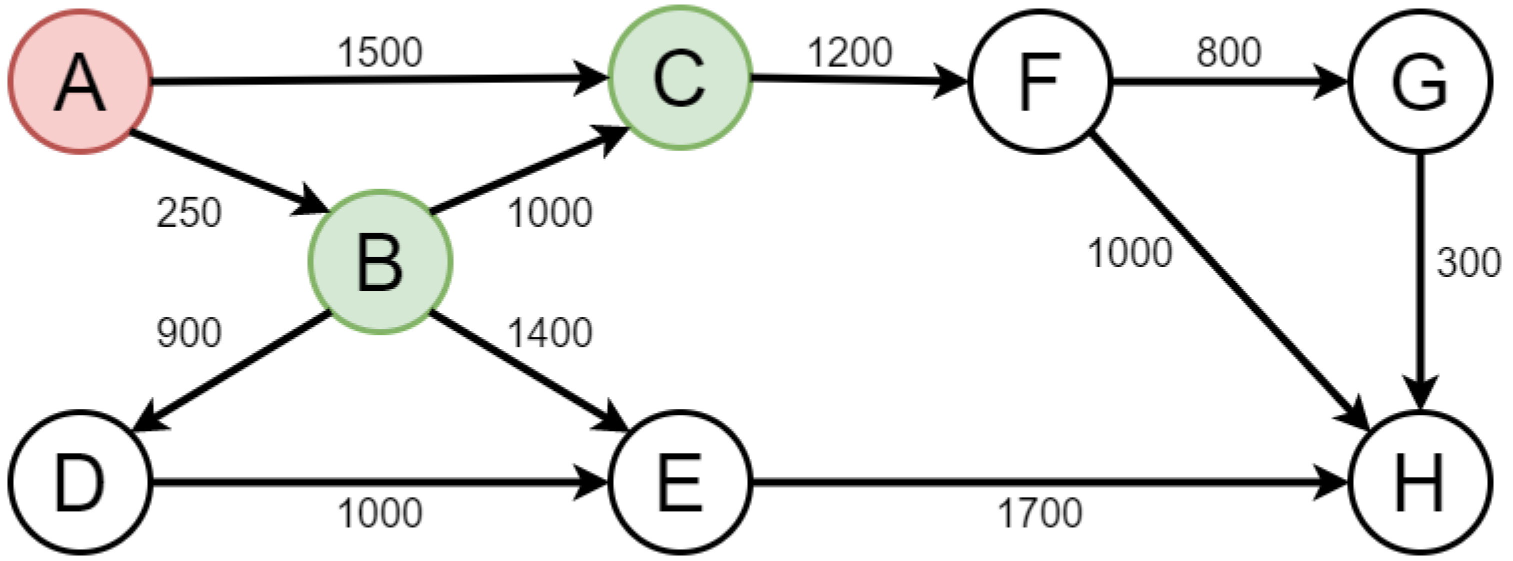
Dijkstra’s Algorithm : Example



	B	C	D	E	F	G	H
From A	∞	∞	∞	∞	∞	∞	∞

$A \rightarrow A$ Finished = {A} = 0

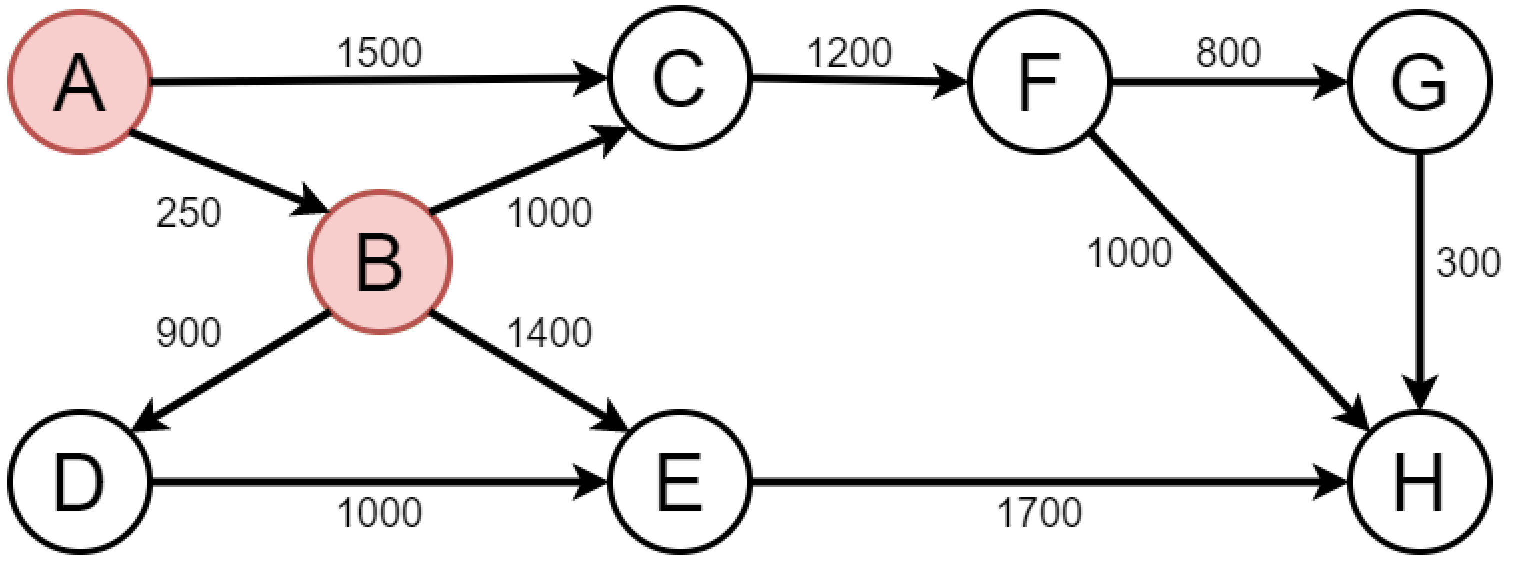
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1500	∞	∞	∞	∞	∞

Finished = {A}

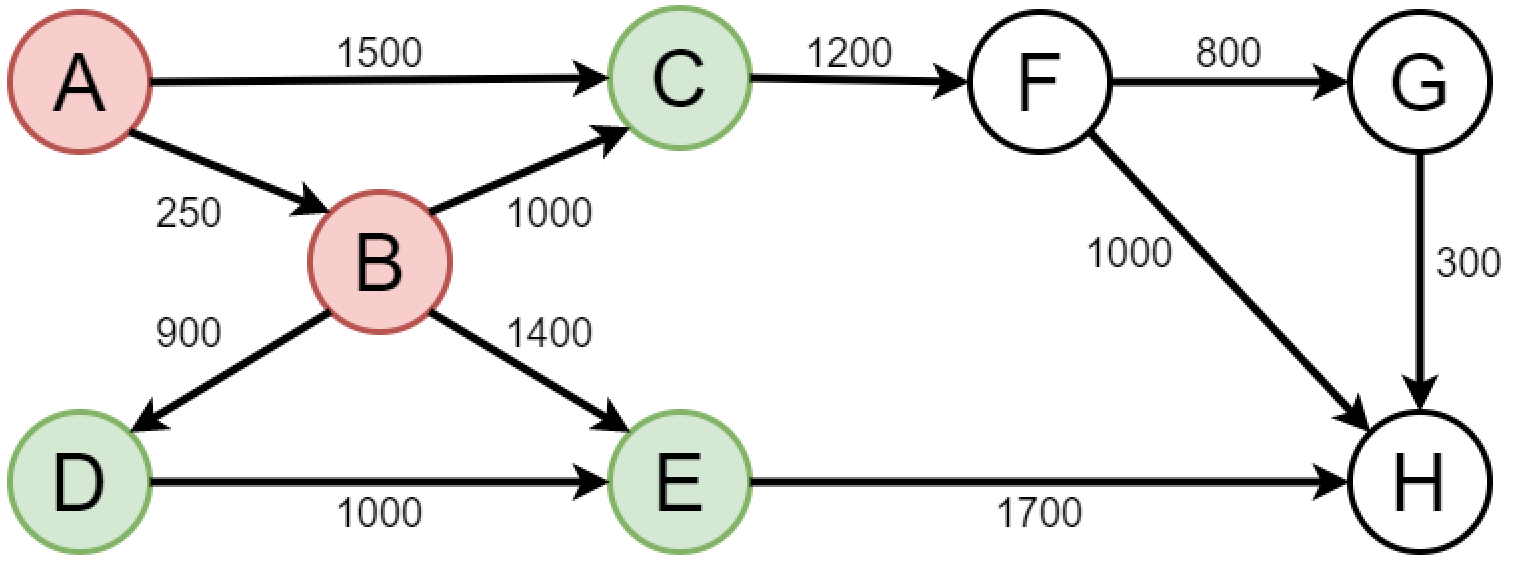
Dijkstra’s Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1500	∞	∞	∞	∞	∞

Finished = {A, B}

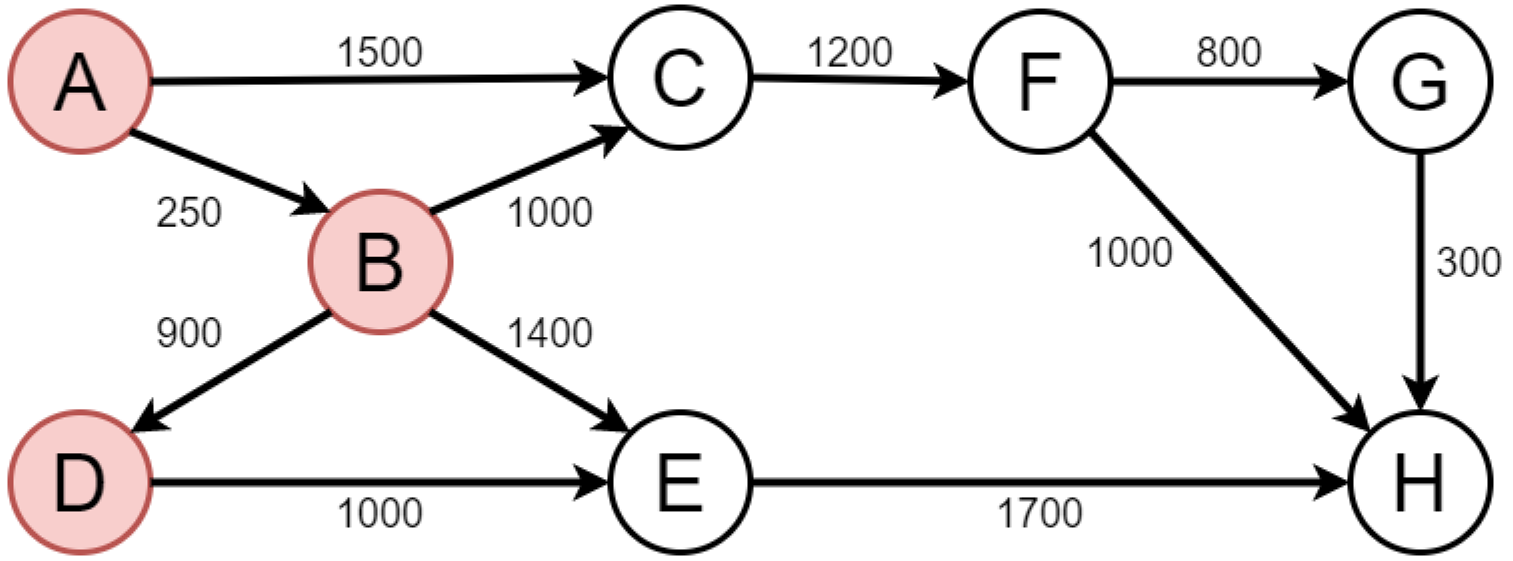
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	∞	∞	∞

Finished = {A, B}

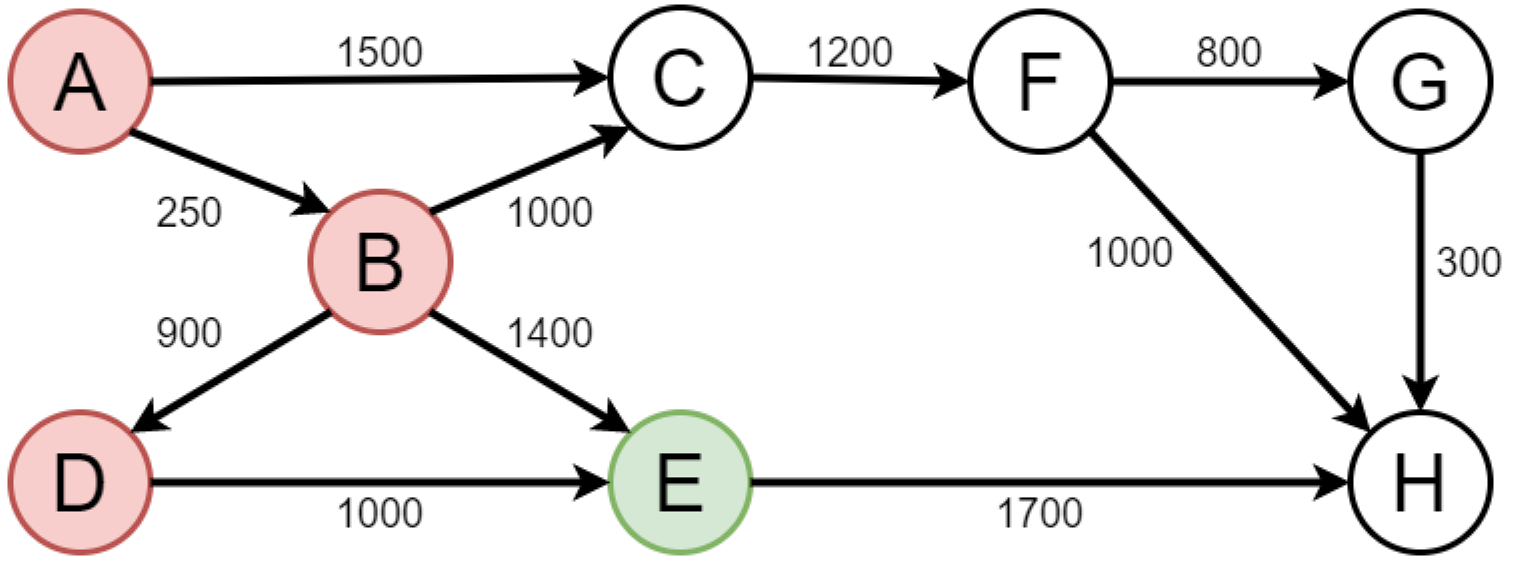
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	∞	∞	∞

Finished = {A, B, D}

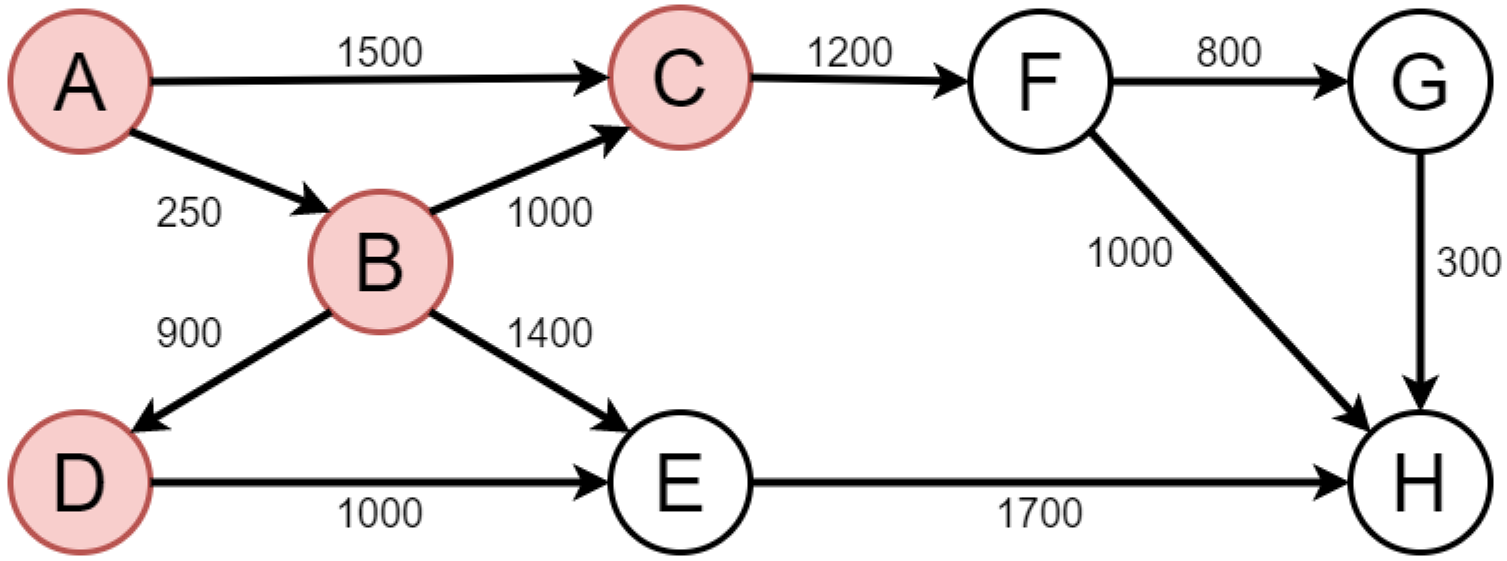
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	∞	∞	∞

Finished = {A, B, D}

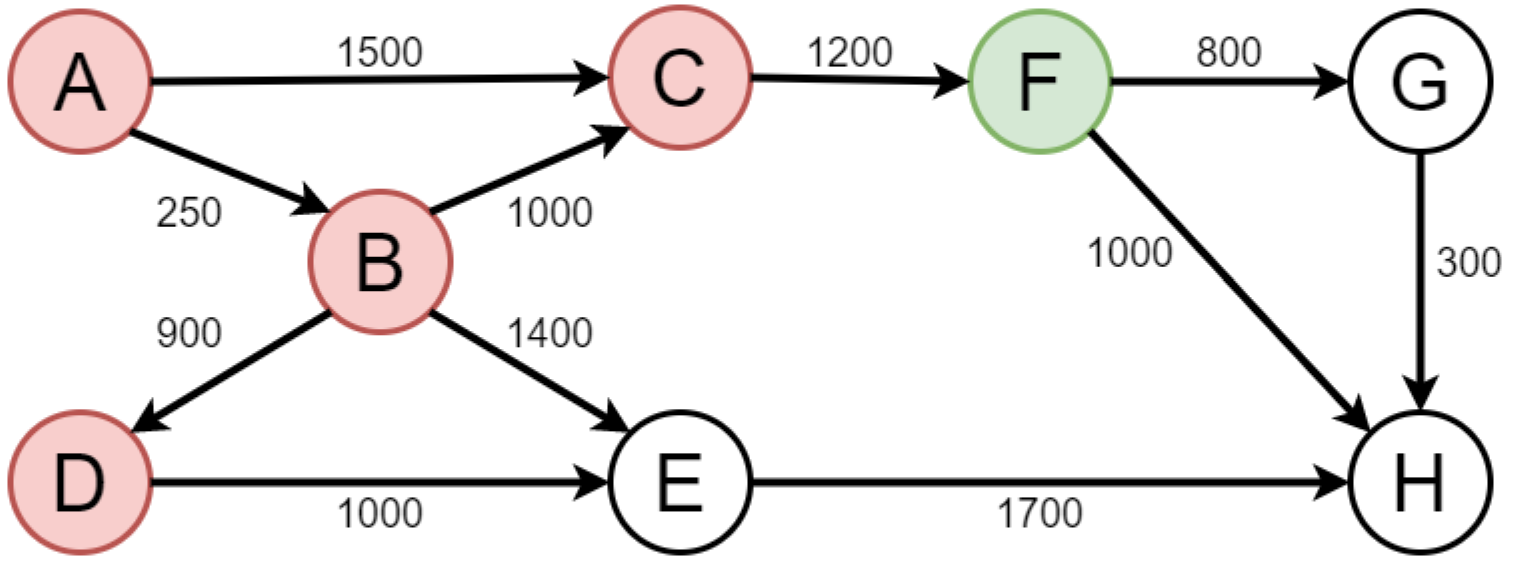
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	∞	∞	∞

Finished = {A, B, D, C}

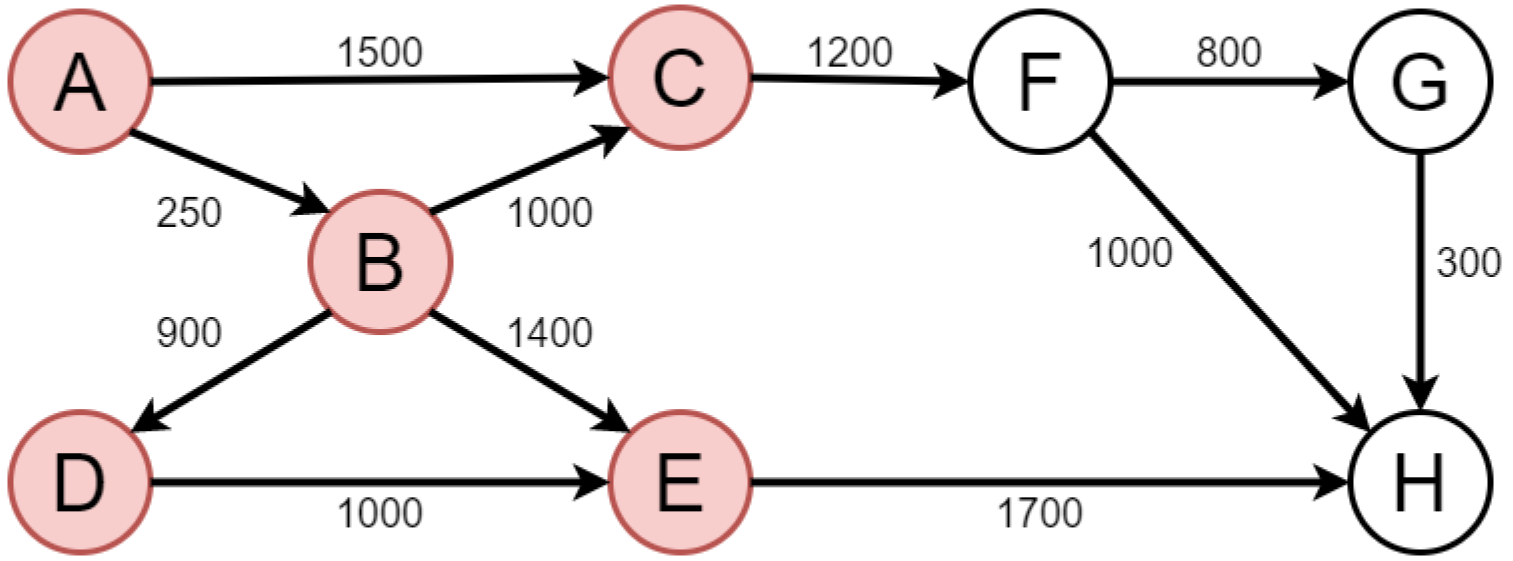
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	∞	∞

Finished = {A, B, D, C}

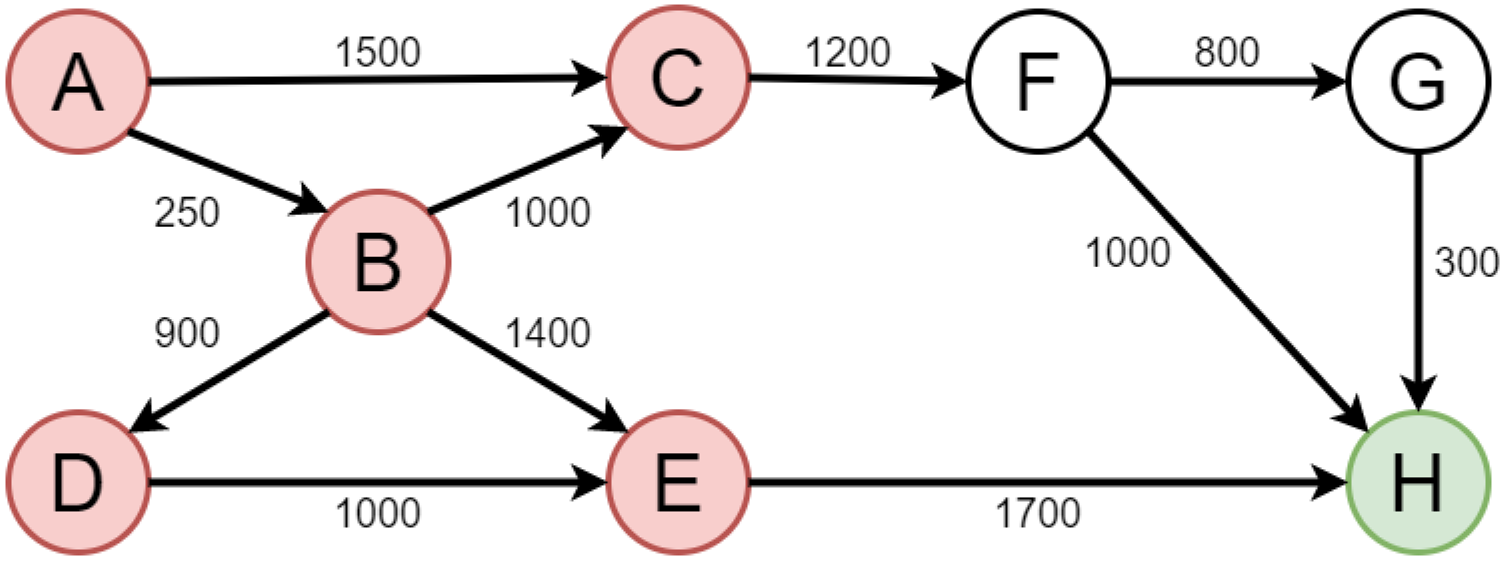
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	∞	∞

Finished = {A, B, D, C, E}

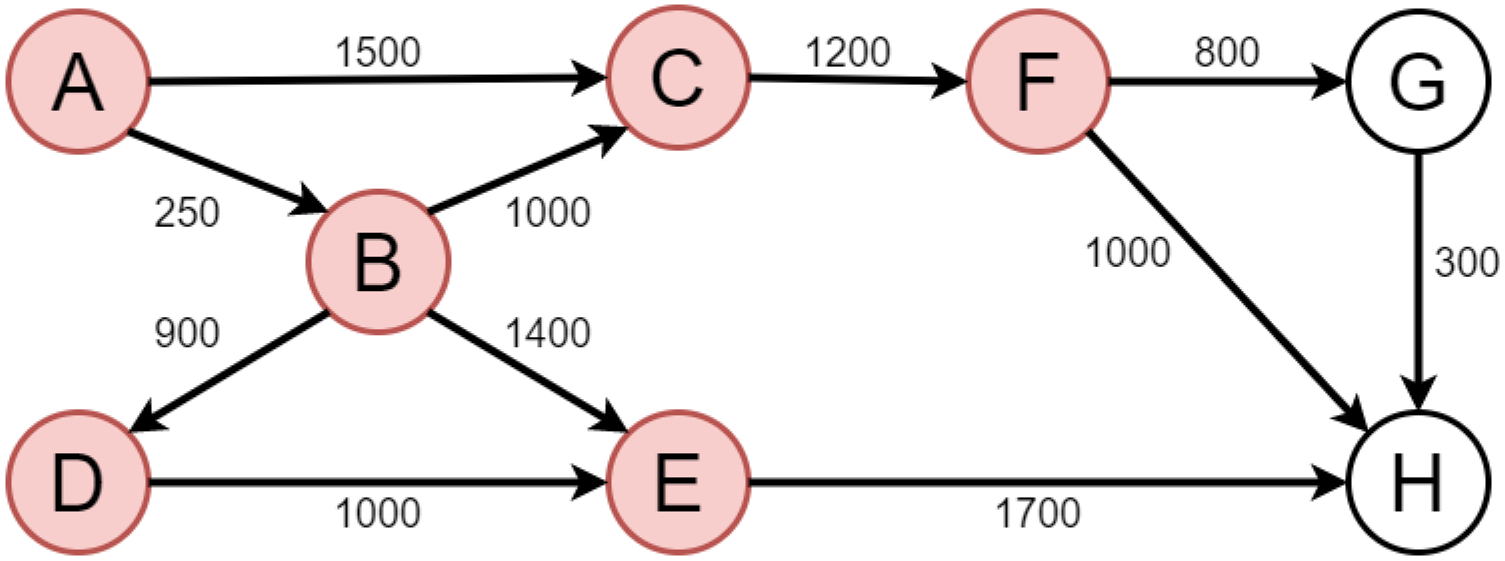
Dijkstra’s Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	∞	3350

Finished = {A, B, D, C, E}

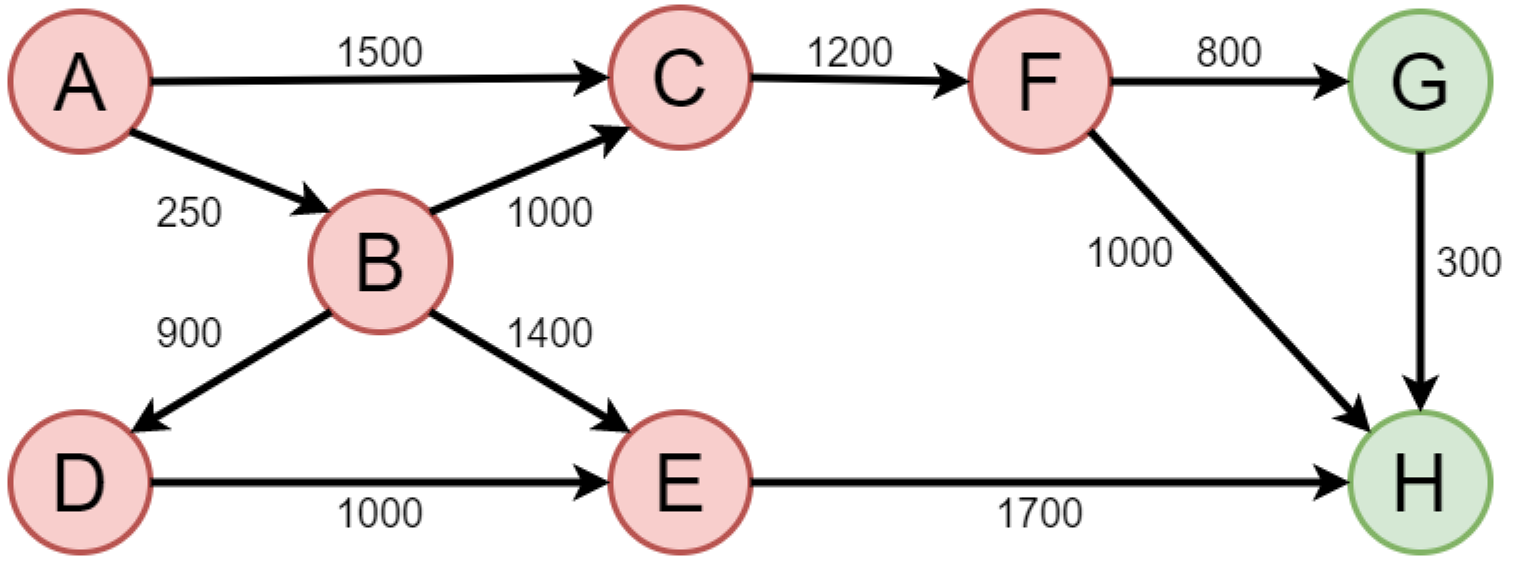
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	∞	3350

Finished = {A, B, D, C, E, F}

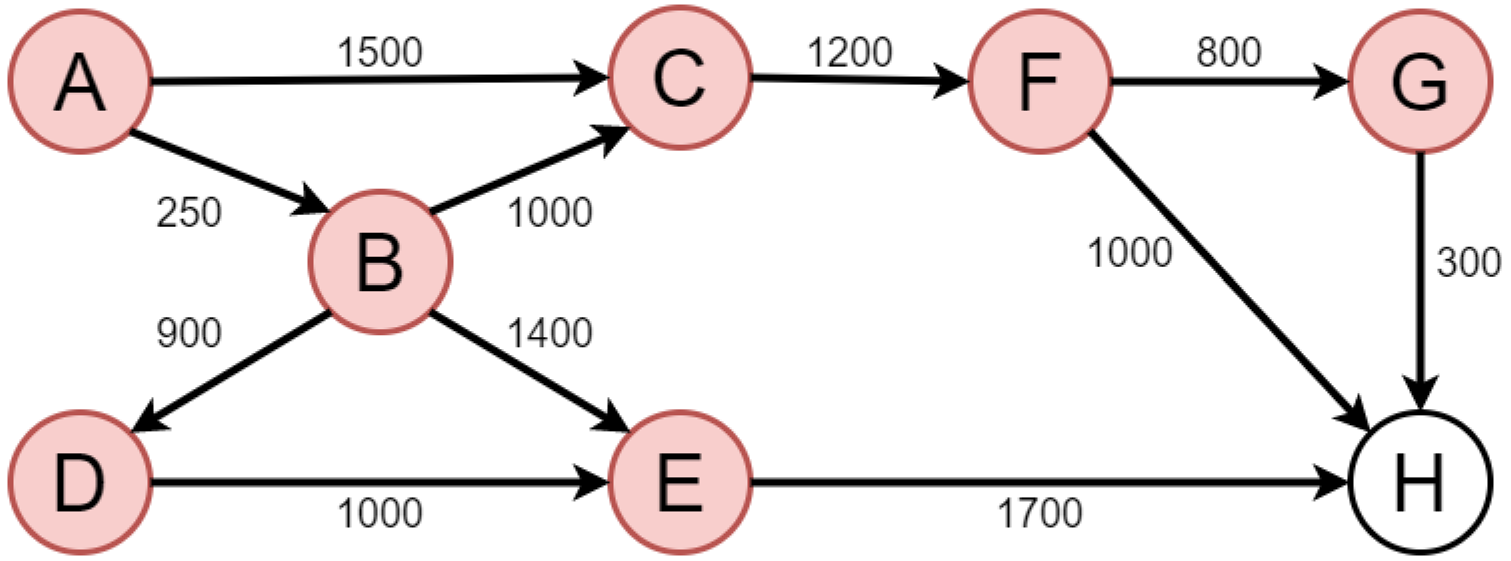
Dijkstra’s Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	3250	3350

Finished = {A, B, D, C, E, F}

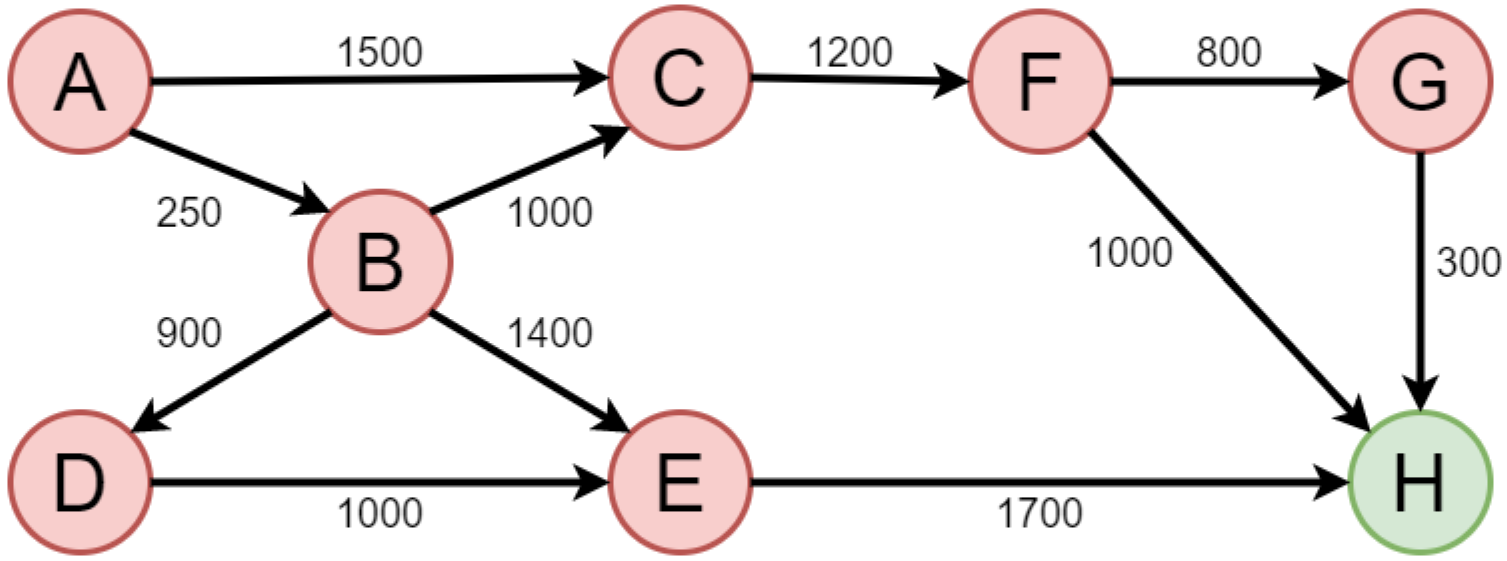
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	3250	3350

Finished = {A, B, D, C, E, F, G}

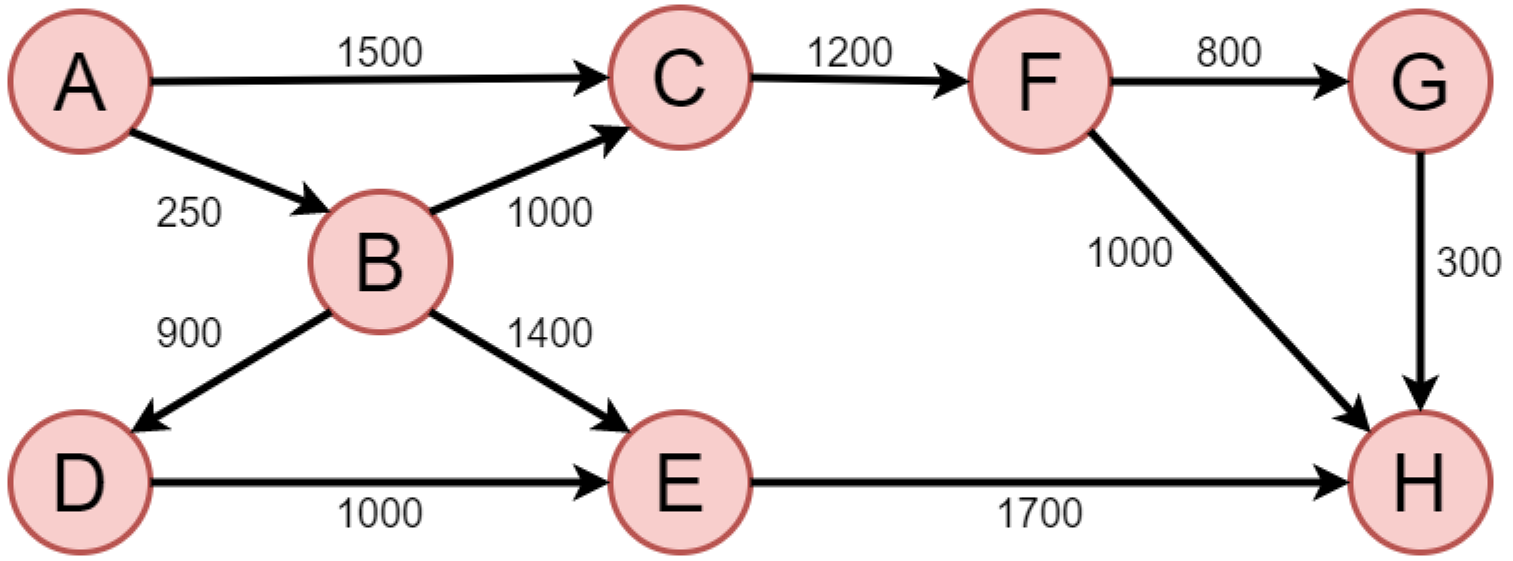
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	3250	3350

Finished = {A, B, D, C, E, F, G}

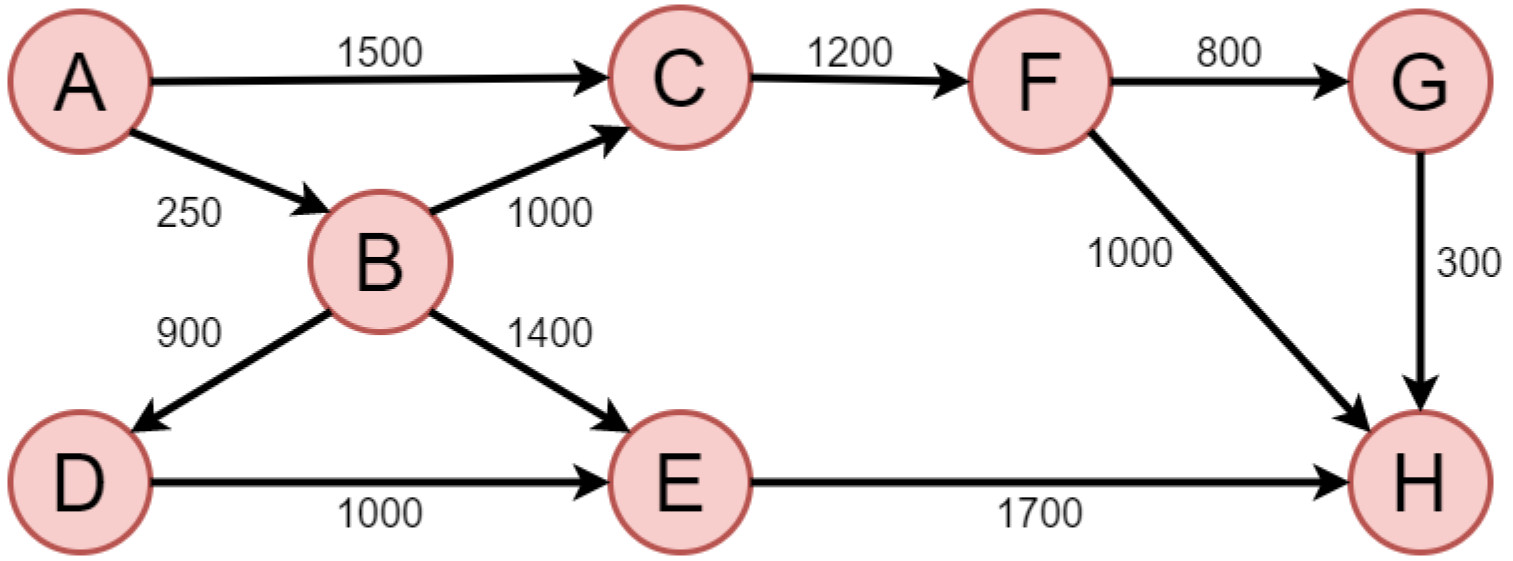
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	3250	3350

Finished = {A, B, D, C, E, F, G, H}

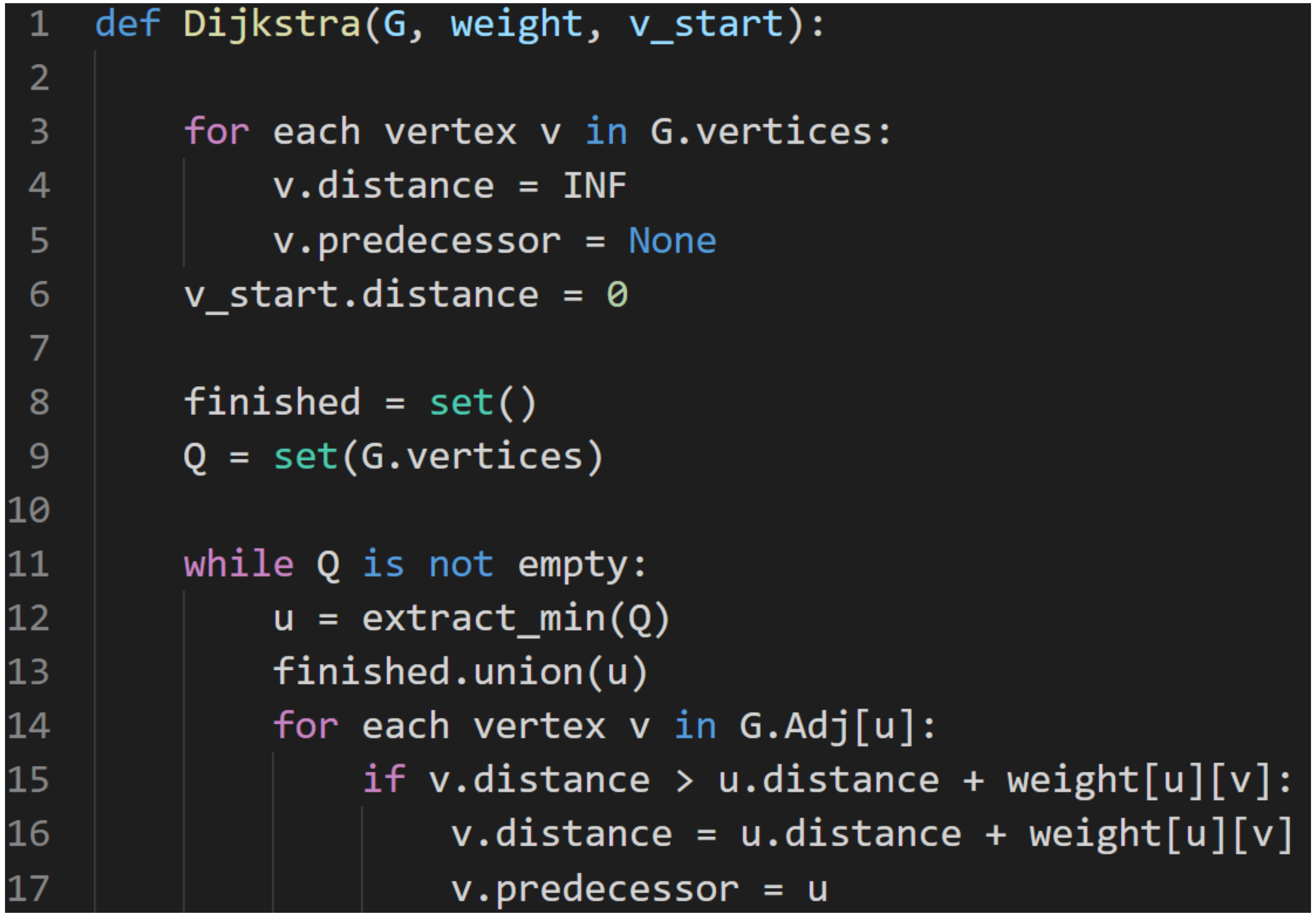
Dijkstra's Algorithm : Example



	B	C	D	E	F	G	H
From A	250	1250	1150	1650	2450	3250	3350

Finished = {A, B, D, C, E, F, G, H}

Dijkstra’s Algorithm : Pseudo Code



```
def Dijkstra(G, weight, v_start):

    for each vertex v in G.vertices:
        v.distance = INF
        v.predecessor = None
    v_start.distance = 0

    finished = set()
    Q = set(G.vertices)

    while Q is not empty:
        u = extract_min(Q)
        finished.union(u)
        for each vertex v in G.Adj[u]:
            if v.distance > u.distance + weight[u][v]:
                v.distance = u.distance + weight[u][v]
                v.predecessor = u
```

Dijkstra's Algorithm : Analysis

节点数

- Time Complexity
  - Original Algorithm : O(V^2) 复杂度
  - Optimized (Fibonacci-Heap): O(E + VlogV)

边数

- Pros : Guarantees to find an optimal path
- Cons : Very slow when there are numerous vertices

Best-First Search

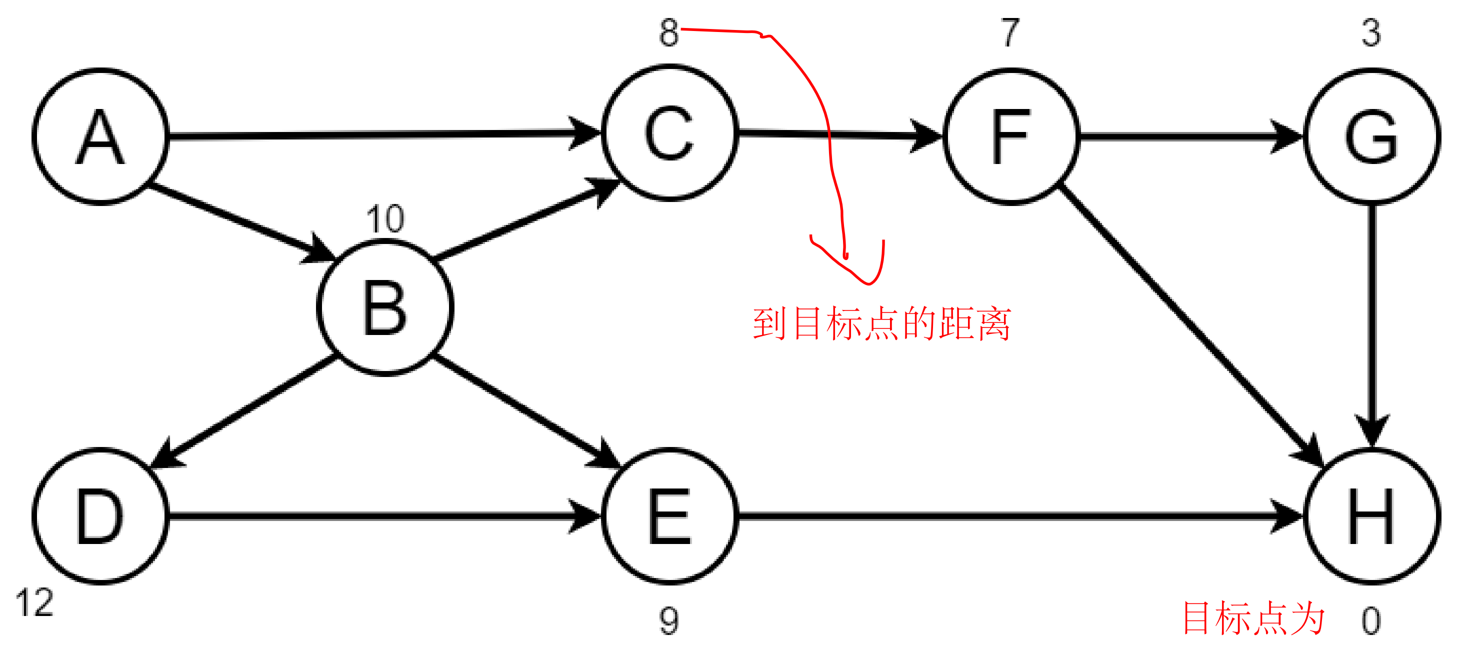
- Heuristic Search - explores a graph by expanding the most promising vertex

每个节点都定义一个 $\leftarrow$

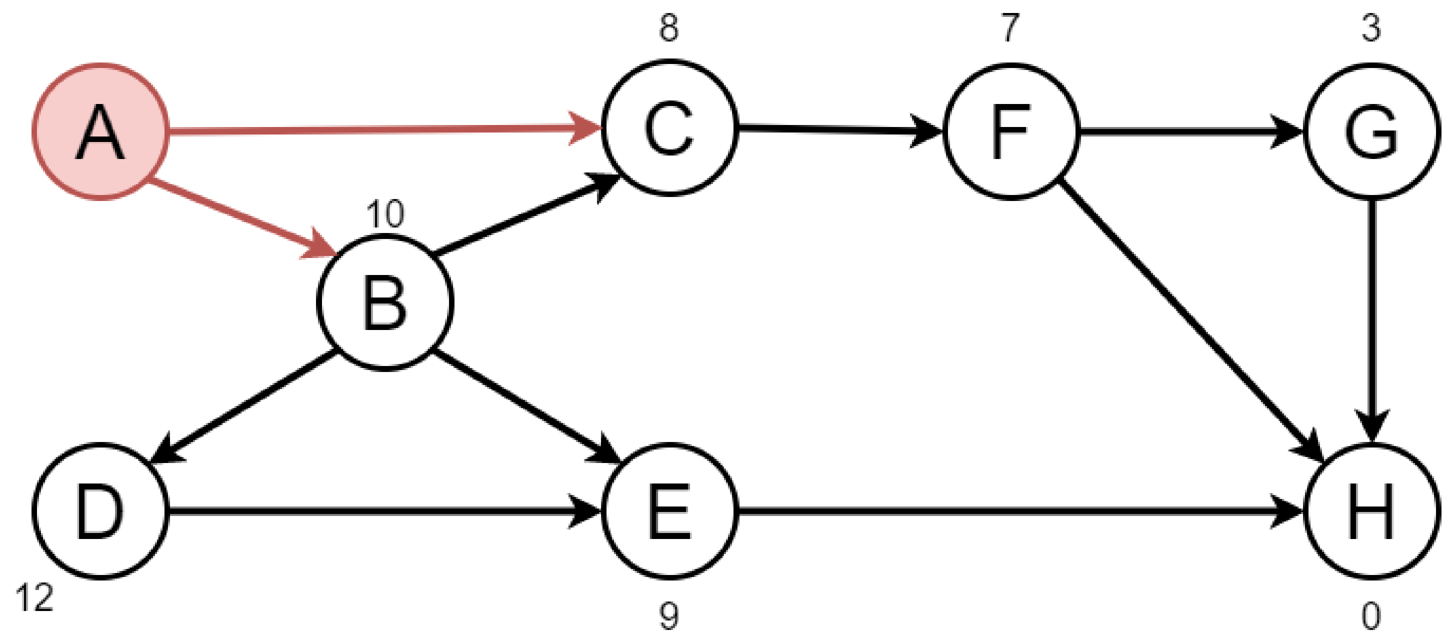
- Promise of a vertex - Heuristic Function f(v)
  - Estimation of the cheapest cost from v to the goal 最小cost预测值
  - e.g. Euclidean Distance, Manhattan Distance 用这两个去定义f（v）

Best-First Search : Example

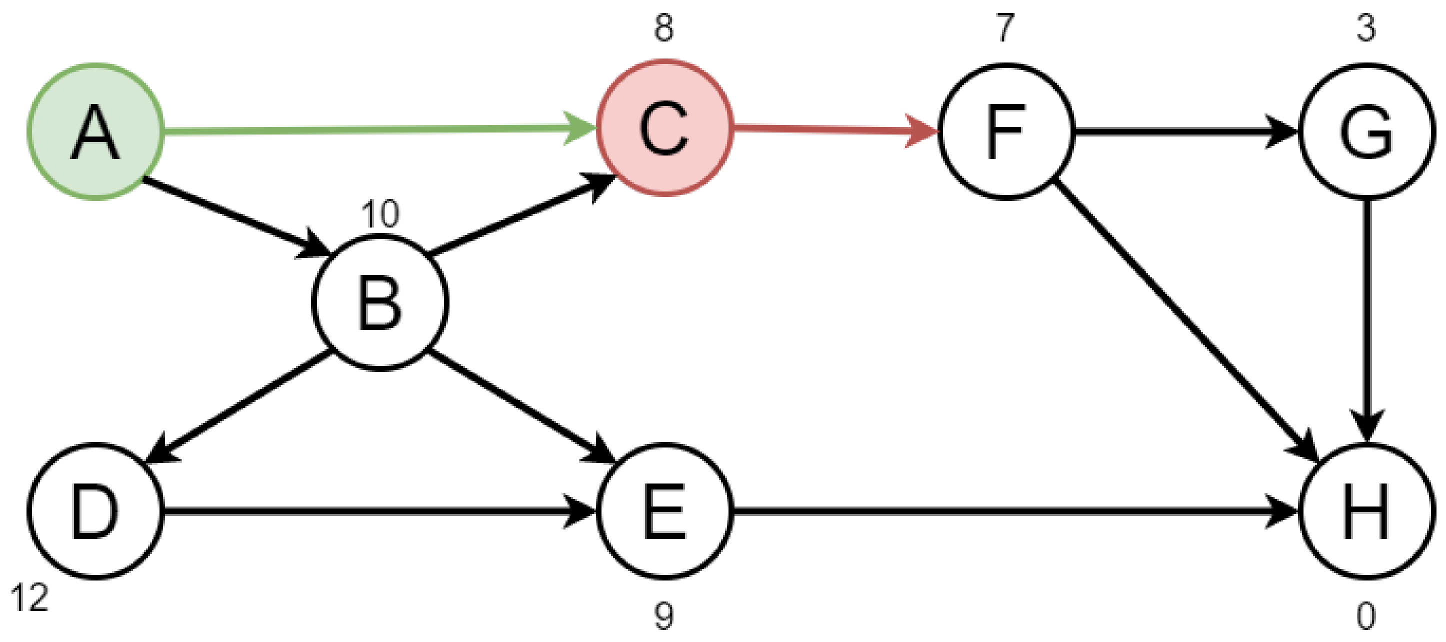
在每个节点上记录f(v)的值



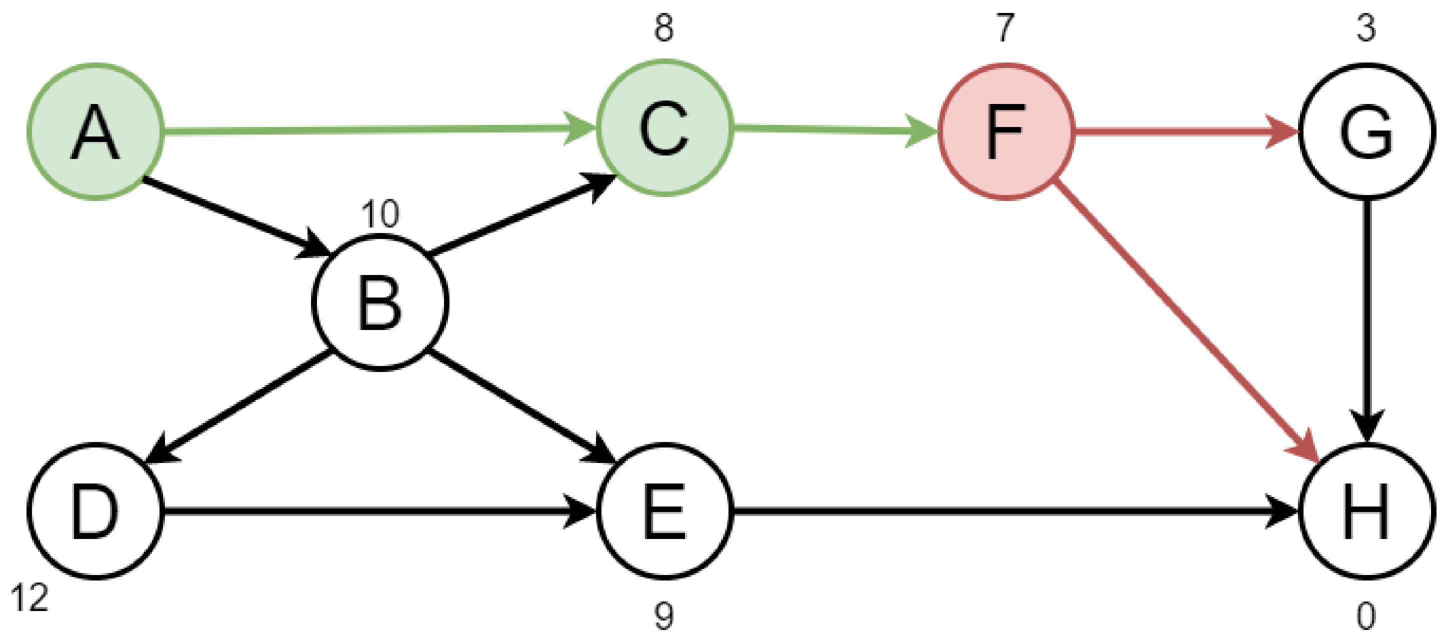
Best-First Search : Example



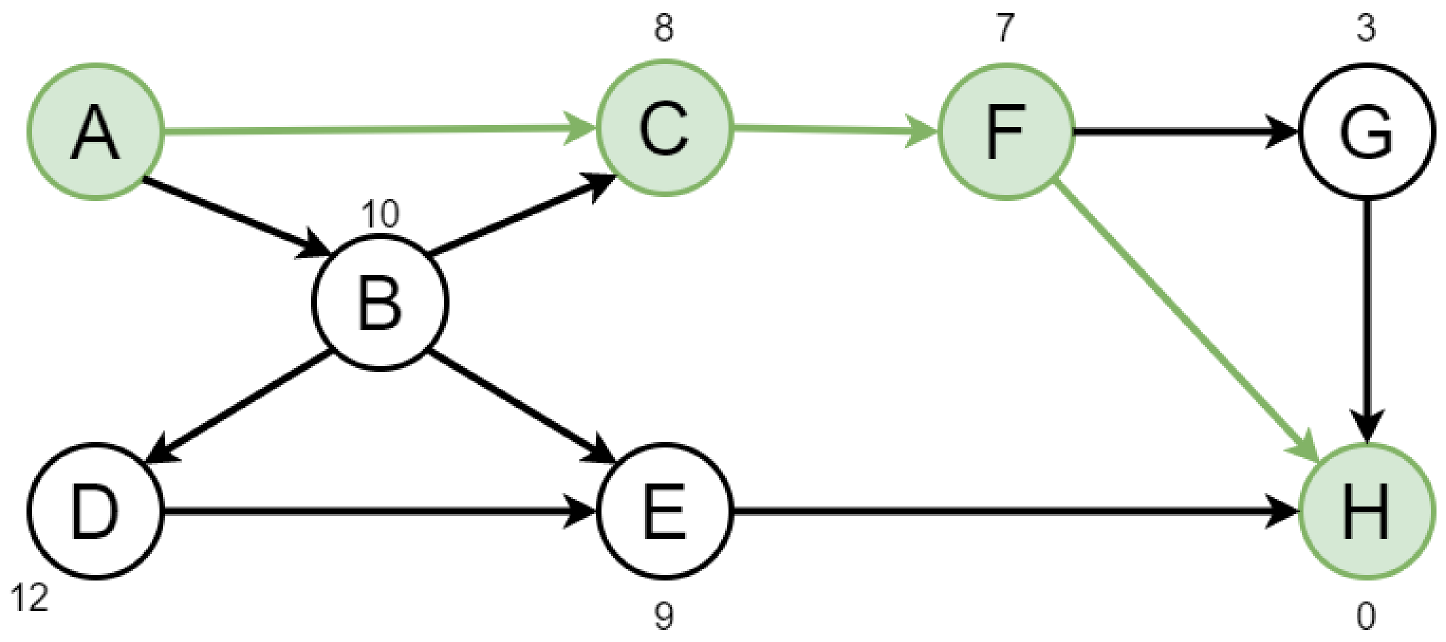
Best-First Search : Example



Best-First Search : Example



Best-First Search : Example



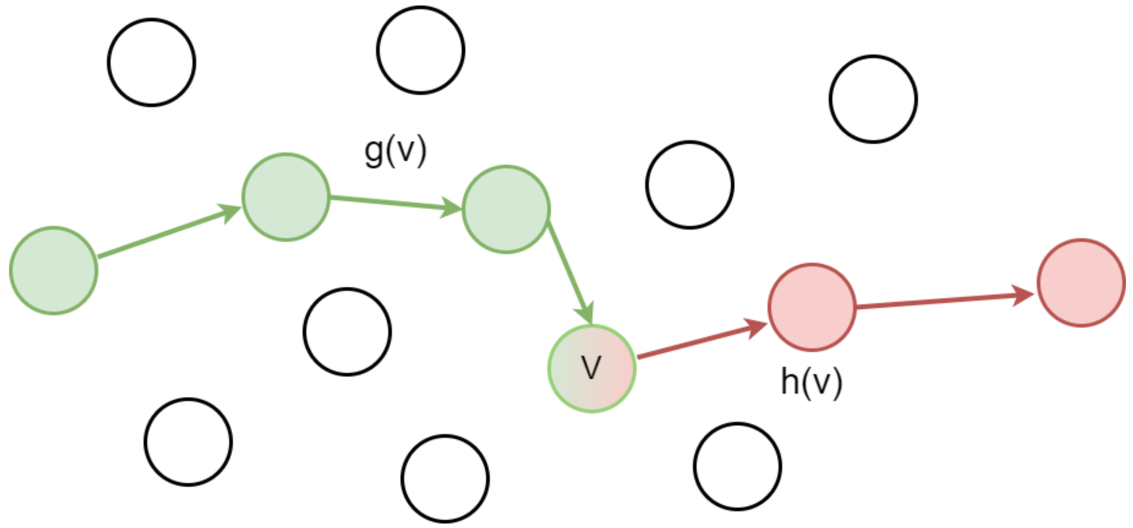
Best-First Search : Analysis

- Pros : Fast
- Cons : Heuristic estimation doesn’t guarantee the best path

f(v)是直觉定义的，不能保证就是最佳路径。

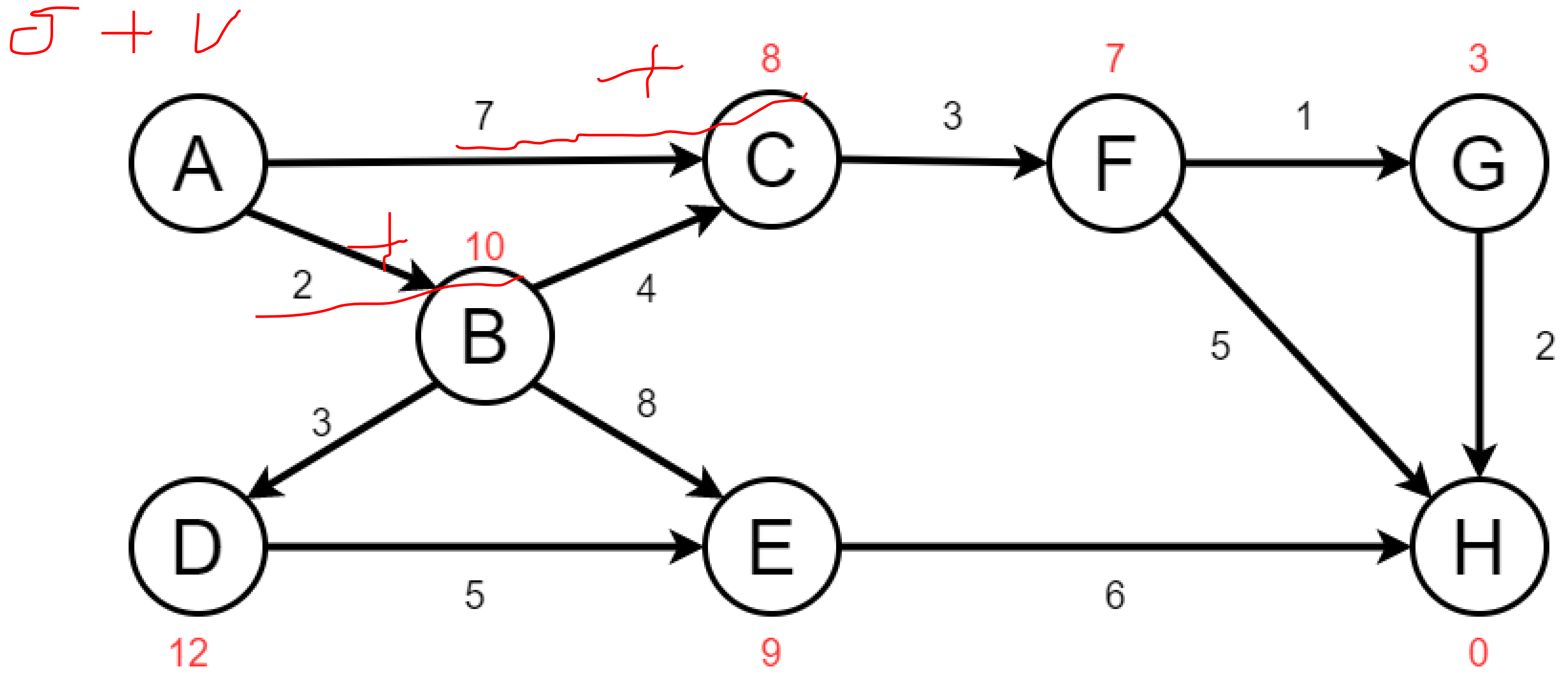
A* Algorithm

- Combine the advantages of both Dijkstra's and BFS

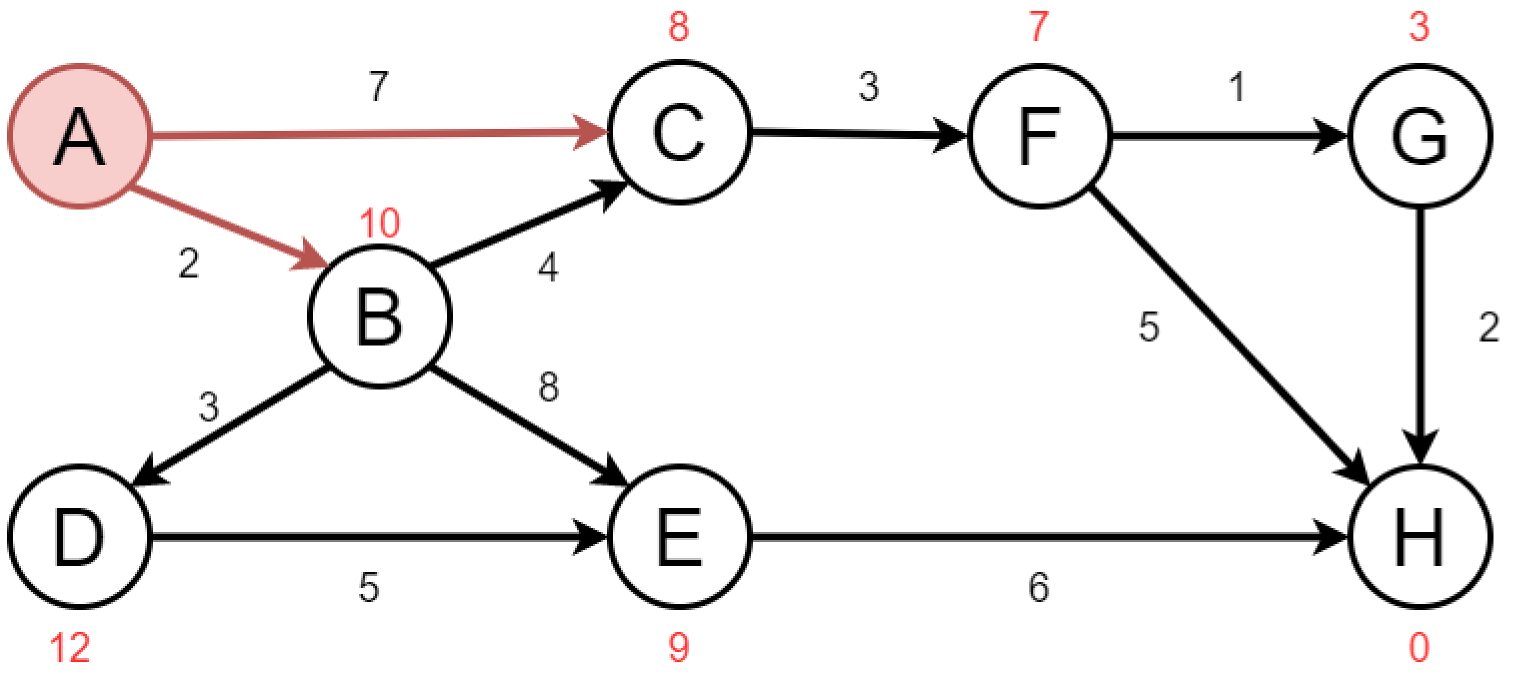


- Consider both previous and future parameters:
  - g(v) : calculated path cost from start to v (Dijkstra) 考虑了过去路径的最佳计算值
  - h(v) : heuristic path estimation from v to target (BFS) 也考虑了未来路径的预测值

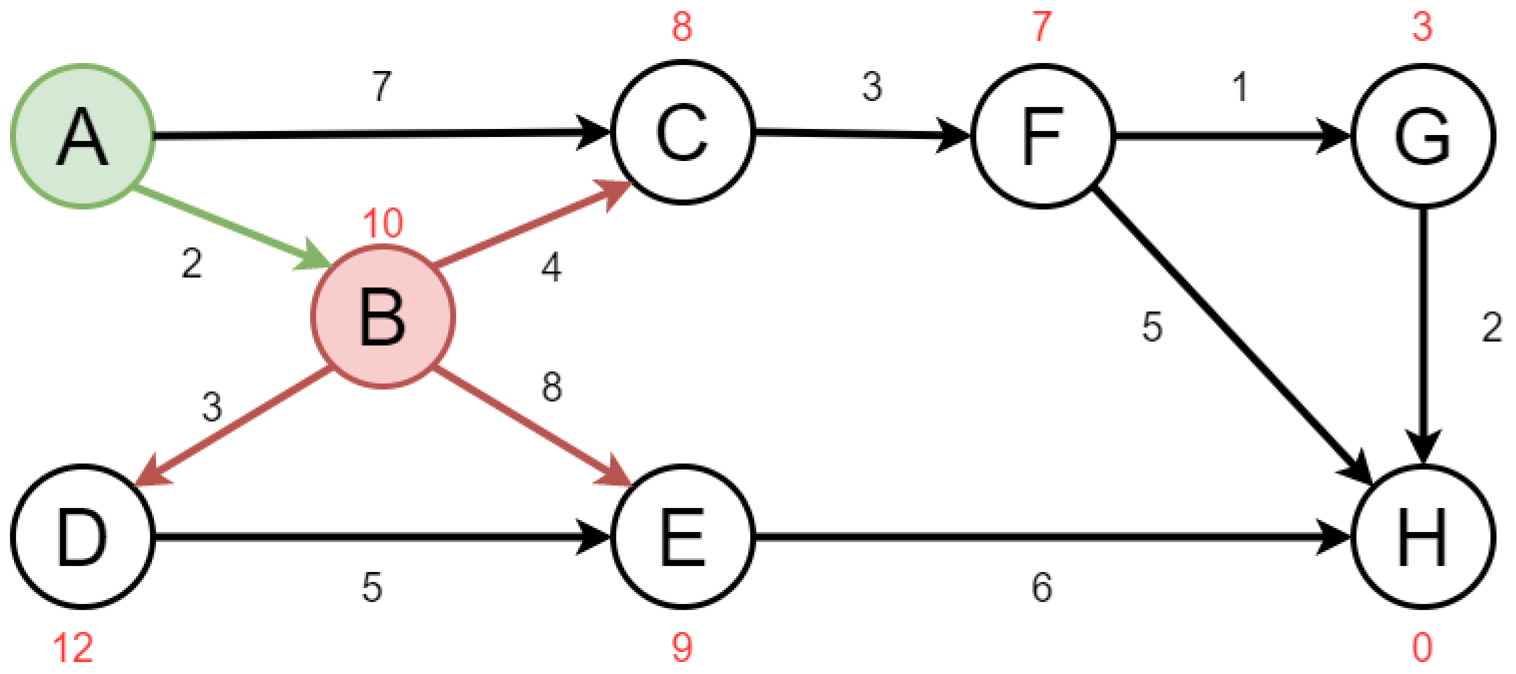
A* Algorithm : Example



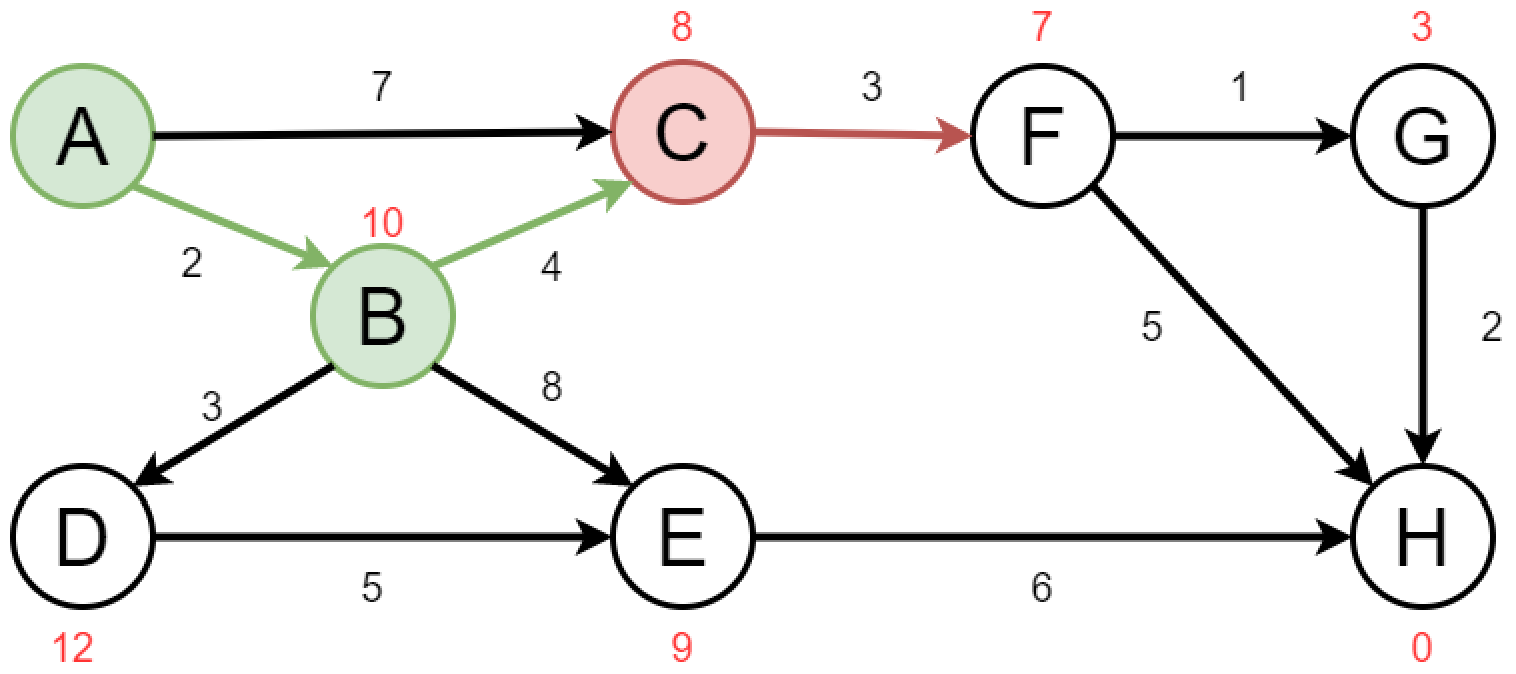
A* Algorithm : Example



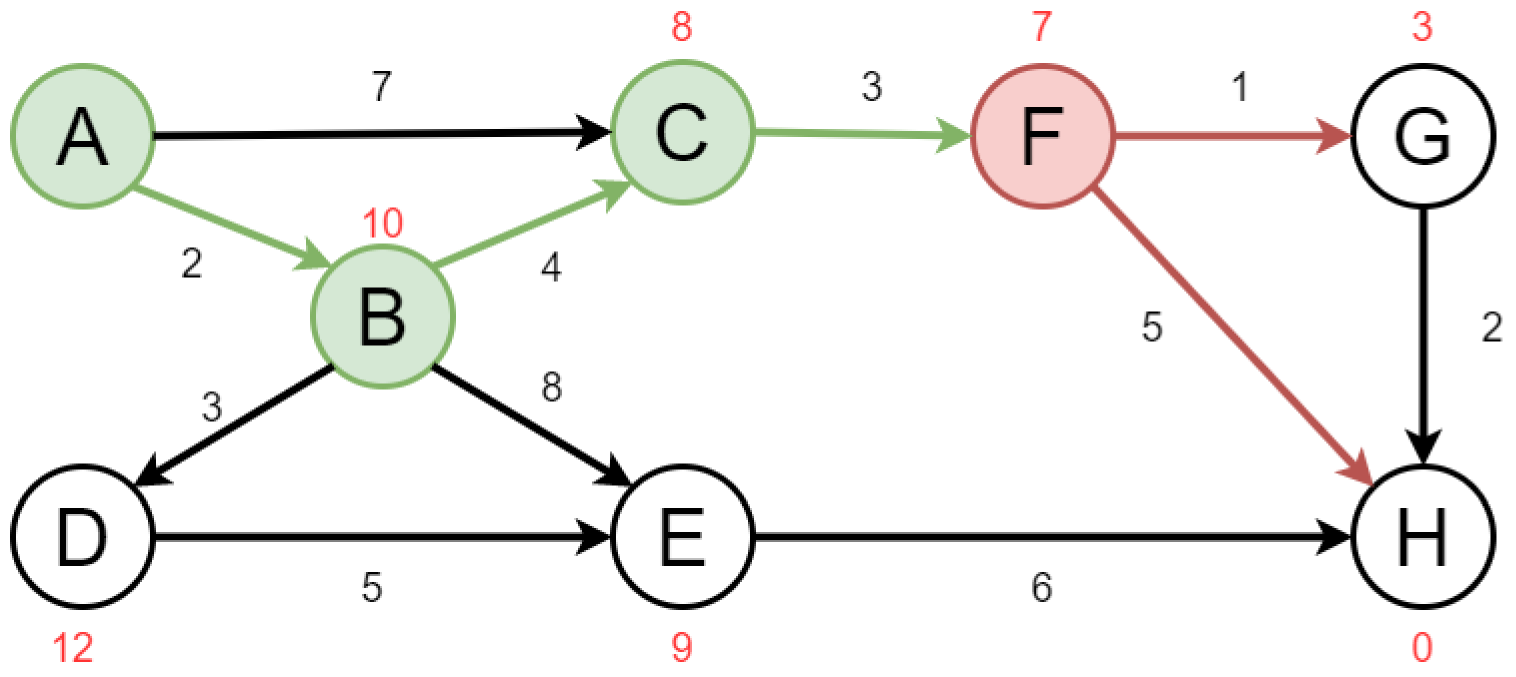
A* Algorithm : Example



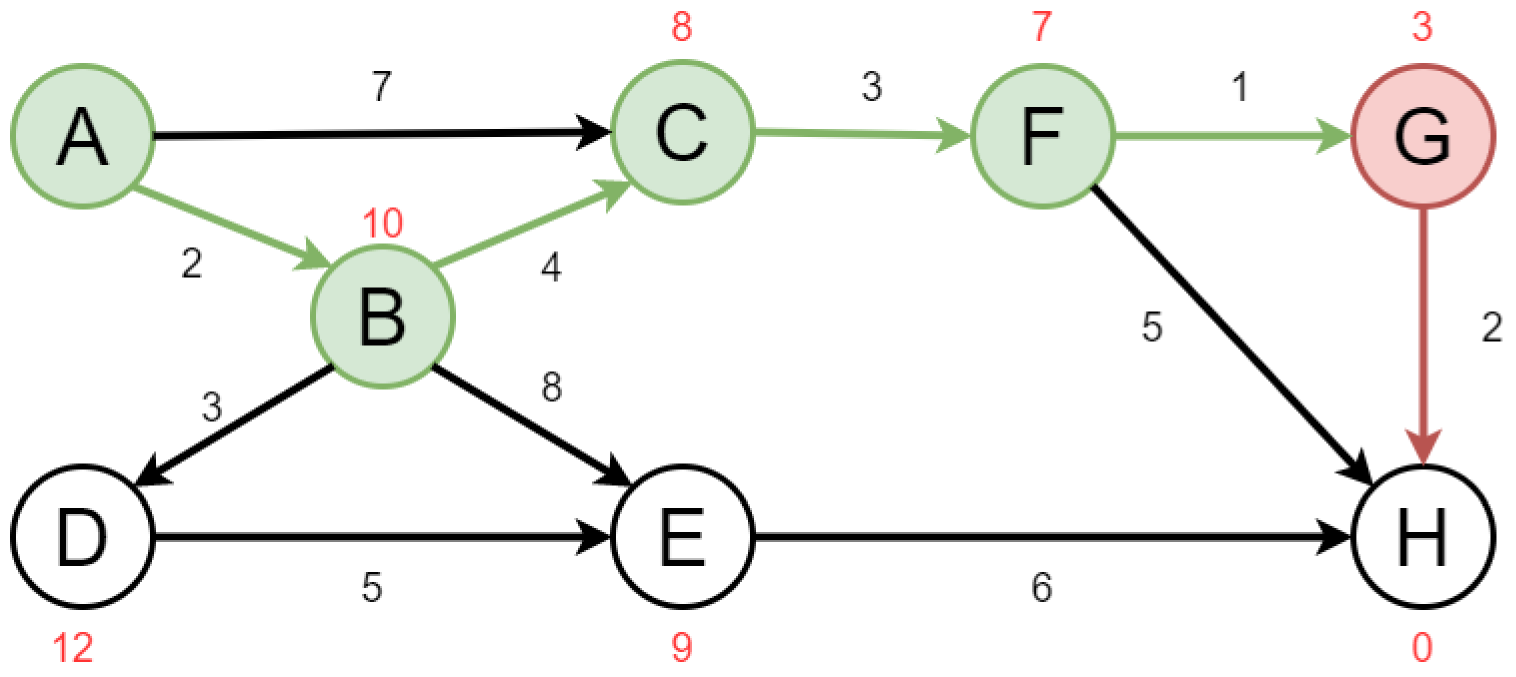
A* Algorithm : Example



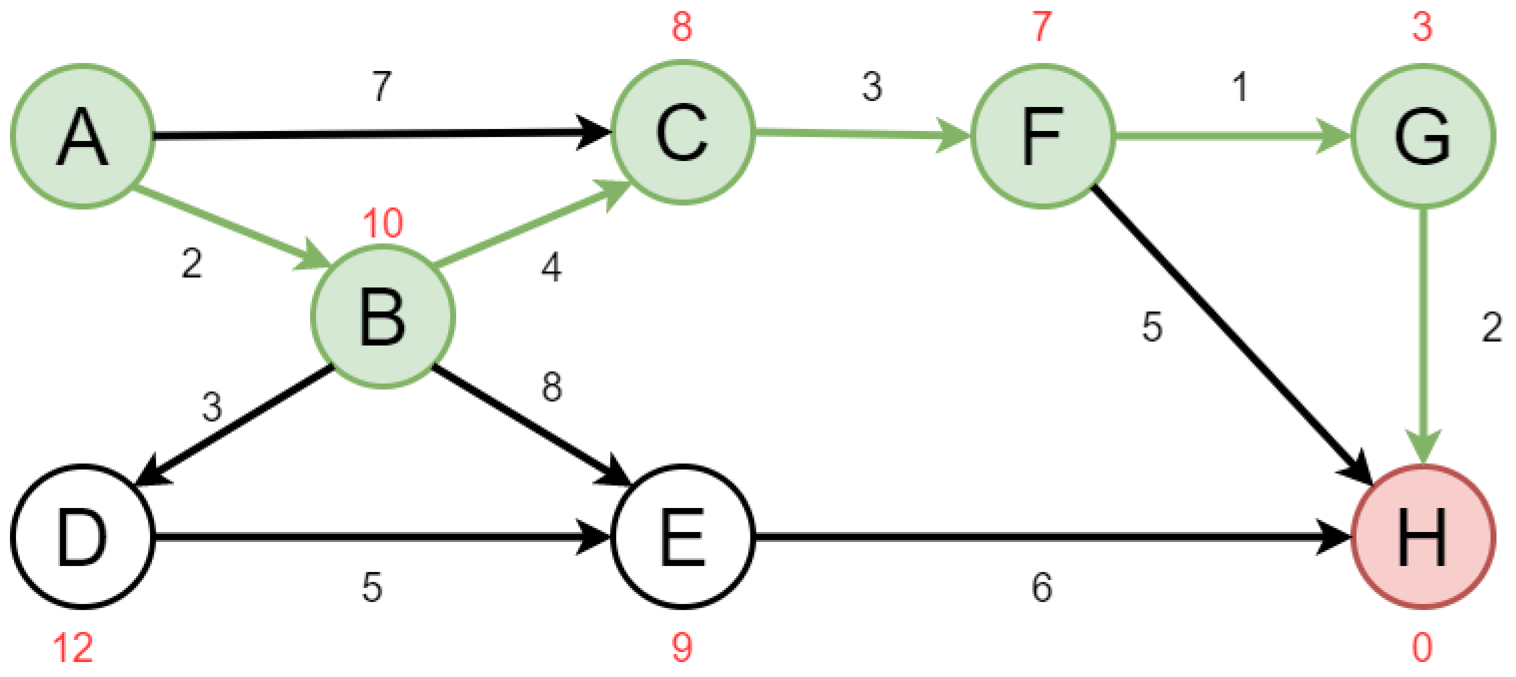
A* Algorithm : Example



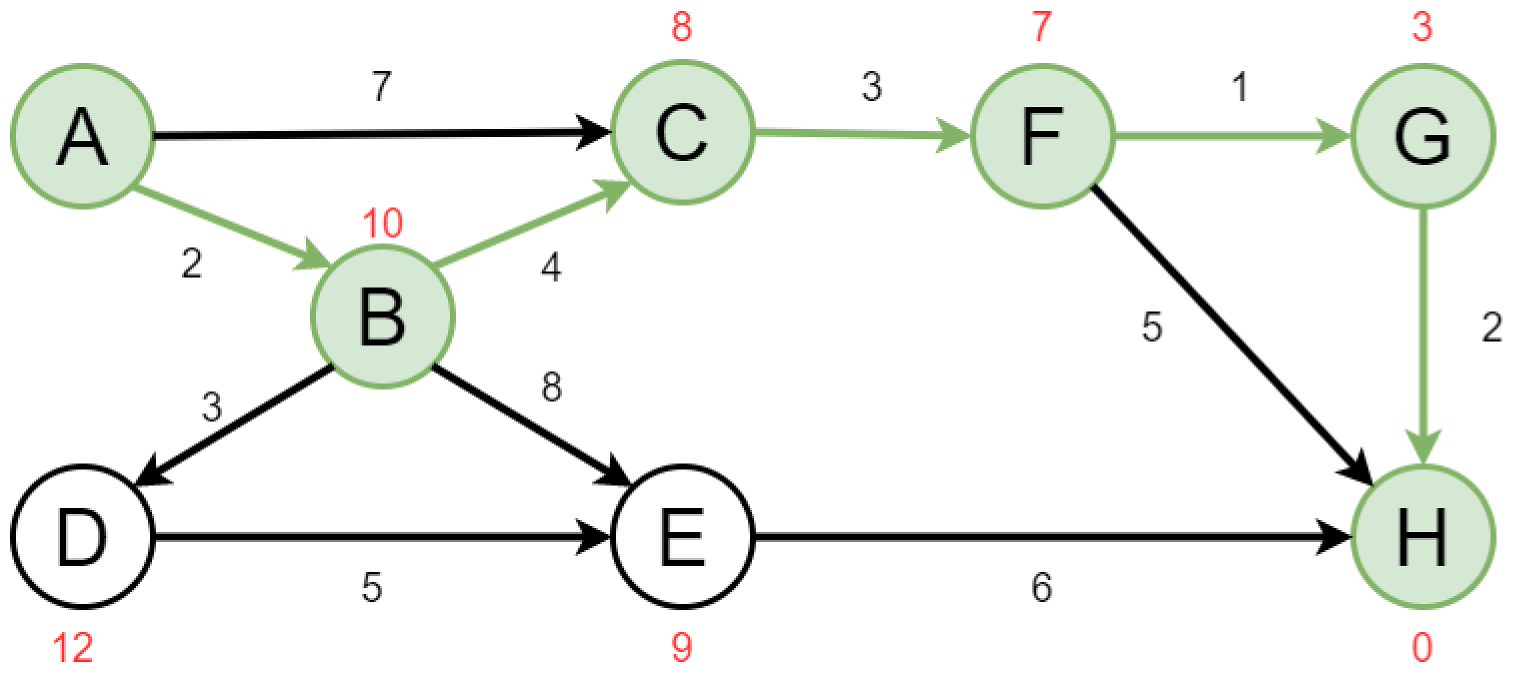
A* Algorithm : Example



A* Algorithm : Example



A* Algorithm : Example



A* Algorithm : Analysis

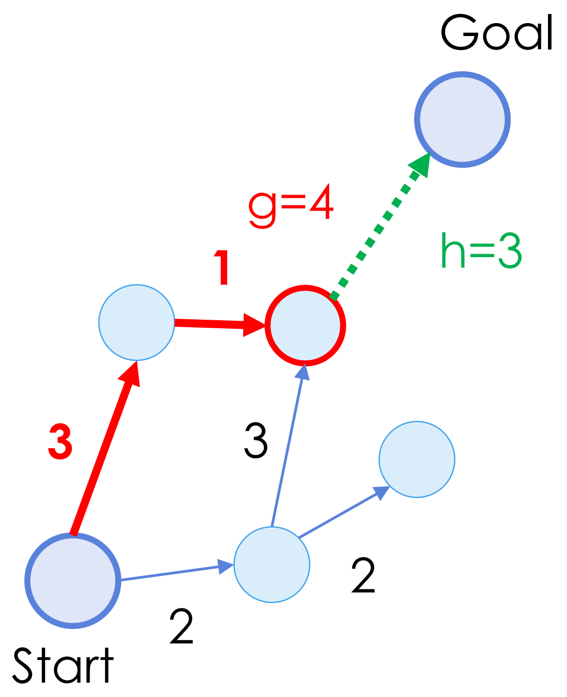
- In easy cases, A* can find a great path as fast as BFS
- In hard cases, A* can find a great path as good as Dijkstra

- Special condition of A*
  - h(v) = 0
    - degrade to Dijkstra's Algorithm
  - g(v)=0 or h(v) >> Edge weight
    - degrade to Best-First Search

- In summary, a good choice of heuristic function is needed

Heuristic Function

- For path Planning, we can utilize the distance between the node and the goal as the heuristic Function



Path Planning in Grid Space

预测未来路径距离的定义

- For mobile robot, we can discrete the 2D plane into grid space.
- The definition of distance can be in the following forms

2	1	2
1	0	1
2	1	2

$L_1\ Distance$:
$$|1| + |1| = 2$$

曼哈顿距离

$\sqrt{2}$	1	$\sqrt{2}$
1	0	1
$\sqrt{2}$	1	$\sqrt{2}$

$L_2\ Distance$:
$$\sqrt{1^2 + 1^2} = \sqrt{2}$$

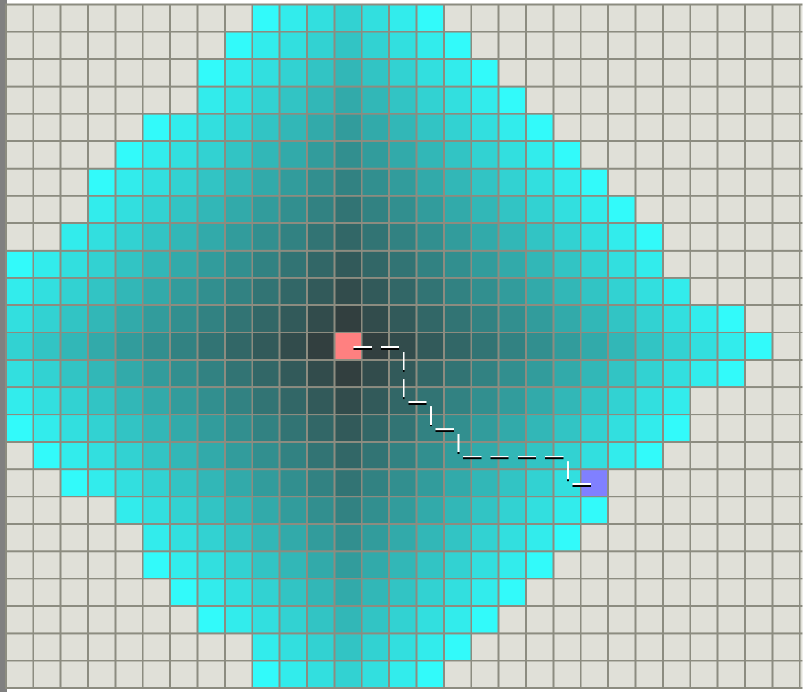
欧氏距离

1	1	1
1	0	1
1	1	1

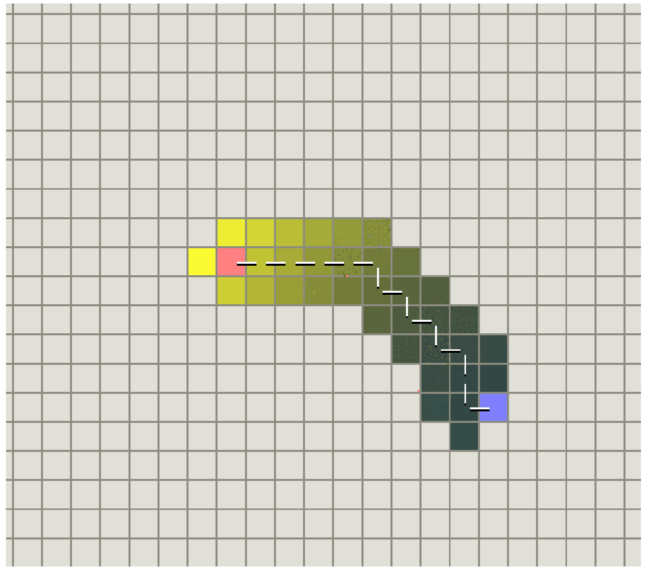
$L_\infty\ Distance$:
$$\sqrt[\infty]{1^\infty + 1^\infty} = \max(1,1) = 1$$

切里雪夫距离

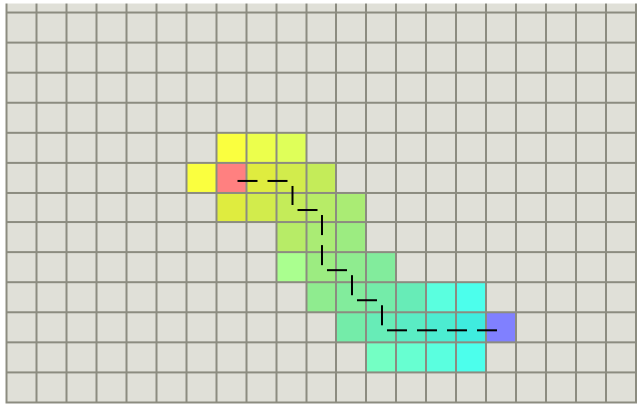
Comparison : Easy Case



Dijkstra
Great Path
Slow

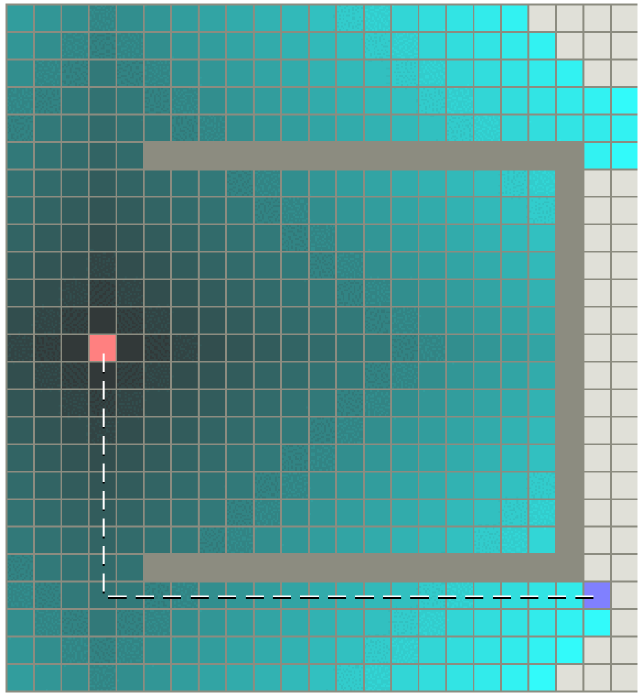


Best-First Search
Great Path
Fast

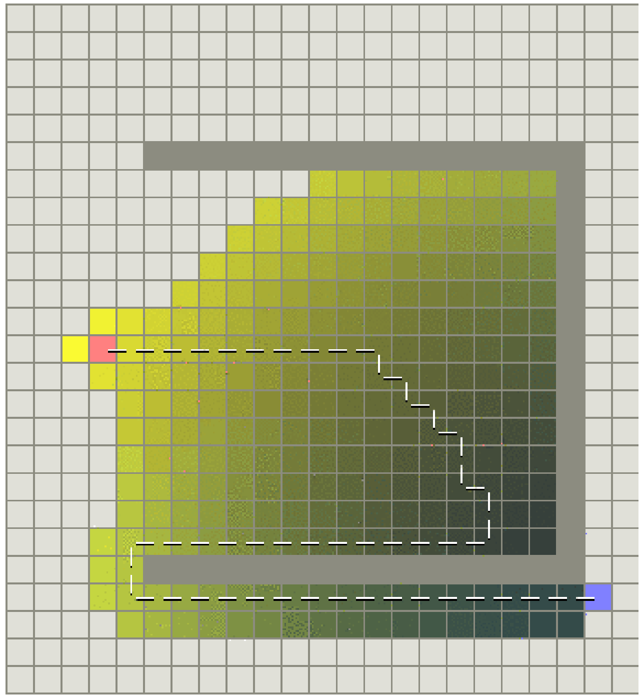


A* Search
Great Path
Fast

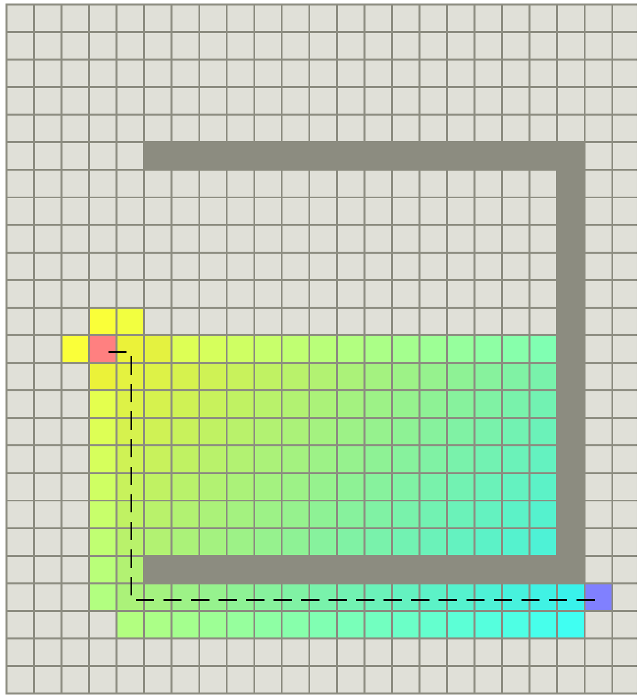
Comparison : Hard Case



Dijkstra
Great Path
Slow



Best-First Search
Bad Path
Fast

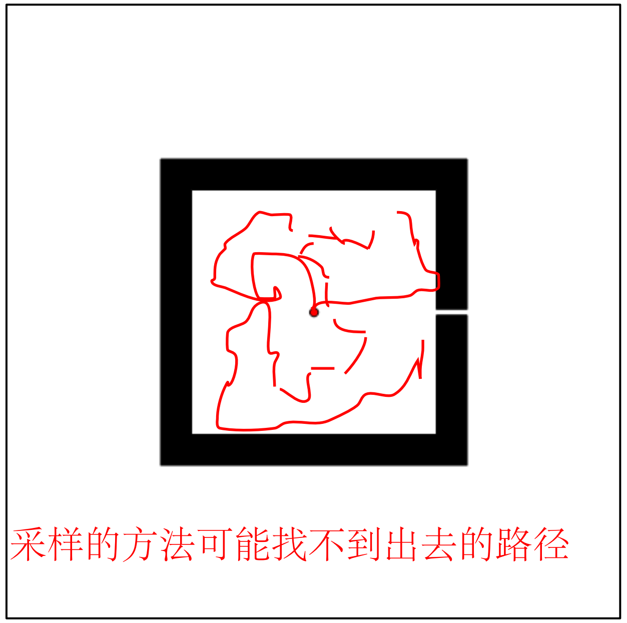


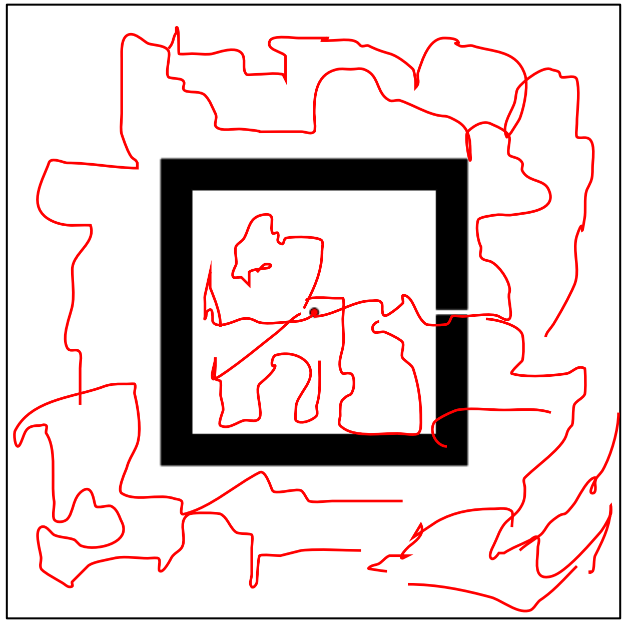
A* Search
Great Path
Fast

Sampling Based Planning Methods

- A* is resolution complete, while it still takes much time to search each grid in 2D space.
  需要花时间

  采样 减少搜寻数量
- Sampling based methods decrease the search state in 2D space, while the algorithms only output sub-optimal solution (probabilistic completeness).
  采样足够大的时候才能找到最佳解





采样方法

Probabilistic Road-Map (PRM)

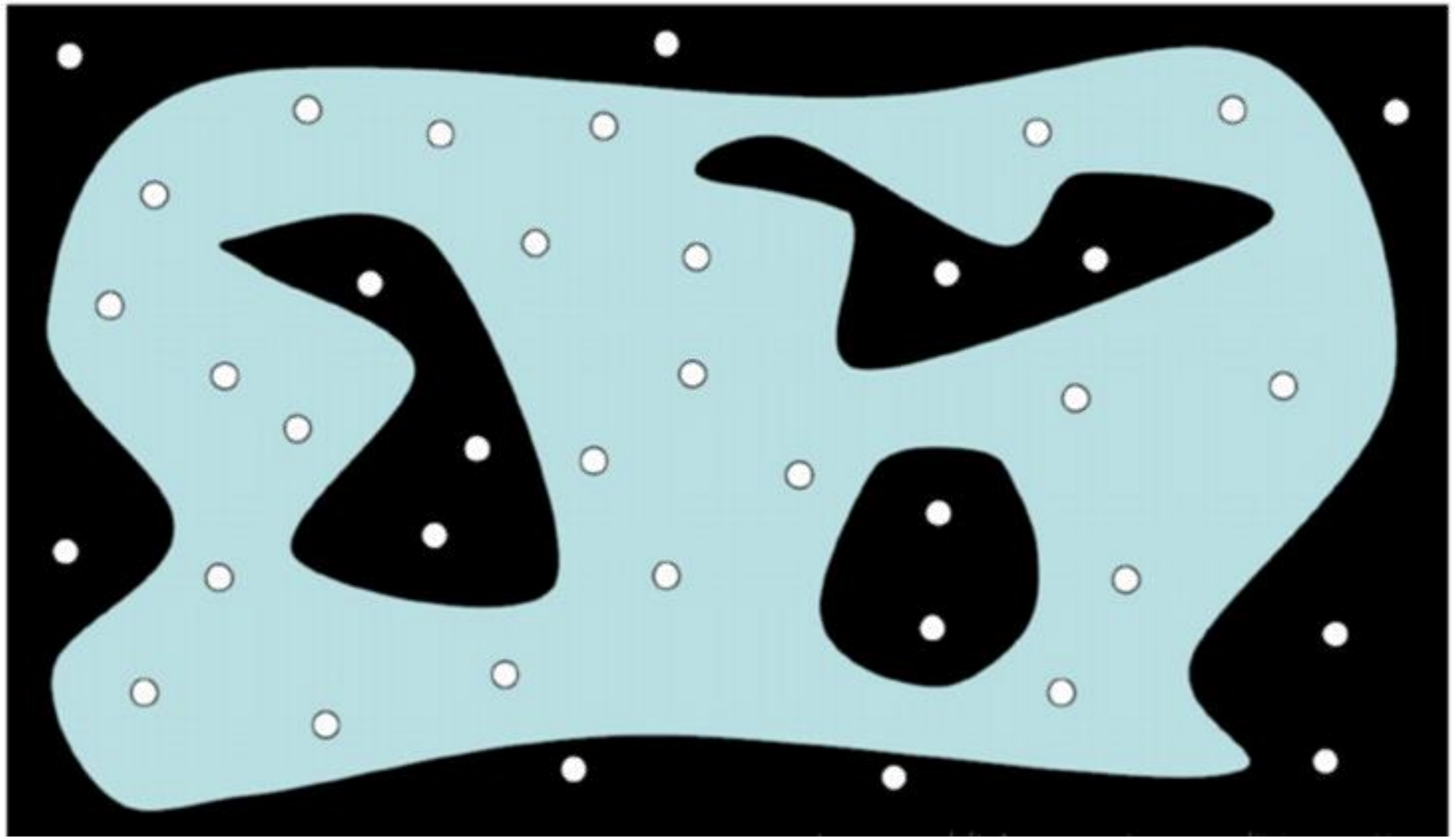
- To reduce search cost, PRM algorithm constructs a graph of the 2D space by randomly sampling positions. j建立稀疏位置节点的图，来减少搜索节点数量

- PRM Algorithm

1. Randomly generate points in free area. 在空间中随机撒点
2. Connect k-nearest neighbor points. 连接灵晶的点
3. Delete the edges which cross the occupied area. 删除通过障碍物的连线
4. Connect connected components.
5. Plan the path in the generated graph. 用图搜寻算法

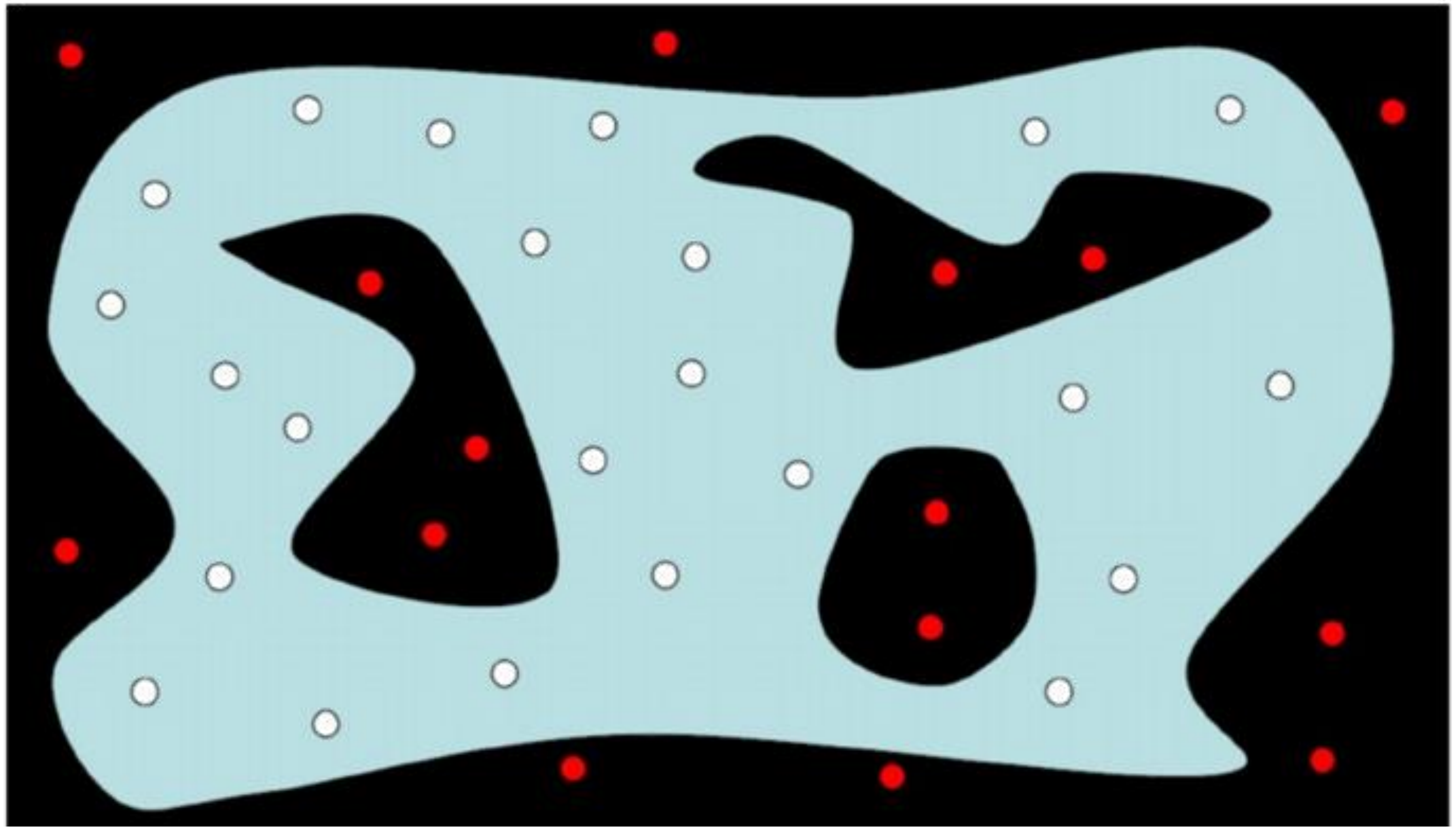
Probabilistic Road-Map (PRM)

- Randomly sample points in 2D space.



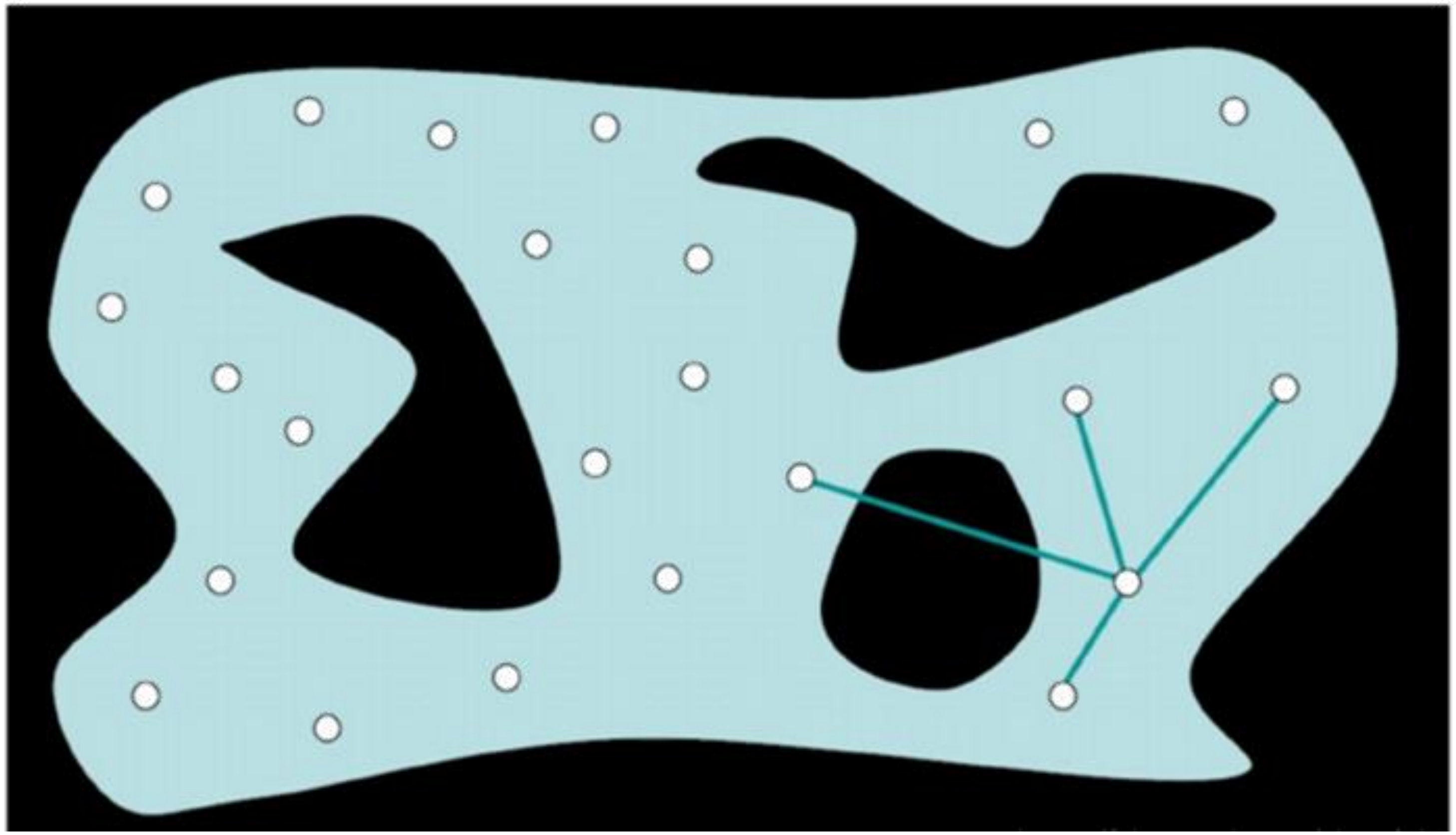
Probabilistic Road-Map (PRM)

- Delete the points which are at the occupied area.



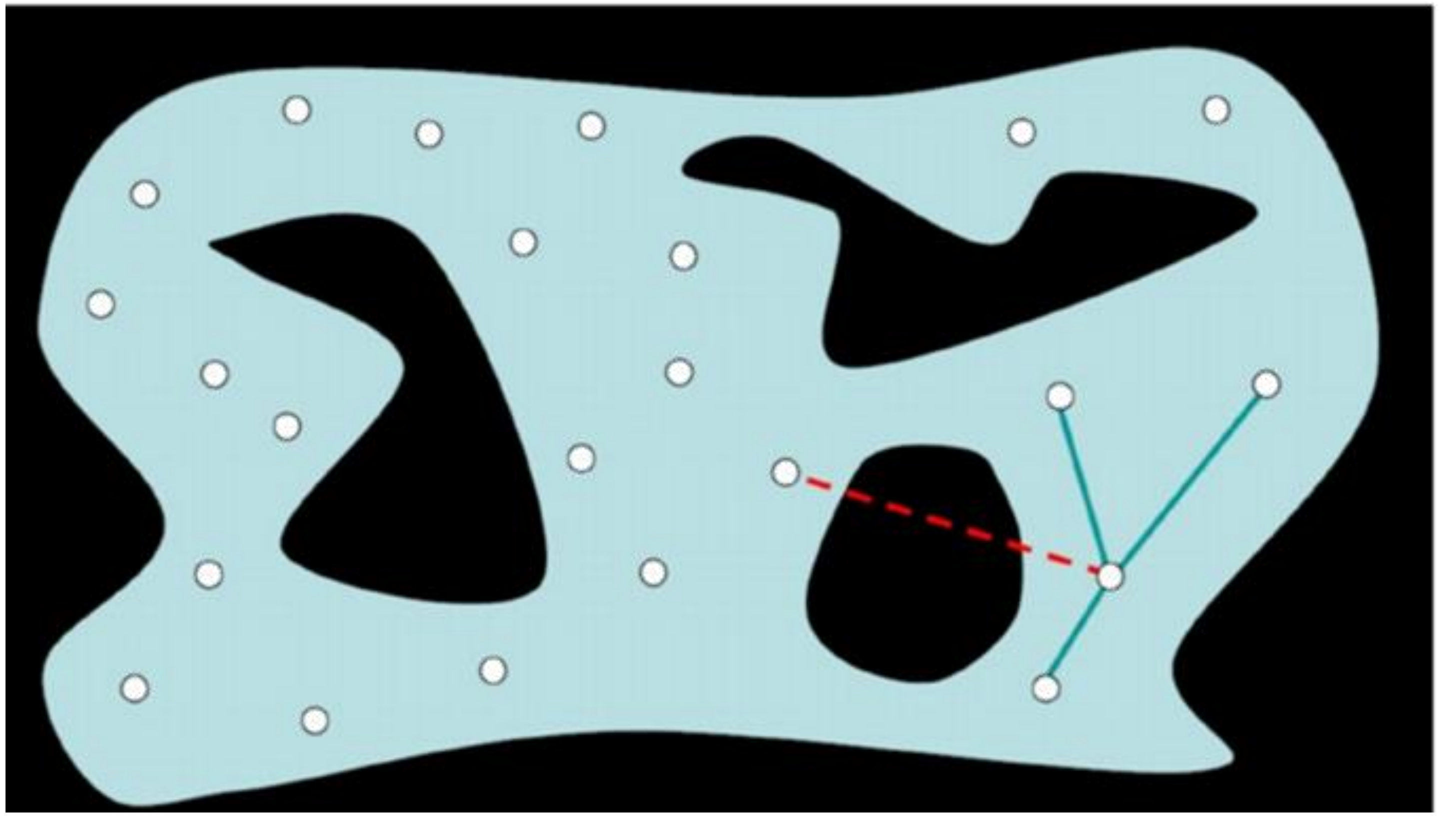
Probabilistic Road-Map (PRM)

- Connect the k-nearest neighbors.



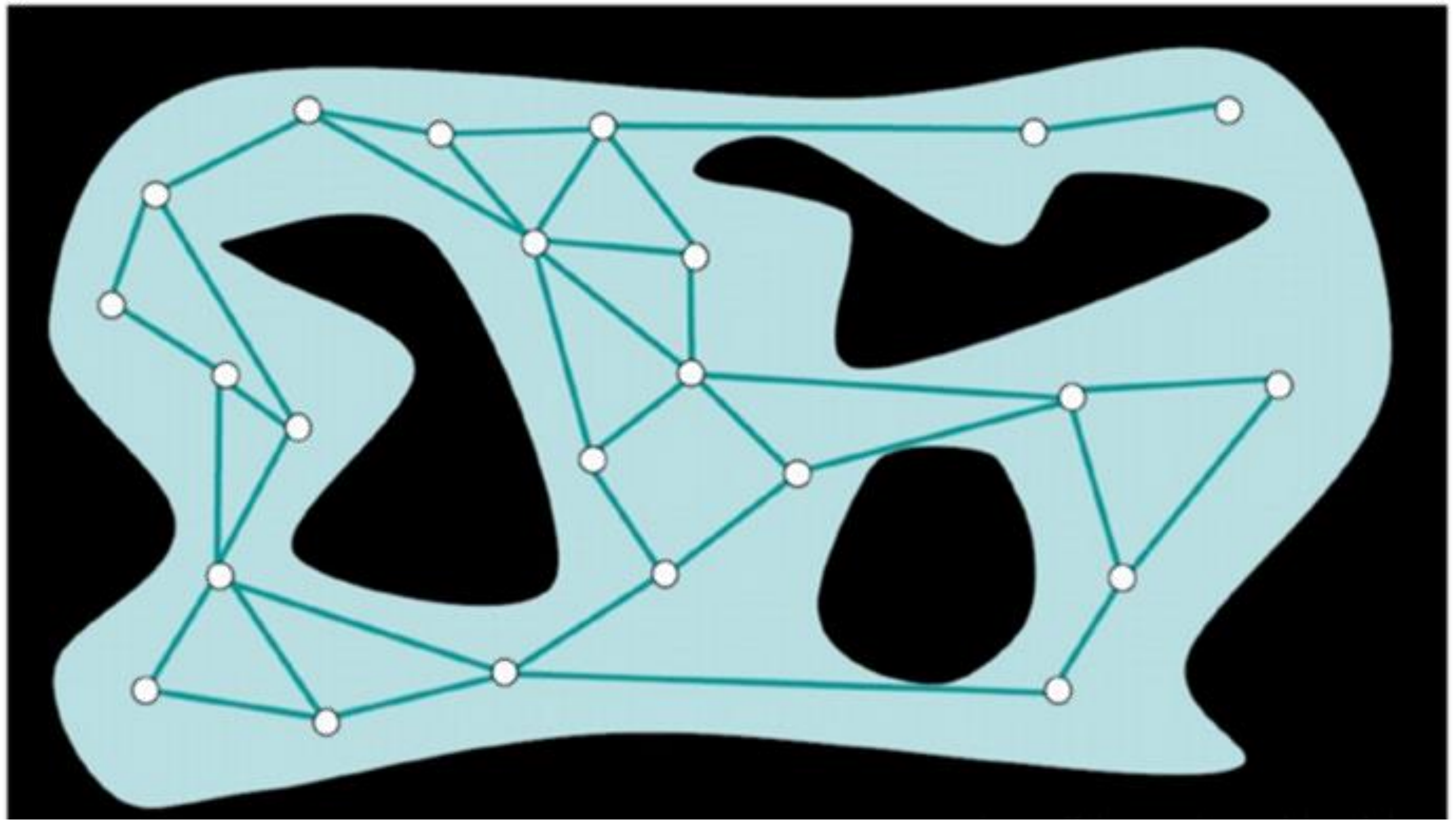
Probabilistic Road-Map (PRM)

- Delete the edges that cross through the occupied area.



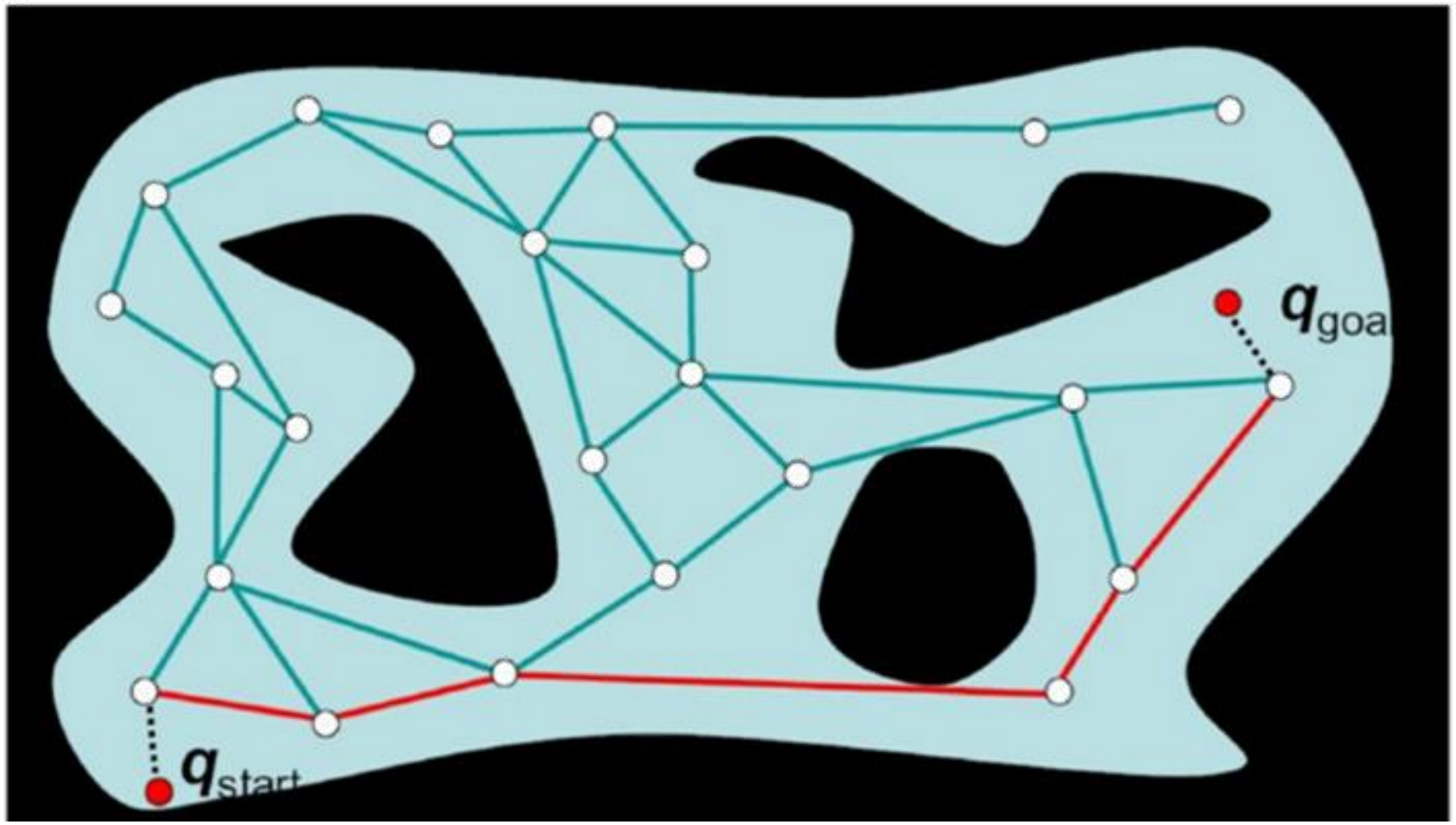
Probabilistic Road-Map (PRM)

- Connect each connected component to get the probabilistic road-map.



Probabilistic Road-Map (PRM)

- Connect source and destination points to the graph and plan the path, and plan the path utilizing graph search-based methods.



Rapidly Exploring Random Tree (RRT)

需要花费时间建图，需要反复计算邻近的点和碰撞检测

- Probabilistic Road-Map (PRM) spends lots of time to build the graph. To solve this problem, Rapidly exploring Random Tree (RRT) dynamically generates the tree branch and checks collision, which is more efficient.
  从起点出发动态的生成路径，并进行碰撞检测，只需要对当前延申的路径进行计算
- The improved version of RRT is RRT*, which is the most commonly used path planning algorithm for mobile robot. 最常使用
- RRT Algorithm:

```
$G$.initialize($x_{start}$)   # Initialize Graph
for i=1 to max_iter do
    $x_{rand} \leftarrow$ sample()   # Sample the points in 2d space.
    $x_{nearest} \leftarrow$ nearest($x_{rand}$ , $G$)   # Find the nearest point in graph.
    $x_{new} \leftarrow$ steer($x_{rand}$ , $x_{nearest}$ , step_size)   # Collision detection.
    $G$.add_node($x_{new}$)
    $G$.add_edge($x_{new}$ , $x_{parent}$)
    if $||x_{new} - x_{goal}|| <$ threshold then
        return $G$
```

从空间中随机采集一个点

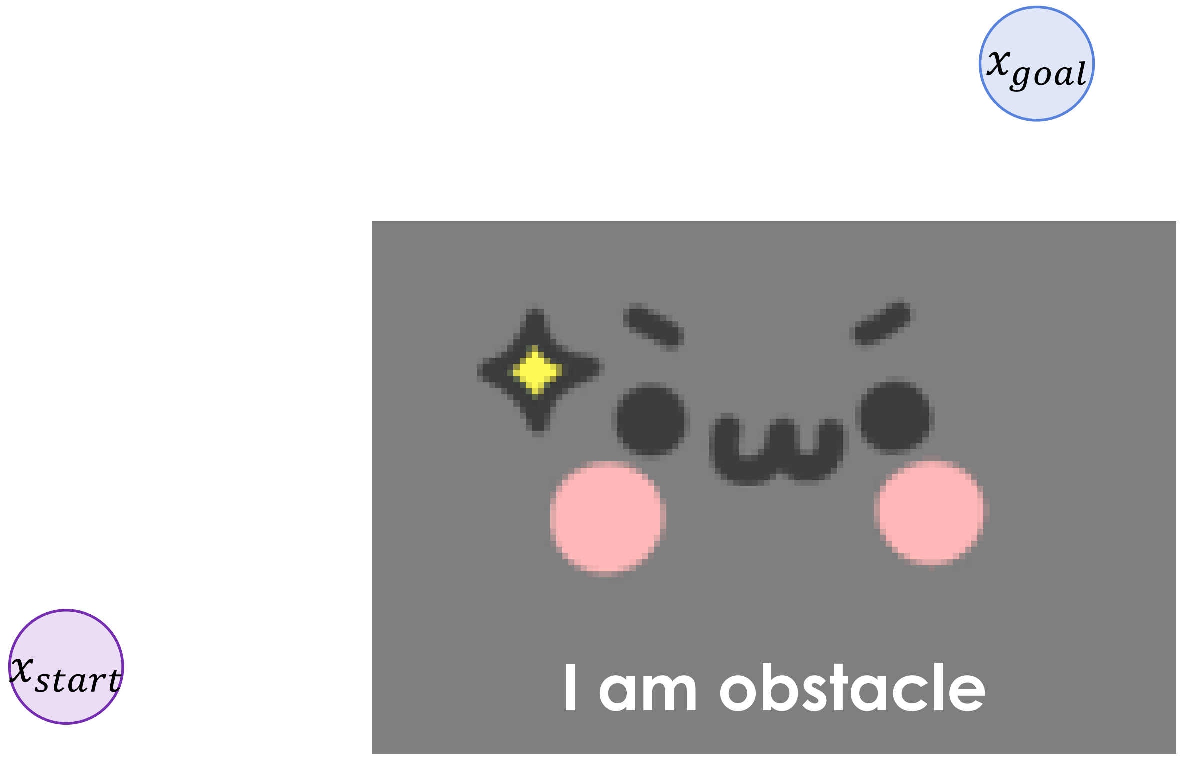
在图中找离随机点最近的点

沿着最近的点到随机点的方向，延申出新的点

确认是否碰撞

Rapidly Exploring Random Tree (RRT) (Cont.)

- Initialize graph

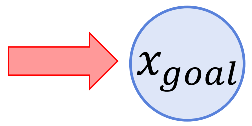


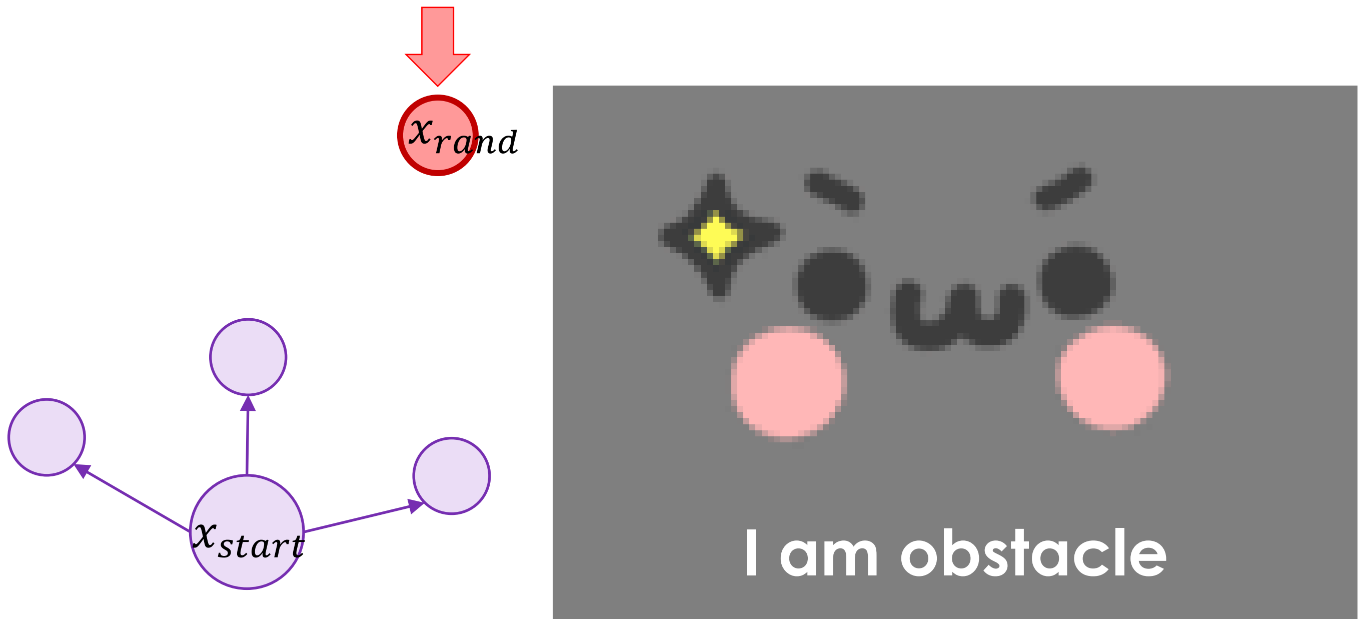
Rapidly Exploring Random Tree (RRT) (Cont.)

- Sample points on 2d space

贪心算法

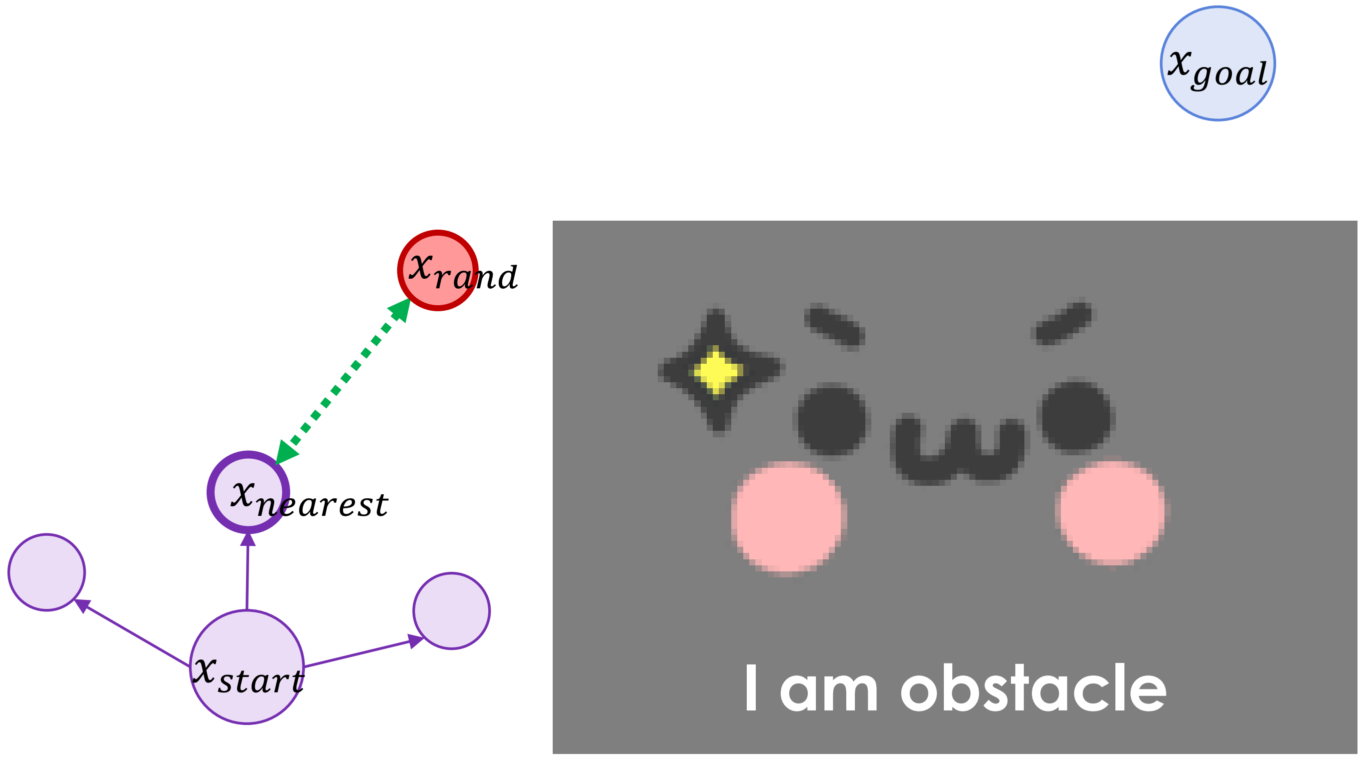
Probability (p) for sampling random point,
and probability (1-p) for choosing the goal point.





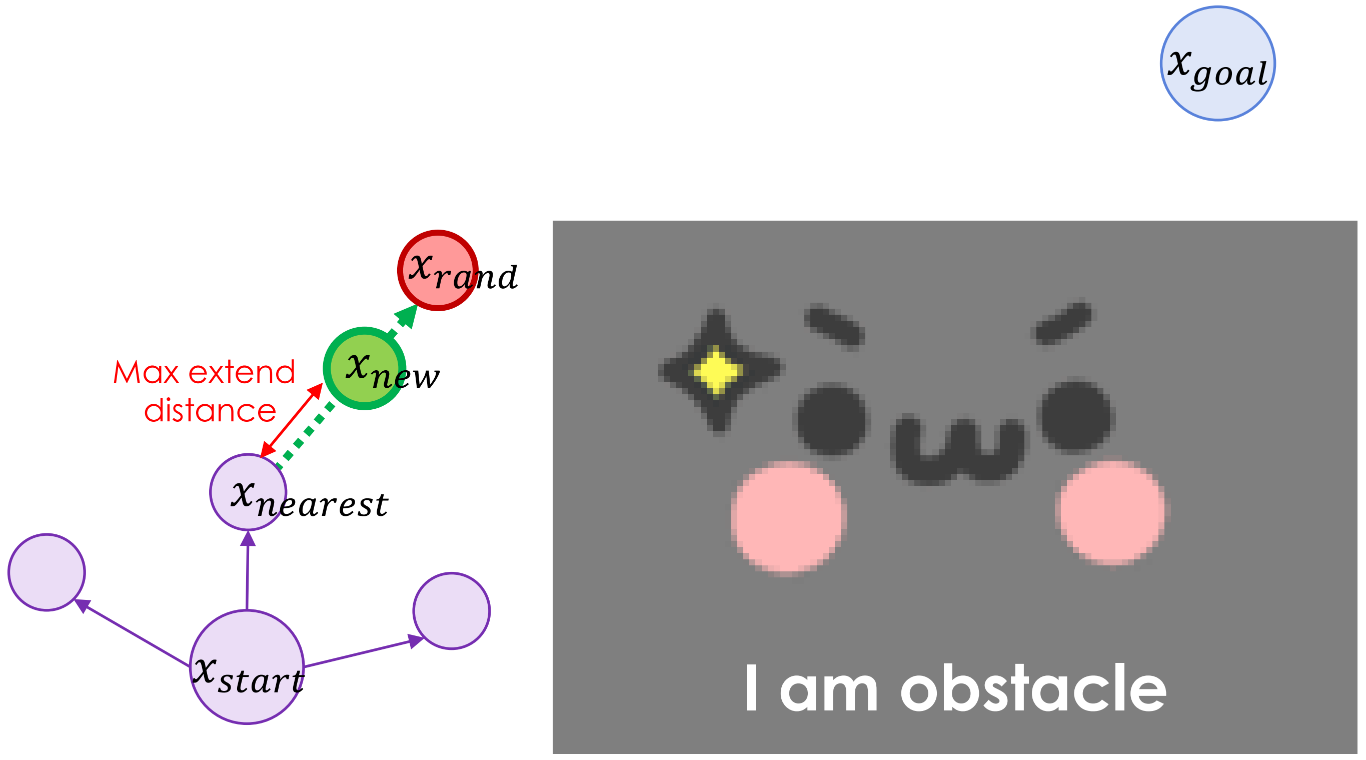
Rapidly Exploring Random Tree (RRT) (Cont.)

- Find the nearest point in graph.



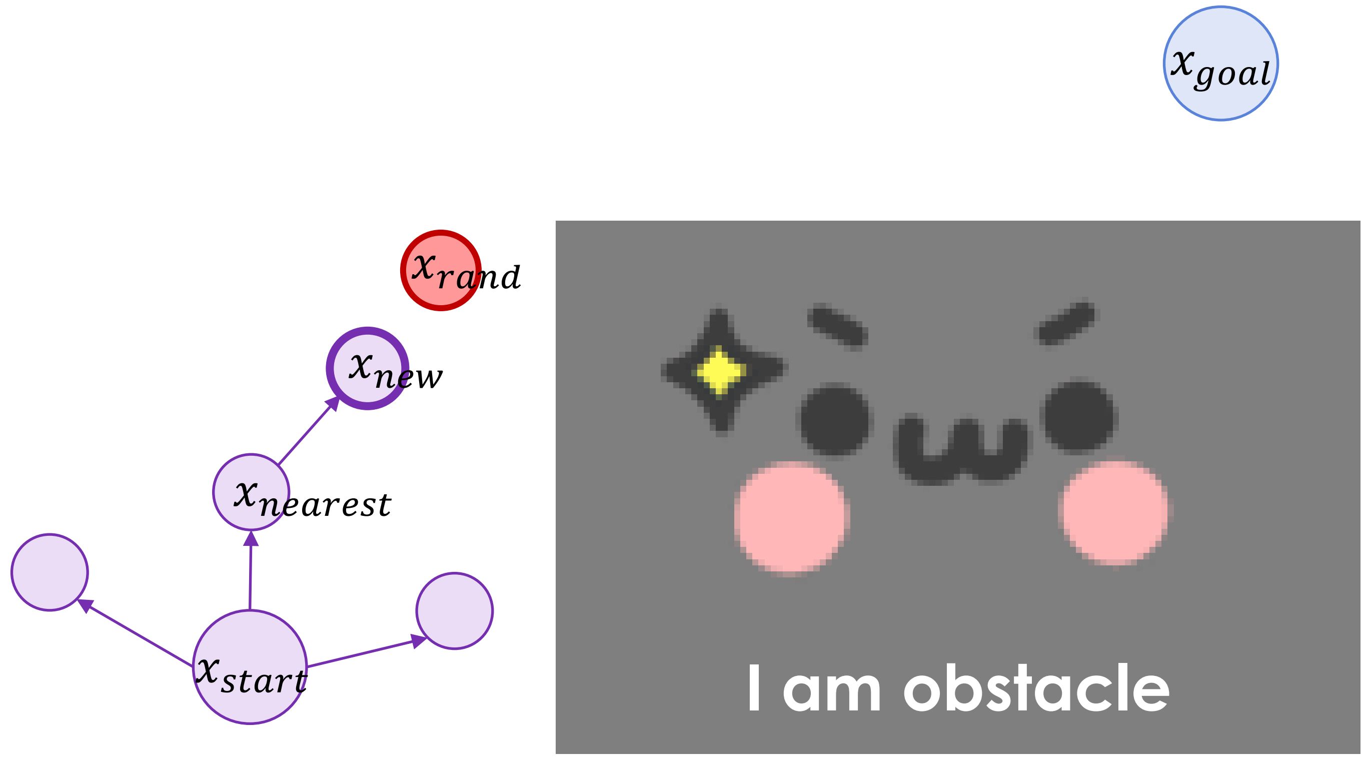
Rapidly Exploring Random Tree (RRT) (Cont.)

- Extend the branch of the nearest node and check the collision.

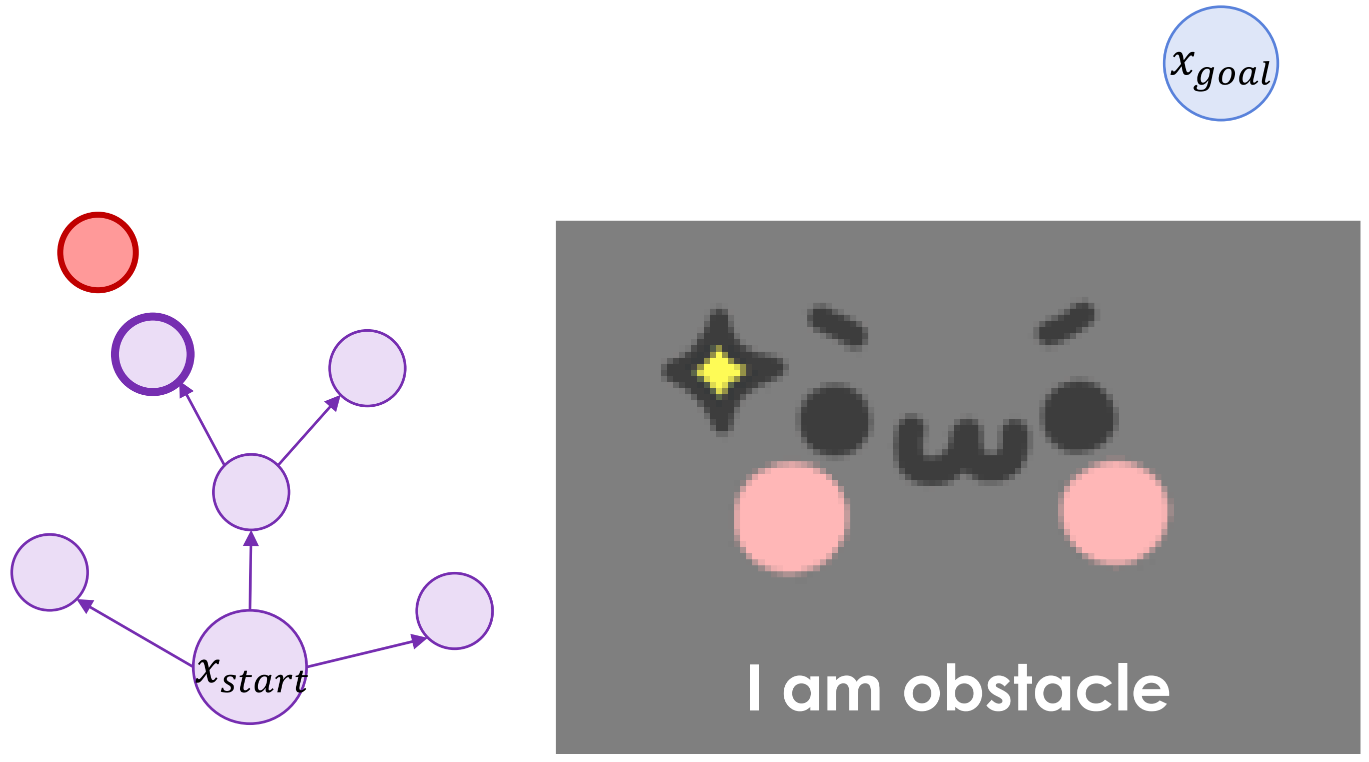


Rapidly Exploring Random Tree (RRT) (Cont.)

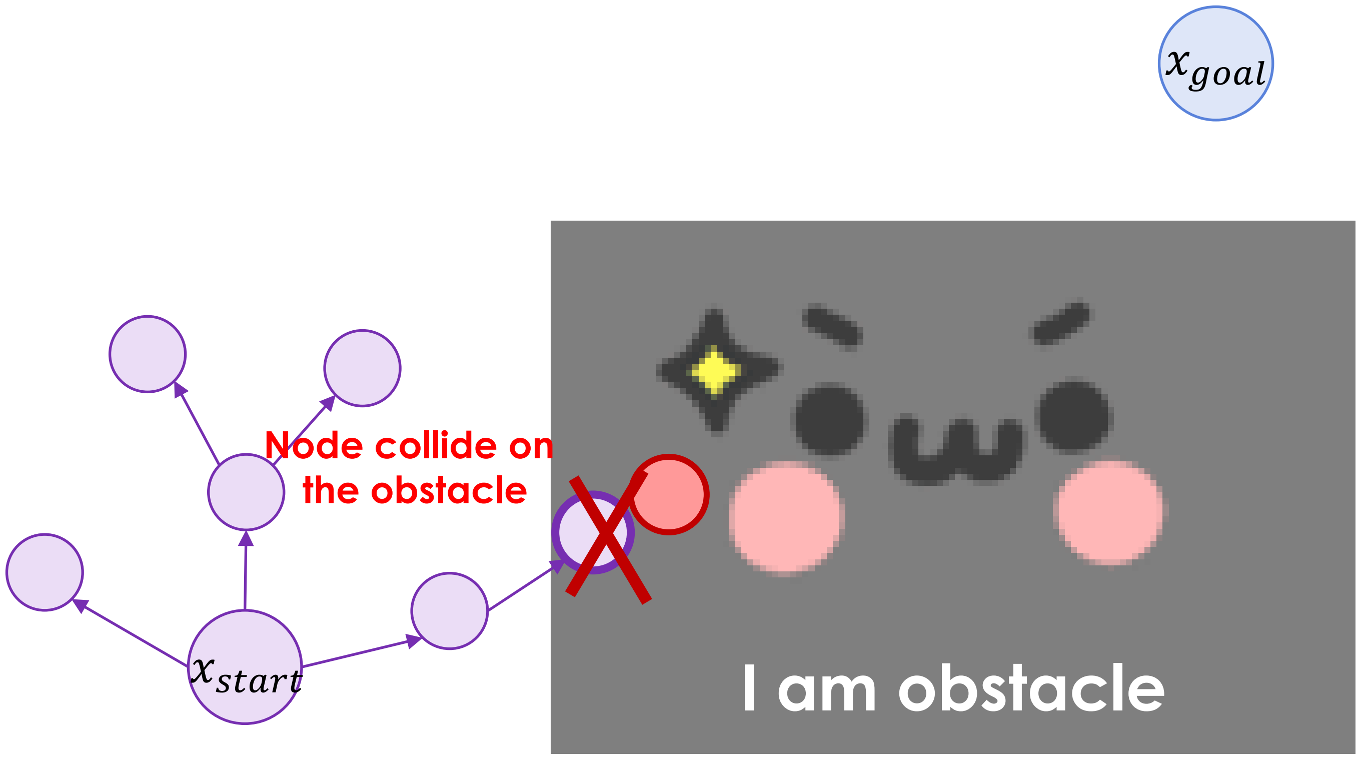
- Add the new node and edge into the graph



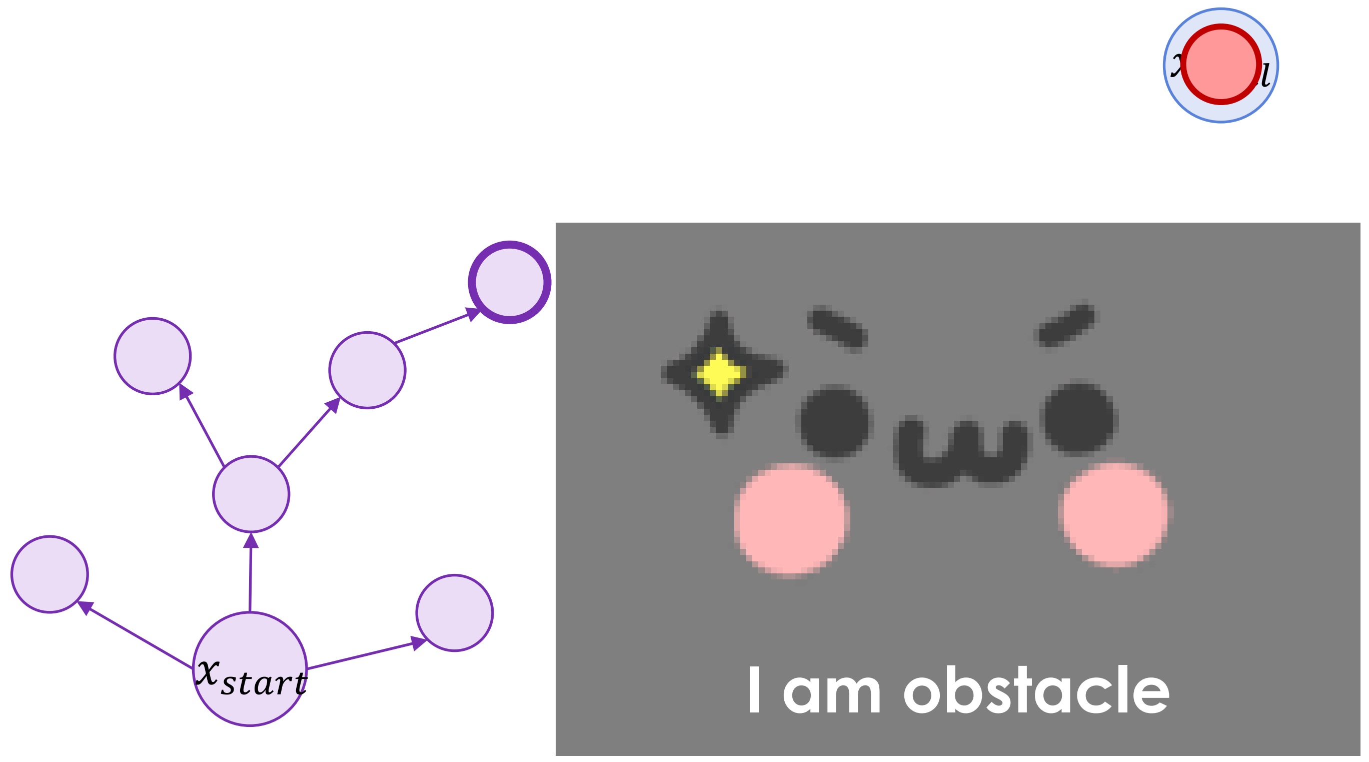
Rapidly Exploring Random Tree (RRT) (Cont.)



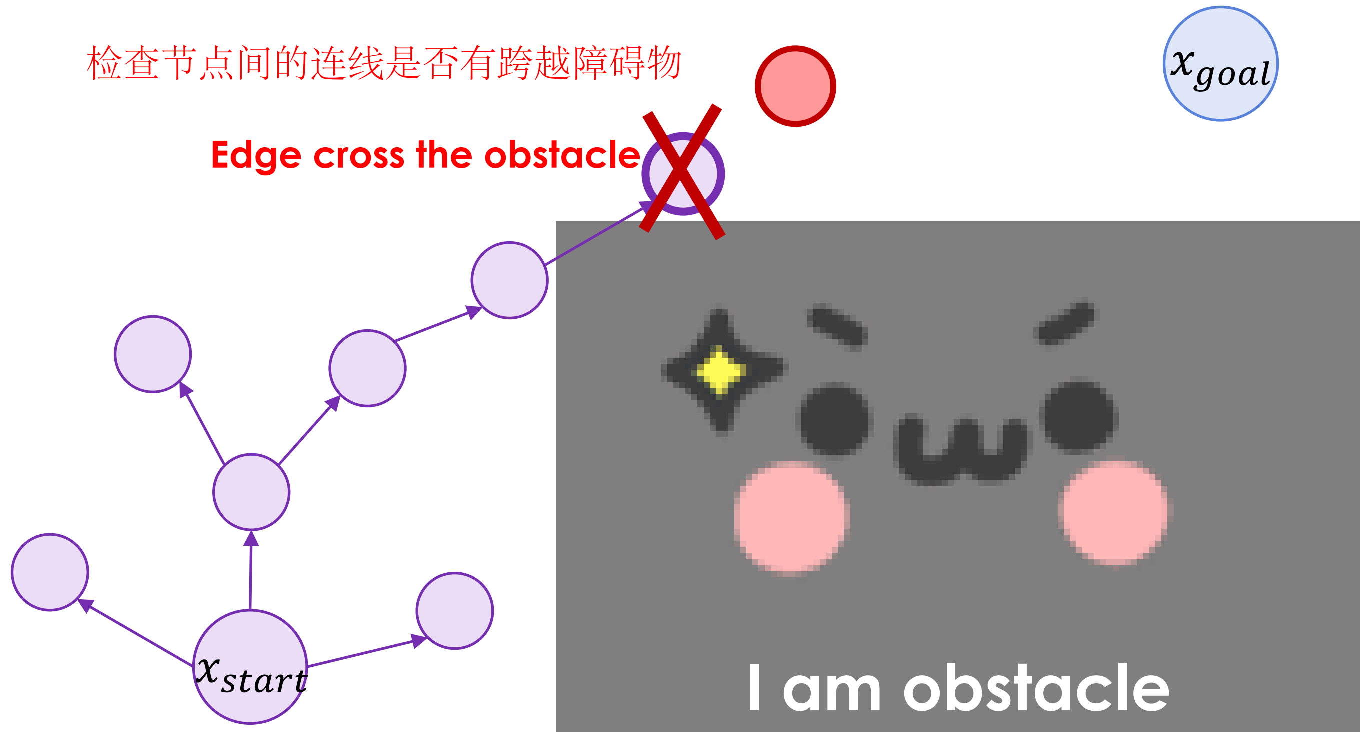
Rapidly Exploring Random Tree (RRT) (Cont.)



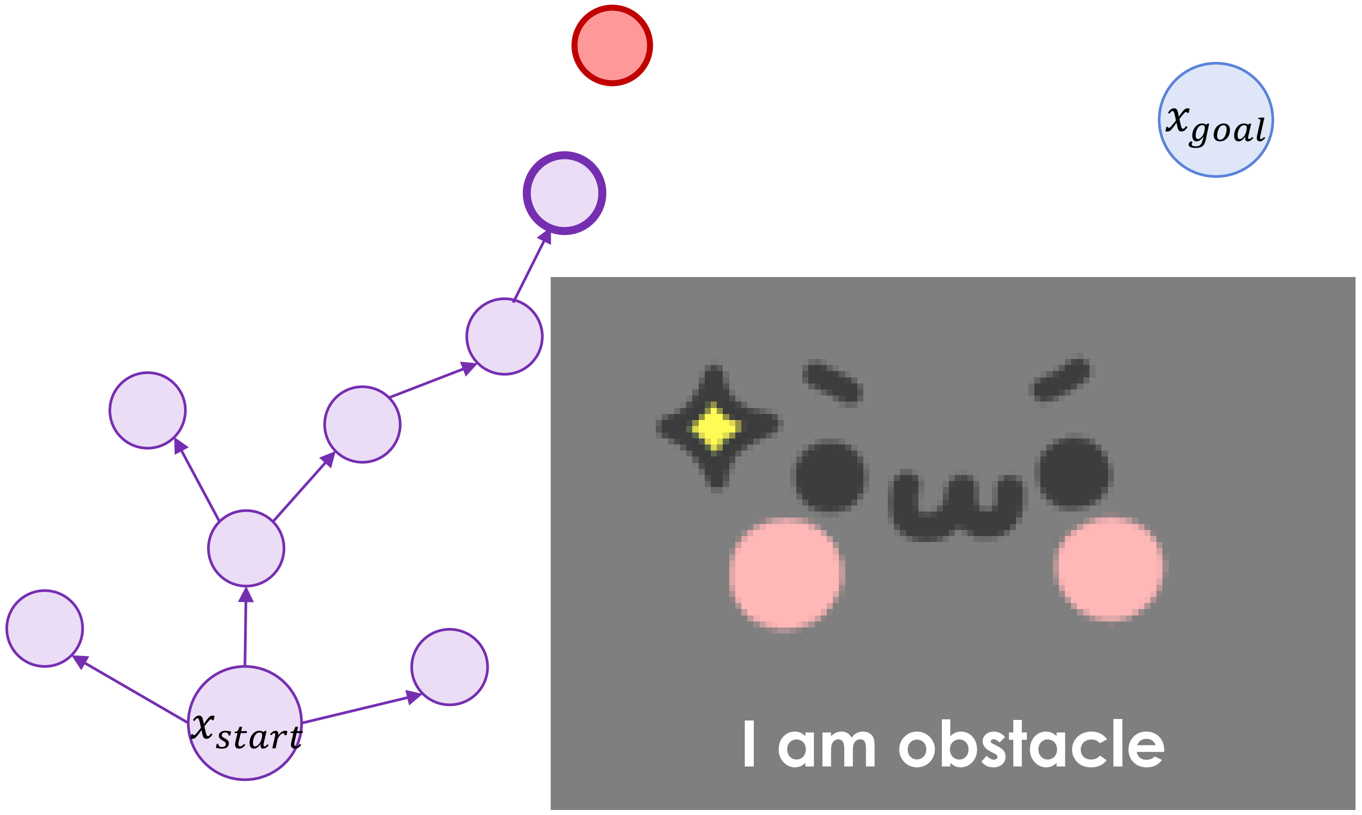
Rapidly Exploring Random Tree (RRT) (Cont.)



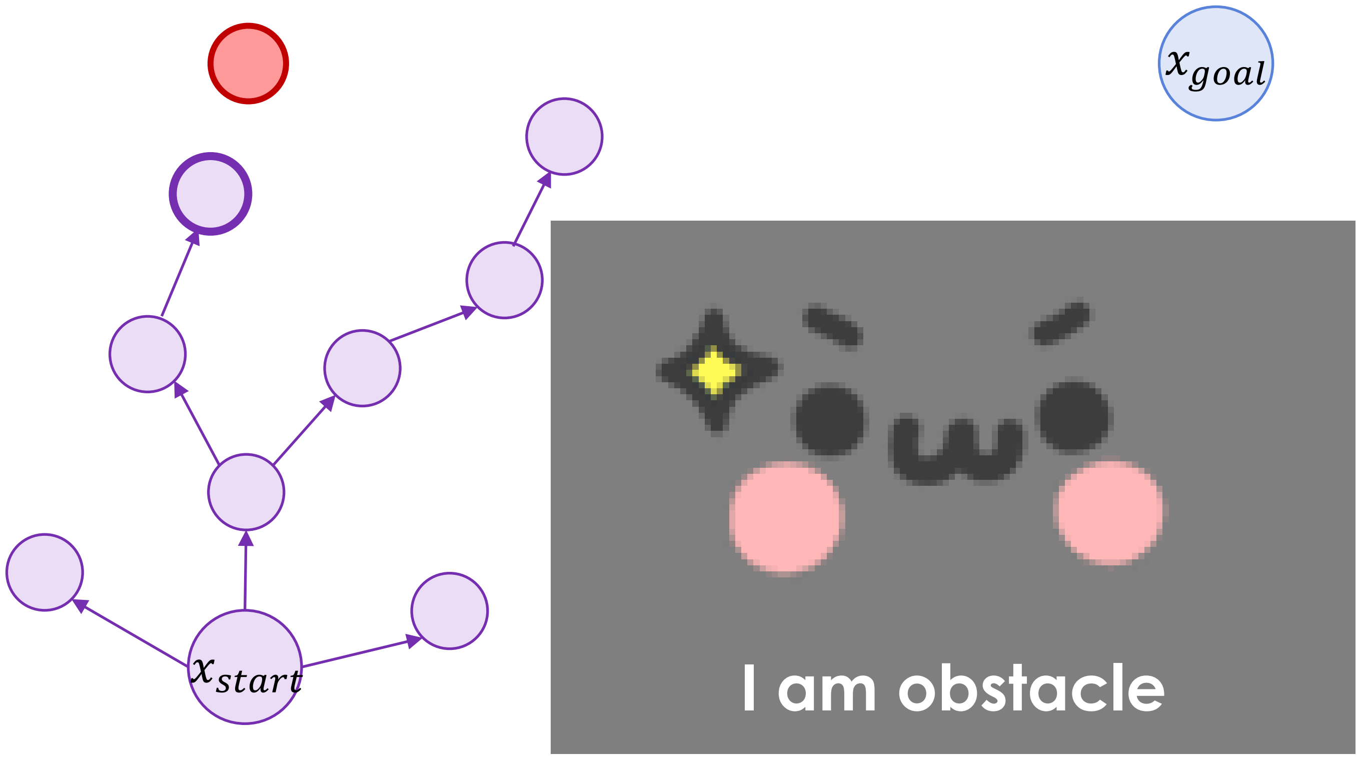
Rapidly Exploring Random Tree (RRT) (Cont.)



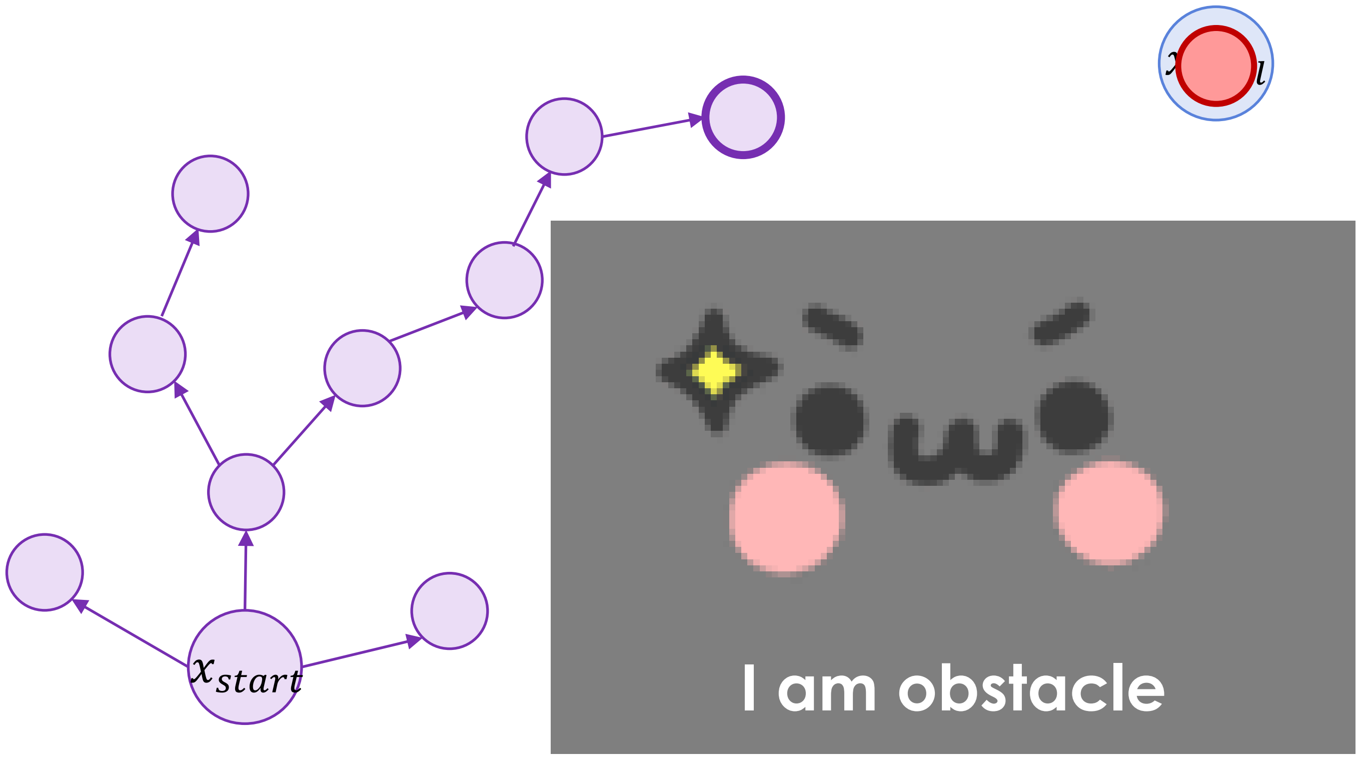
Rapidly Exploring Random Tree (RRT) (Cont.)



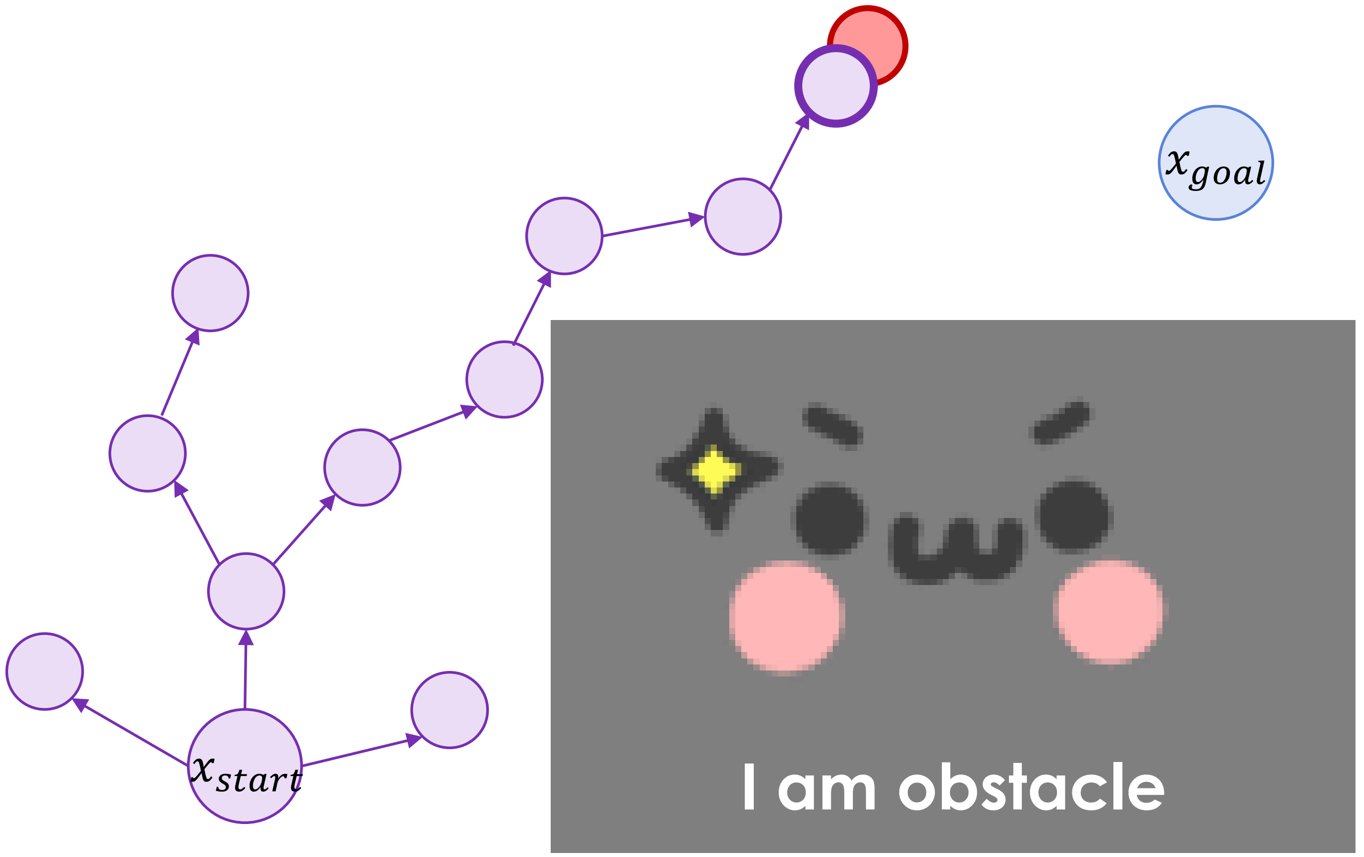
Rapidly Exploring Random Tree (RRT) (Cont.)



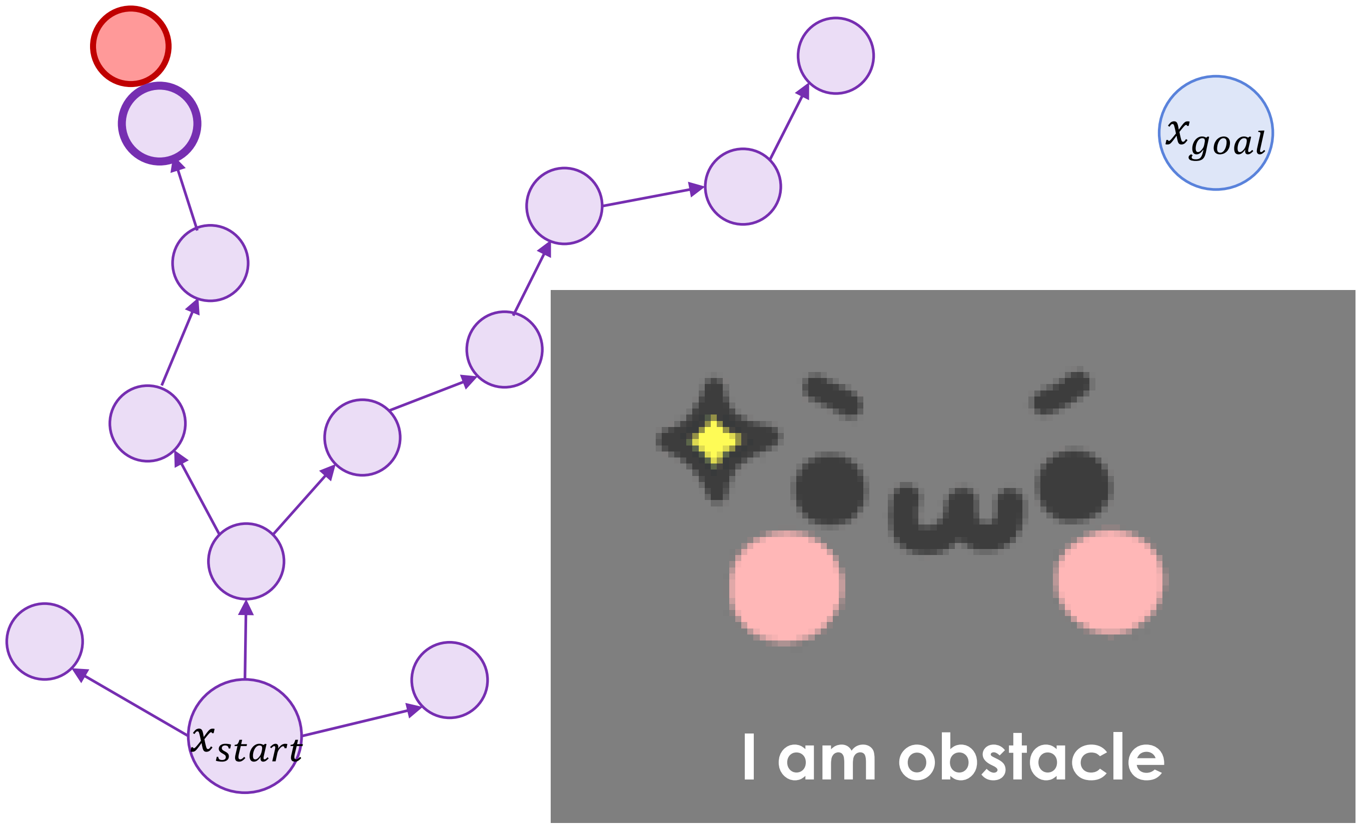
Rapidly Exploring Random Tree (RRT) (Cont.)



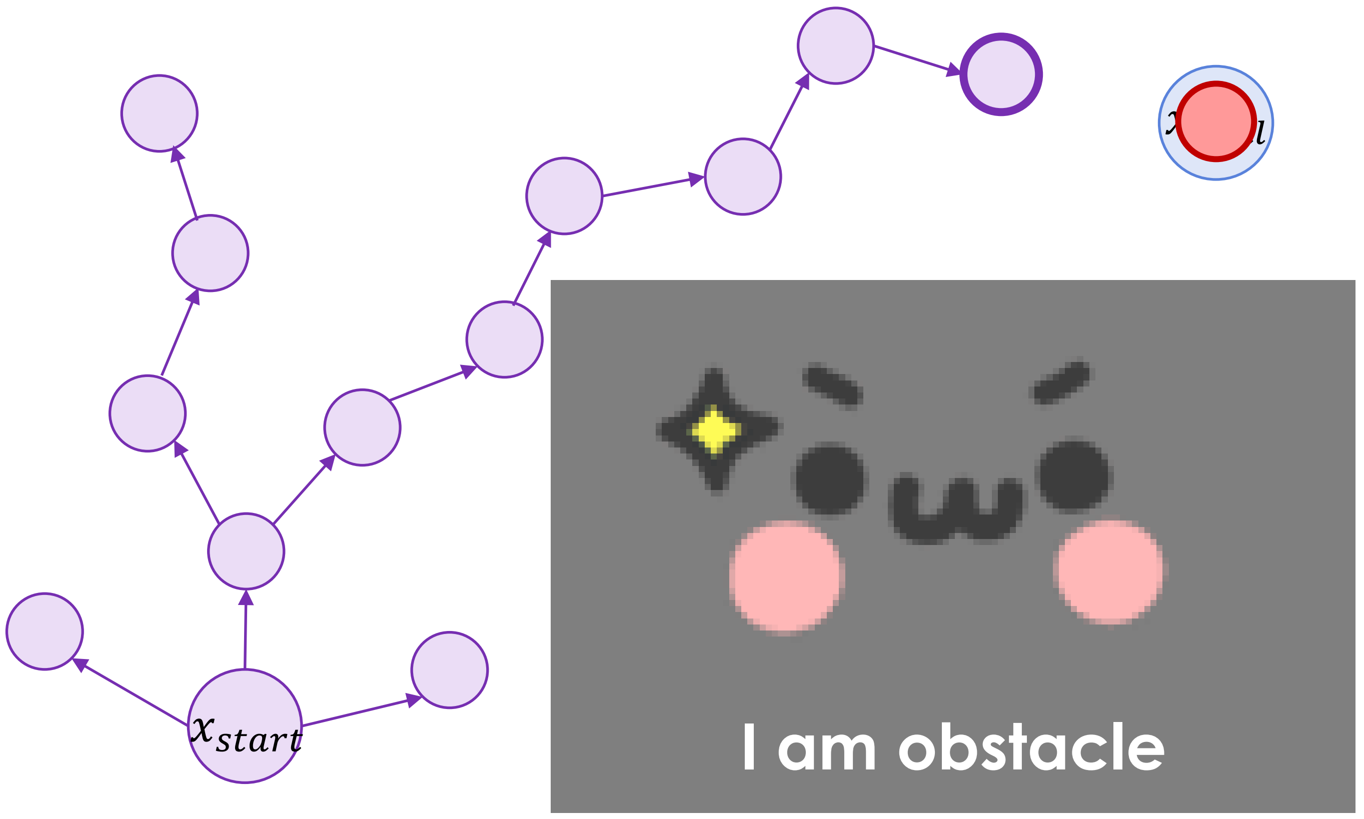
Rapidly Exploring Random Tree (RRT) (Cont.)



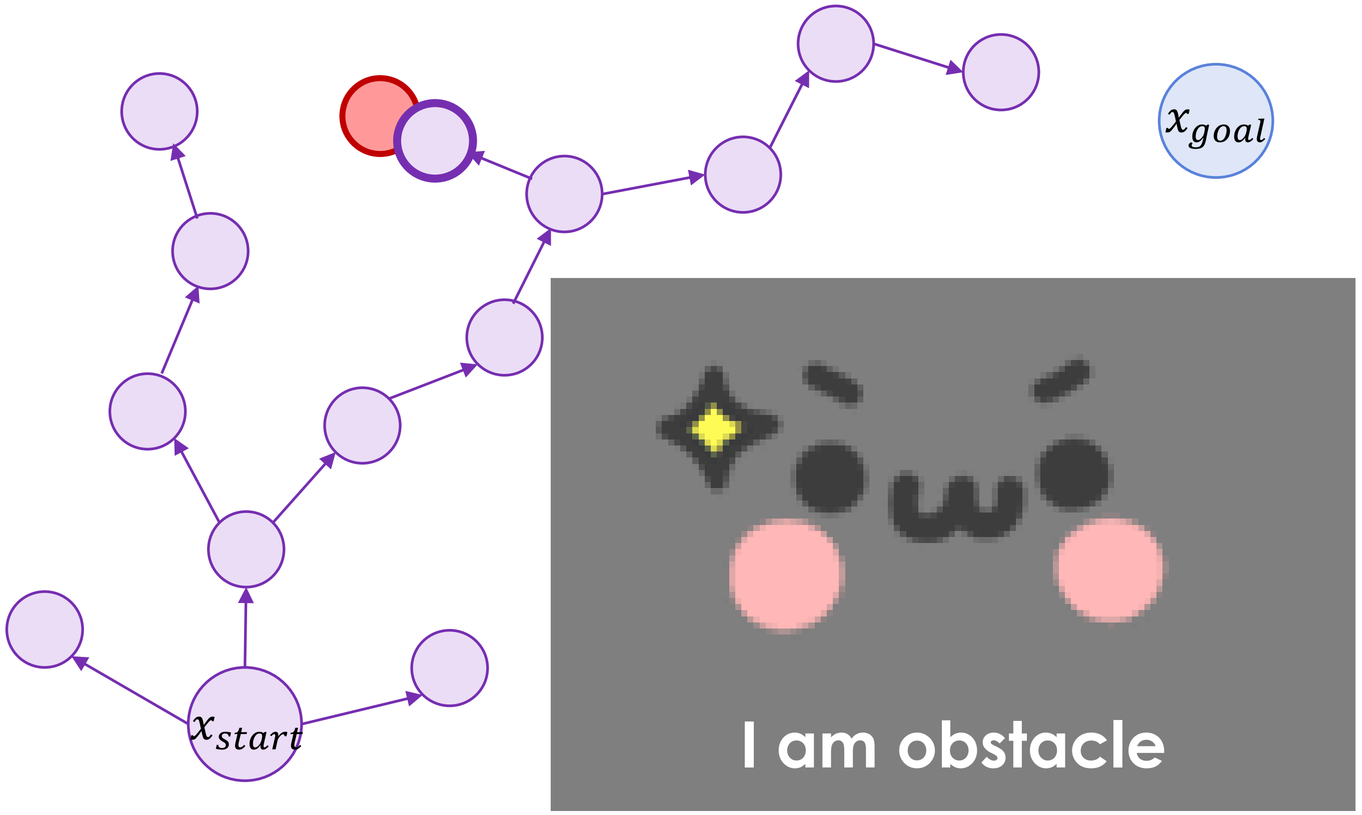
Rapidly Exploring Random Tree (RRT) (Cont.)



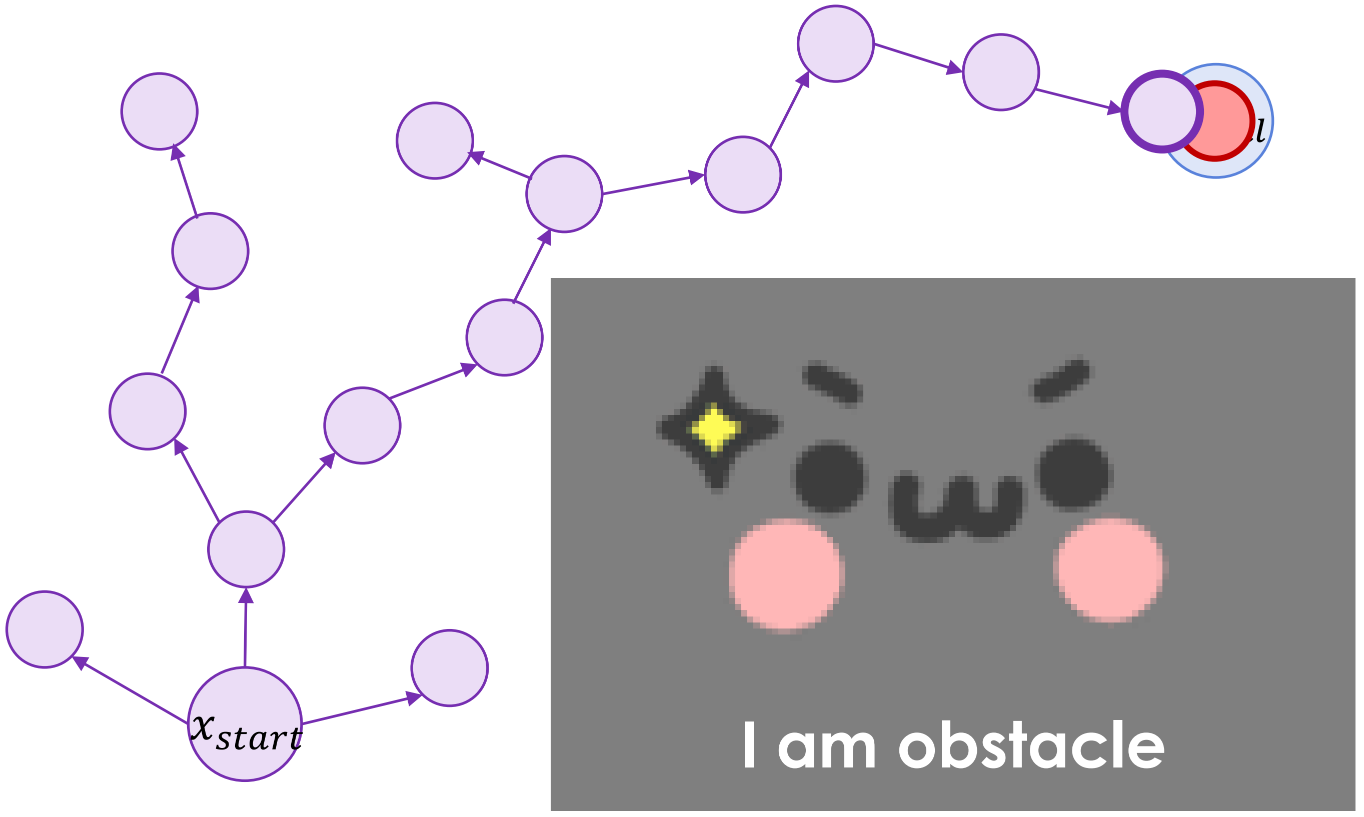
Rapidly Exploring Random Tree (RRT) (Cont.)



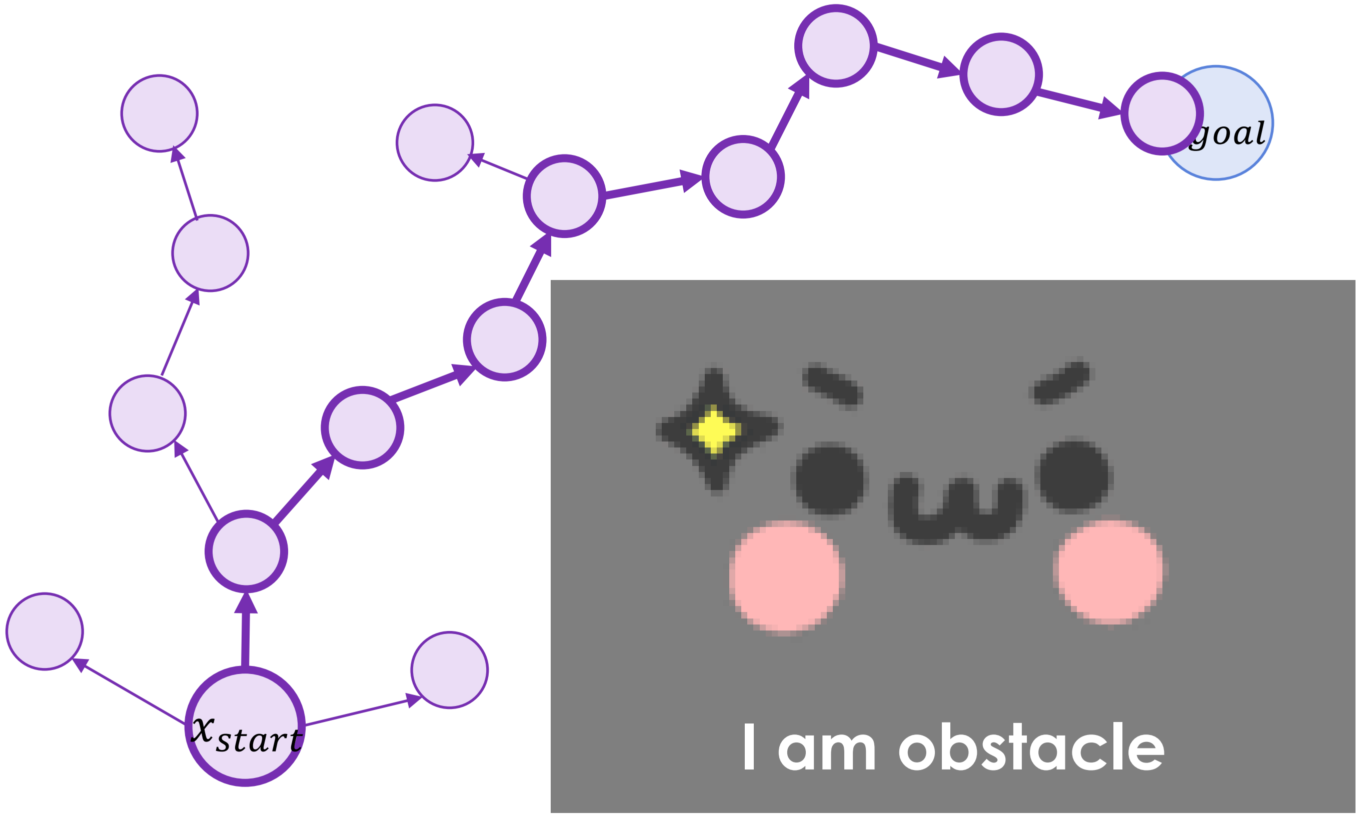
Rapidly Exploring Random Tree (RRT) (Cont.)



Rapidly Exploring Random Tree (RRT) (Cont.)



Rapidly Exploring Random Tree (RRT) (Cont.)



RRT*

添加新节点没有考虑距离因素，产生的路径非常曲折

- The path planned by RRT is winding and jerky, RRT star utilizes the re-parent and the re-wire process to improve the smoothness.

保持路径平滑

- RRT* Algorithm

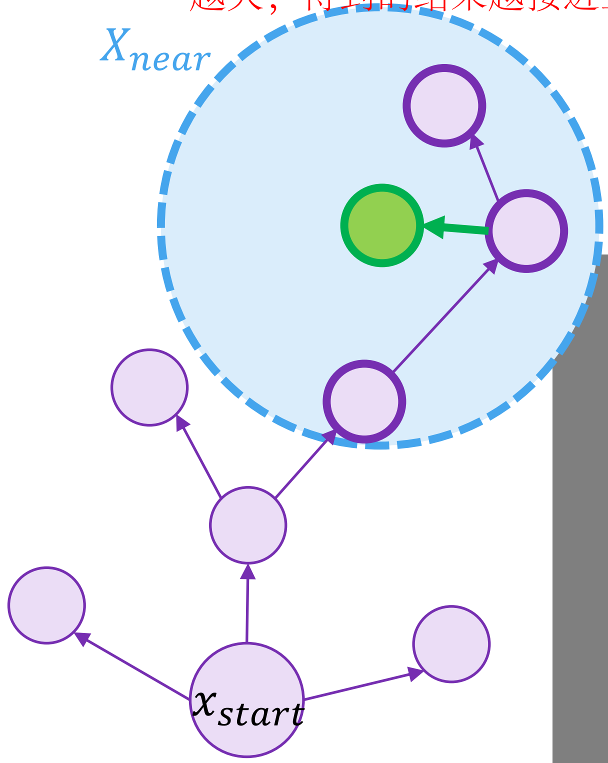
```
$G$.initialize($x_{start}$)   # Initialize Graph
for i=1 to max_iter do
    $x_{rand} \leftarrow$ sample()   # Sample the points in 2d space.
    $x_{nearest} \leftarrow$ nearest($x_{rand}$ , $G$)   # Find the nearest point in graph.
    $x_{new} \leftarrow$ steer($x_{rand}$ , $x_{nearest}$ , step_size)   # Collision detection.
    $X_{near} \leftarrow$ near_node($x_{new}$ , $G$)
    $x_{parent} \leftarrow$ best_parent($x_{parent}$ , $X_{near}$)   # Re-Parent
    $G$.add_node($x_{new}$)
    $G$.add_edge($x_{new}$ , $x_{parent}$)
    $G$.rewire($x_{new}$ , $X_{near}$) # Rewire
    if $||x_{new} - x_{goal}|| <$ threshold then
        return $G$
```

对邻近的点集合进行重新规划

RRT* - Near Nodes

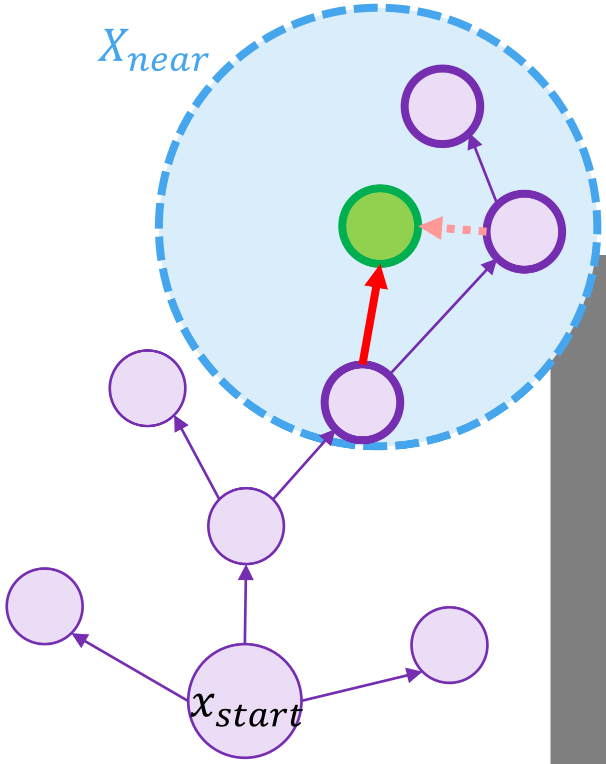
- Find the near nodes set of the new node in a distance range.

越大，得到的结果越接近正确的



RRT* - Re-Parents Process

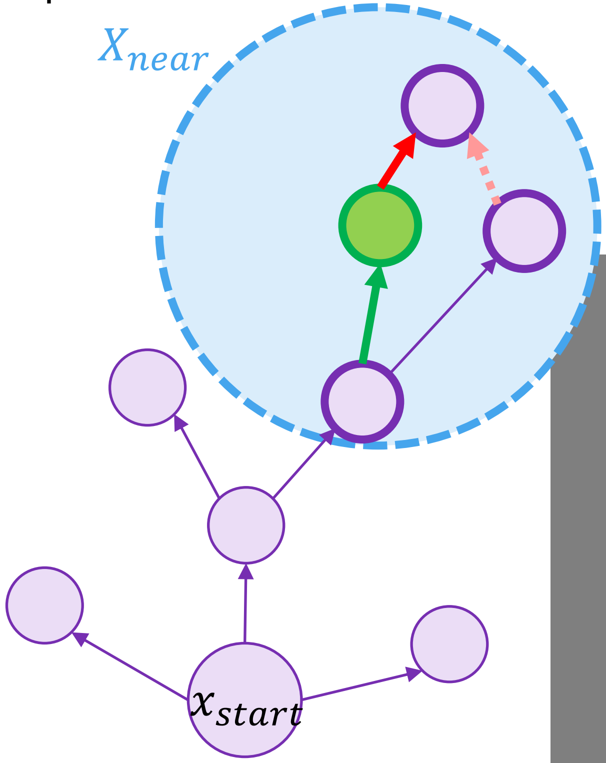
- Check if the cost of the path is smaller when selecting a different parent for the new node in the near nodes sets.



找出代价最小的

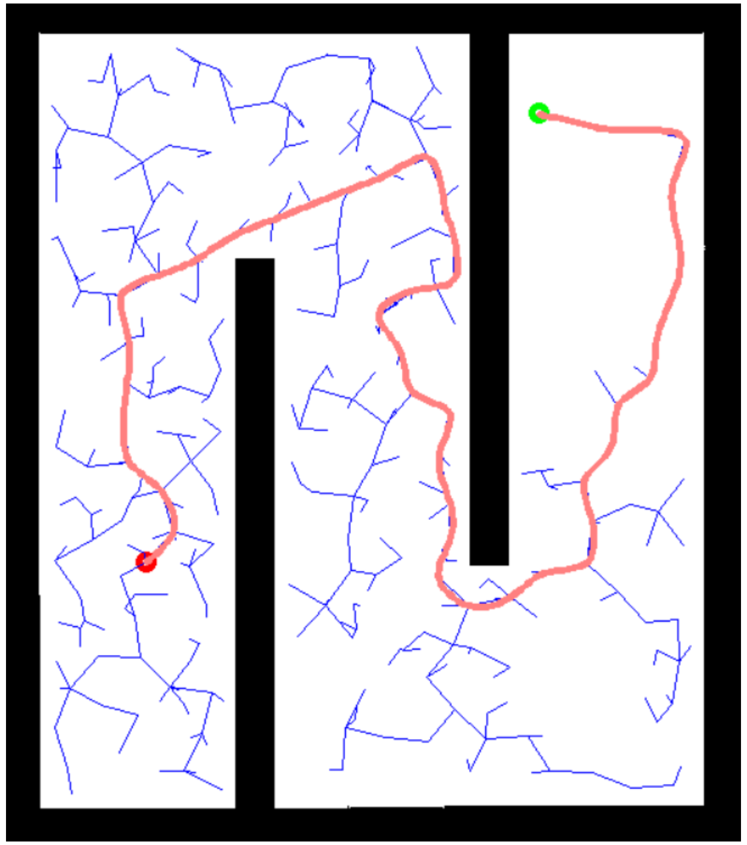
RRT* - Re-Wire Process

- Check if the cost of path is smaller when selecting the new node as the parent of the node in near nodes set.

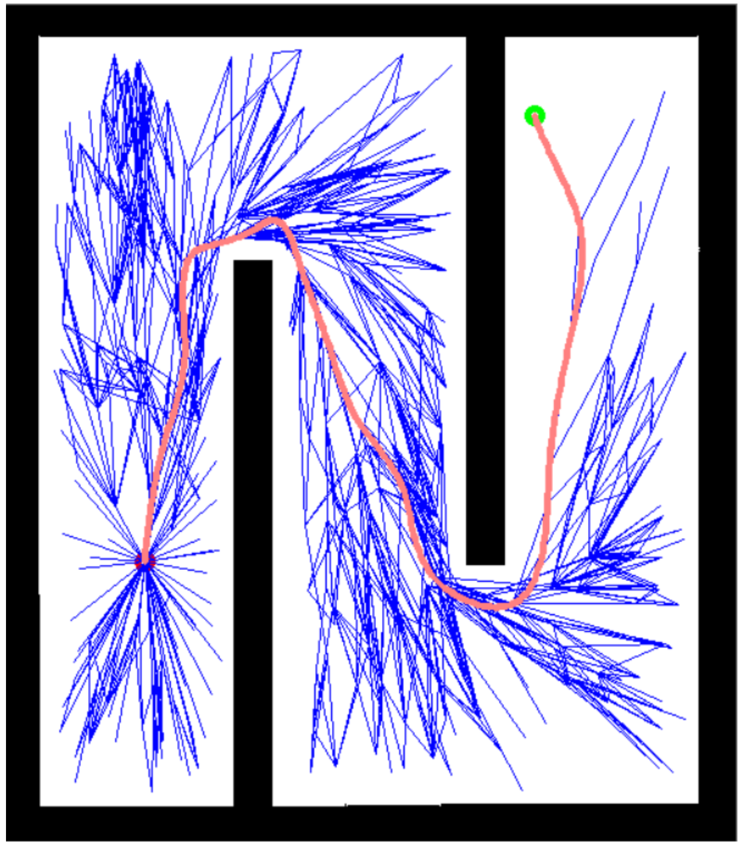


Comparison of RRT and RRT*

RRT



RRT* 优化采样范围

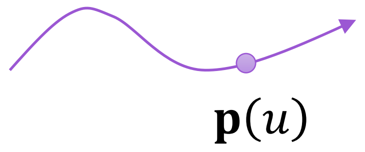


Curve Interpolation

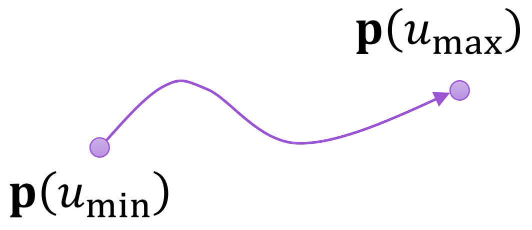
表示

Representation of Curves

- Explicit Representation: 隐式
  - $y = f(x)$ or $x = g(y)$ 没有办法表示所有的点。垂直方向的直线就不能用y=f(x)来表示
  - No guarantee that either form exists for a given curve (e.g., vertical line and circle)
- Implicit Representation:
  - $f(x, y) = 0$ 以特定的函数表示
    - Line: $ax + by + c = 0$
    - Circle: $x^2 + y^2 - r^2 = 0$
  - Does represent all lines and circles, but difficult to obtain all points on the curve
- Parametric Representation:
  - Curve: $\mathbf{p}(u) = [x(u) \quad y(u)]^T$
    - $\frac{d\mathbf{p}(u)}{du} = \left[\frac{dx(u)}{du} \quad \frac{dy(u)}{du}\right]^T$ 每个点对u做偏微分就可以得到这个点的切线方向，即这个点的速度大小
      - the velocity with which the curve is traced out (tangent direction at point **p**)
  - Most flexible and robust form, but not unique



Parametric Cubic Polynomial Curves



- We can use polynomial functions to form curves:
  - $\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \cdots \boldsymbol{c_n}u^n$
  - degree of freedom: n+1 实际考虑三次
  - cubic polynomial curve: $\mathbf{p}(u) = \boldsymbol{c_0} + \boldsymbol{c_1}u + \boldsymbol{c_2}u^2 + \boldsymbol{c_3}u^3 = \mathbf{u}^T\mathbf{c}$
    - Low freedom, but sufficient to produce the desired shape in a small region
  - $0 \le u \le 1$
- The problem is how to efficiently find out the coefficient $\boldsymbol{c_i}$
- Least square curve fitting:

给定一个点可以得到两个方程

$$x(u) = c_{x0} + c_{x1}\,u + c_{x2}u^2 + c_{x3}u^3$$
$$y(u) = c_{y0} + c_{y1}\,u + c_{y2}u^2 + c_{y3}u^3$$

$\Rightarrow$ Need 4 points to solve 8 unknowns

$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$$

$$\mathbf{c}_k = \begin{bmatrix} c_{kx} \\ c_{ky} \end{bmatrix}$$

Least Square Curve Fitting

- Given 4 control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$, and $\mathbf{p}_3$
  - Assume the 4 points are with equally spaced values *u*:

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix}$$

四个已知坐标点

$$\left.\begin{aligned} \mathbf{p}_0 &= \mathbf{p}(0) = \mathbf{c}_0 \\ \mathbf{p}_1 &= \mathbf{p}\left(\frac{1}{3}\right) = \mathbf{c}_0 + \frac{1}{3}\mathbf{c}_1 + \left(\frac{1}{3}\right)^2 \mathbf{c}_2 + \left(\frac{1}{3}\right)^3 \mathbf{c}_3 \\ \mathbf{p}_2 &= \mathbf{p}\left(\frac{2}{3}\right) = \mathbf{c}_0 + \frac{2}{3}\mathbf{c}_1 + \left(\frac{2}{3}\right)^2 \mathbf{c}_2 + \left(\frac{2}{3}\right)^3 \mathbf{c}_3 \\ \mathbf{p}_3 &= \mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 \end{aligned}\right\} \Rightarrow \mathbf{P} = \mathbf{A}\mathbf{c}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{3} & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \frac{2}{3} & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

nonsingular

$$\Big\downarrow \times A^{-1}$$

$$\mathbf{M}_I = \mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix}$$

Interpolating Geometry Matrix

$$\Rightarrow \mathbf{c} = \mathbf{M}_I \mathbf{P}$$

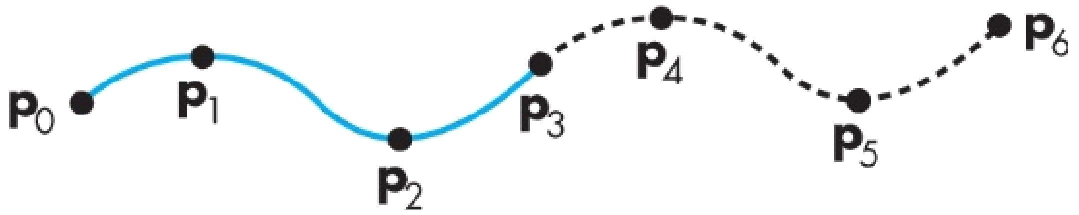
得到所有系数的值

$$\mathbf{p}(u) = \mathbf{u}^T \mathbf{c} = \mathbf{u}^T \mathbf{M}_I \mathbf{P}$$

原来cp维度是8，xy坐标分开表示，满足4维方程

Cubic Interpolating Curves

有多个离散坐标点时该如何？



可以先取出4个连续的点，画出小区段，再把小区段组合起来

- Rather than deriving a single interpolating curve of degree m for all the points, derive a set of cubic interpolating curves.
- If each segment is derived by letting u varying equally over the interval [0,1], then the matrix $\mathbf{M_I}$ is the same for each segment.
- Derivatives at the joint points will not be continuous.

相邻区段接起来有转折，因为没有考虑每个区段第一个点与前一个区段最后一个点的微分连续性

限制条件：每个点应该在曲线上；每个点的前后一二阶微分要连续；头尾端点曲率 为0

$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$$
$$\mathbf{p}'(u) = \mathbf{c}_1 + 2u\mathbf{c}_2 + 3u^2\mathbf{c}_3$$

Hermite Curves

- A Hermite curve is a curve for which the user provides:
  - The endpoints of the curve:

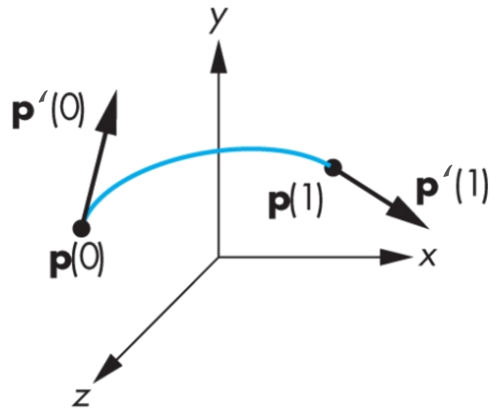
    $\mathbf{p}_0 = \mathbf{p}(0) = \mathbf{c}_0$

    $\mathbf{p}_3 = \mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3$
  - The derivatives of the curve at the endpoints:

    $\mathbf{p}'_0 = \mathbf{p}'(0) = \mathbf{c}_1$

    $\mathbf{p}'_3 = \mathbf{p}'(1) = \mathbf{c}_1 + 2\mathbf{c}_2 + 3\mathbf{c}_3$

头尾两个点的位置和一阶微分



$$\mathbf{Q} = \begin{bmatrix}\mathbf{p}_0\\ \mathbf{p}_3\\ \mathbf{p}'_0\\ \mathbf{p}'_3\end{bmatrix} = \begin{bmatrix}1&0&0&0\\1&1&1&1\\0&1&0&0\\0&1&2&3\end{bmatrix}\mathbf{c} \qquad \mathbf{M_H} = \overset{A^{-1}}{\begin{bmatrix}1&0&0&0\\1&1&1&1\\0&1&0&0\\0&1&2&3\end{bmatrix}^{-1}} = \begin{bmatrix}1&0&0&0\\0&0&1&0\\-3&3&-2&-1\\2&2&1&1\end{bmatrix}$$

$$\mathbf{c} = \mathbf{M_H Q}$$

Hemite Geometry Matrix

$$\mathbf{p}(u) = \mathbf{u}^T\mathbf{c} = \mathbf{u}^T\mathbf{M}_H\mathbf{Q}$$

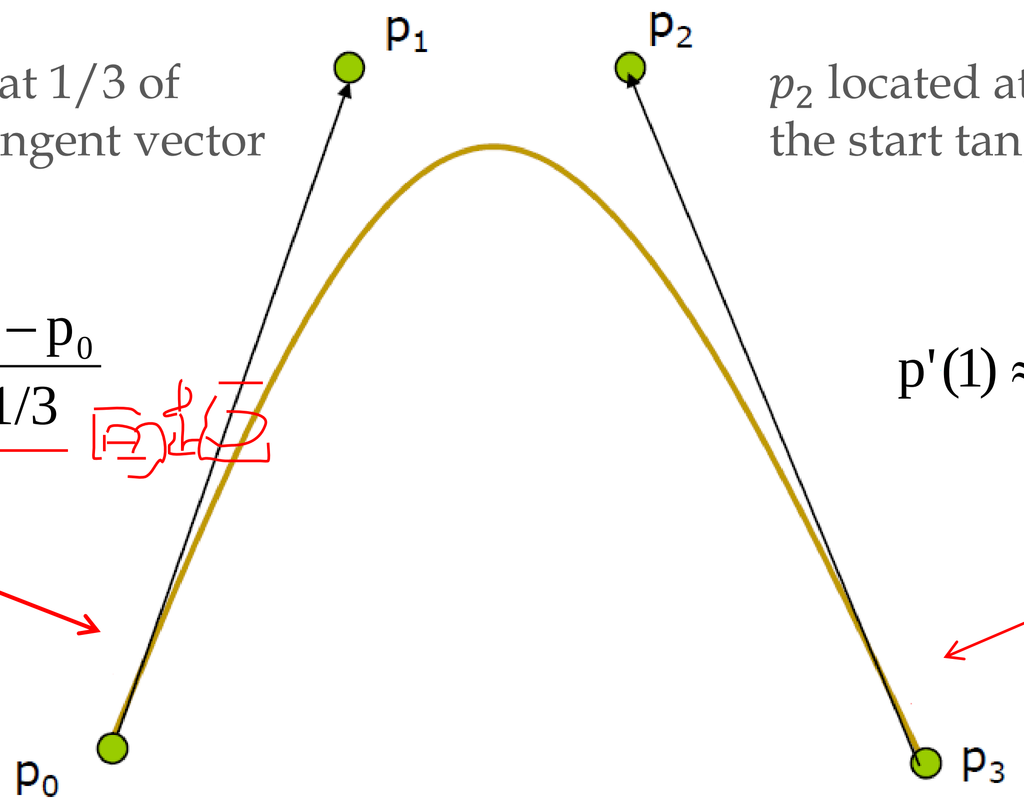
相邻区段就可以平滑串接

只知道四个的的位置，而不知道一阶微分，该如何？

Bezier Curves

- Two control points define endpoints, and two points control the tangents.

通过p1和p2去估算p0和p3的一阶微分值



p_1 located at $1/3$ of the start tangent vector

p_2 located at $2/3$ of the start tangent vector

$$p'(0) \approx \frac{p_1 - p_0}{\underline{1/3}}$$ [illegible]

$$p'(1) \approx \frac{p_3 - p_2}{1/3}$$

slope p′(0)

slope p′(1)

$$\mathbf{p}(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$$

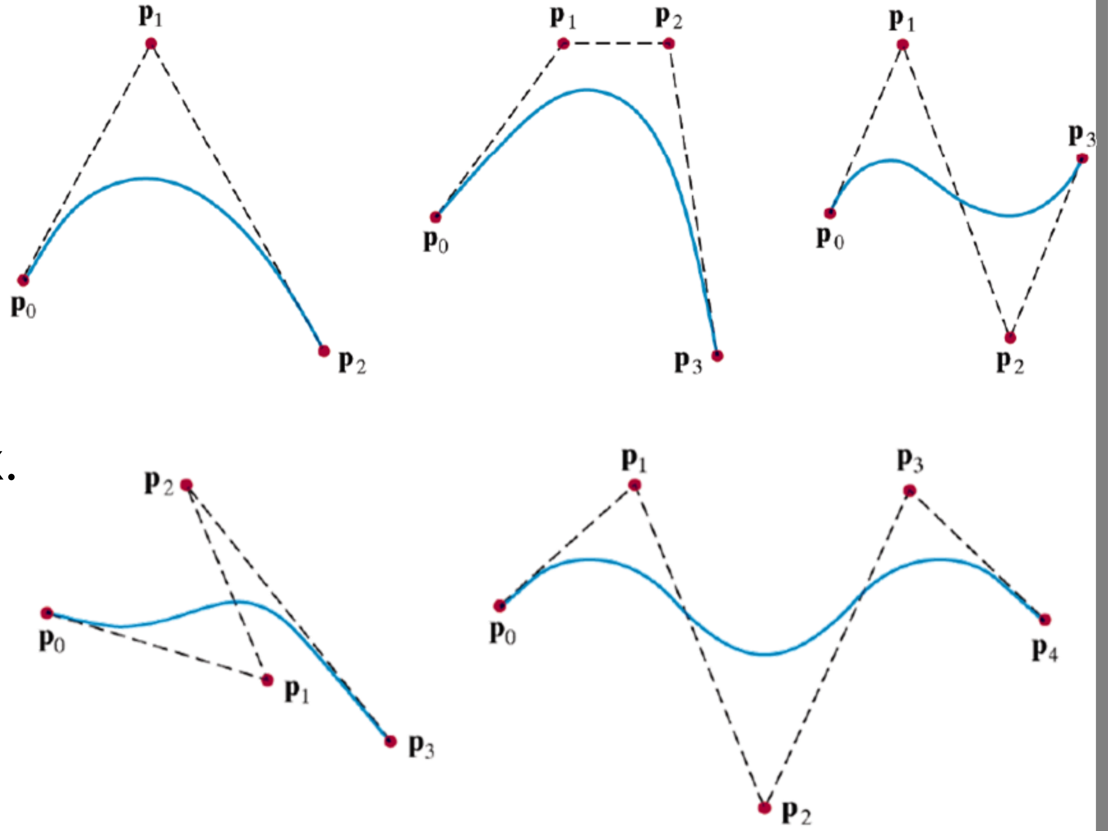
Bezier Curves (Cont.)

- The endsite conditions are the same
  - $P(0) = P_0 = c_0$
  - $P(1) = P_3 = c_0 + c_1 + c_2 + c_3$
- Approximating derivative conditions
  - $P'(0) = 3(P_1 - P_0) = c_1$ 近似值
  - $P'(1) = 3(P_3 - P_2) = c_1 + 2c_2 + 3c_3$
- Replacing the original Hermite matrix.

$$\mathbf{c} = \mathbf{M}_B\mathbf{P} \qquad \mathbf{M}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Bezier Geometry Matrix

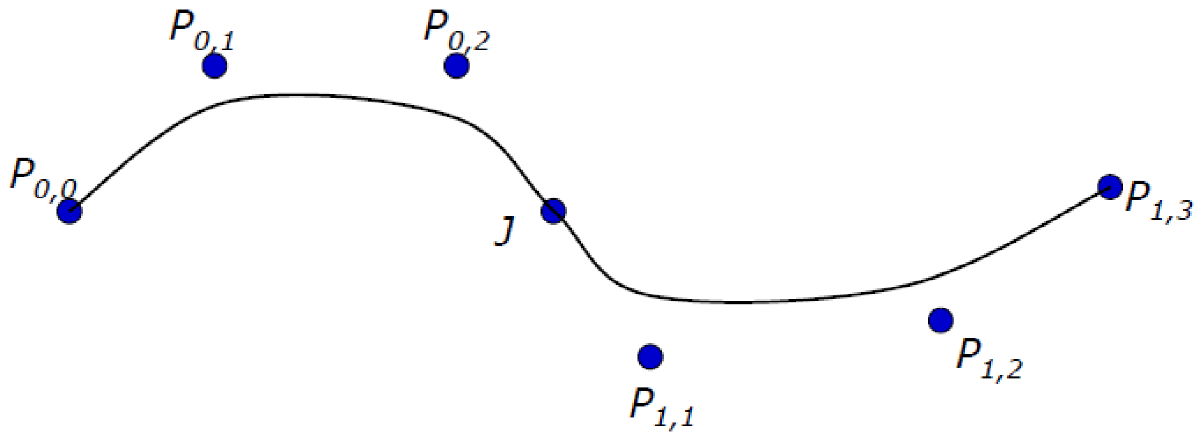
$$\mathbf{p}(u) = \mathbf{u}^T\mathbf{c} = \mathbf{u}^T\mathbf{M}_B\mathbf{P}$$



Bezier Continuity

仍然存在接点处不连续

- We can make a long curve by concatenating multiple short Bezier curves.



- How to keep the continuity?

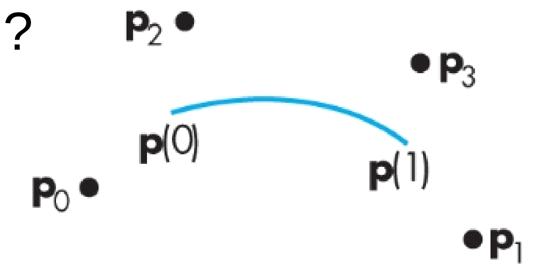
Cubic B-spline Curves

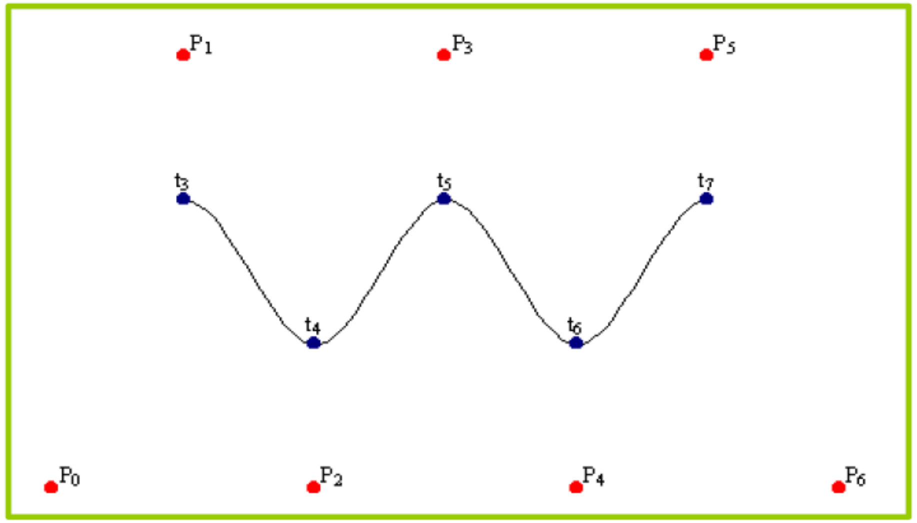
解决一阶微分连续性问题

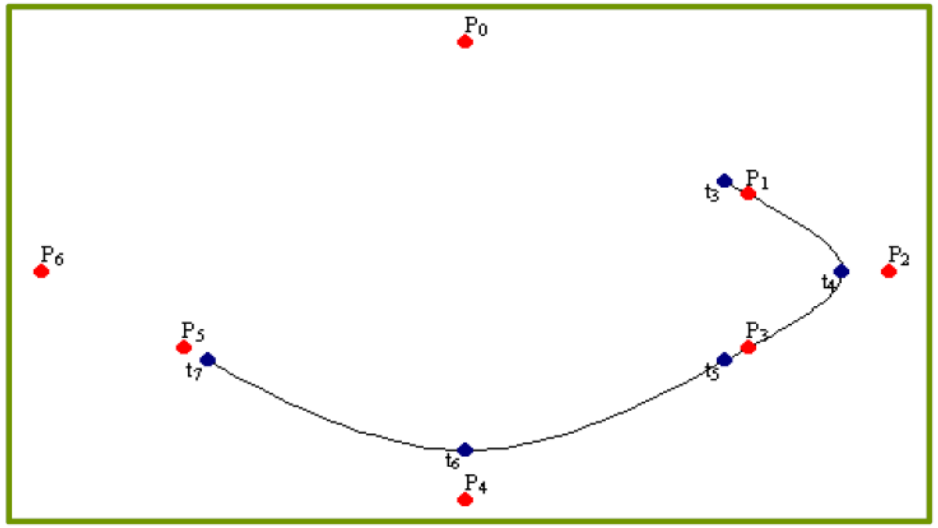
也可以满足二阶微分连续性

- How to reach both C^2 continuity and local controllability ?
  - Slightly loose the endpoint constraints. 放开端点的位置限制
  - B-splines do not interpolate any of control points.

不需要通过控制端点







Figures from CG lecture note, U. Virginia

B-spline Curves (Cont.)

$$\underline{\mathbf{p}(0) = \mathbf{q}(1)} = \frac{1}{6}(\mathbf{p}_{i-2} + 4\mathbf{p}_{i-1} + \mathbf{p}_i)$$

$$\underline{\mathbf{p}'(0) = \mathbf{q}'(1)} = \frac{1}{2}(\mathbf{p}_i - \mathbf{p}_{i-2})$$

(for symmetric)

Since $\mathbf{p}(u) = \mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 u^2 + \mathbf{c}_3 u^3 = u^T \mathbf{c}$

$$\mathbf{p}(0) = \mathbf{c}_0 = \frac{1}{6}(\mathbf{p}_{i-2} + 4\mathbf{p}_{i-1} + \mathbf{p}_i)$$

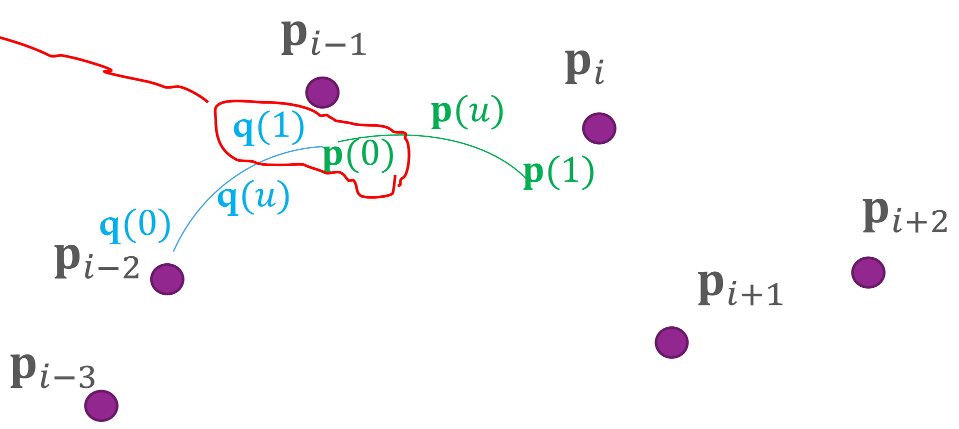
$$\mathbf{p}'(0) = \mathbf{c}_1 = \frac{1}{2}(\mathbf{p}_i - \mathbf{p}_{i-2})$$

$$\mathbf{p}(1) = \mathbf{c}_0 + \mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 = \frac{1}{6}(\mathbf{p}_{i-1} + 4\mathbf{p}_i + \mathbf{p}_{i+1})$$

$$\mathbf{p}'(1) = \mathbf{c}_1 + 2\mathbf{c}_2 + 3\mathbf{c}_3 = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$$

$\Rightarrow \mathbf{P} = \mathbf{A}\mathbf{c}$

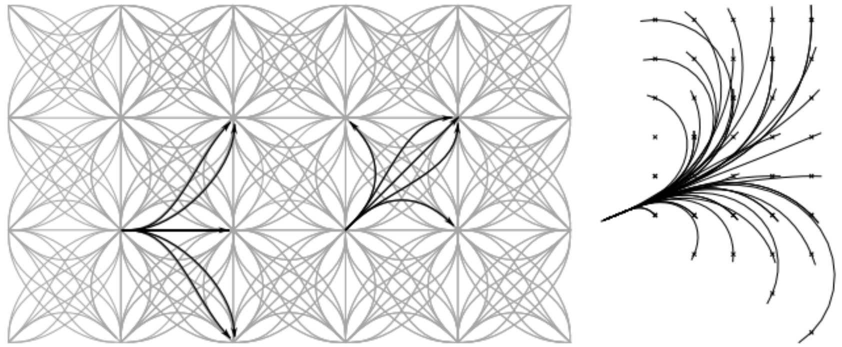
$$\mathbf{M}_S = \mathbf{A}^{-1} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad \Rightarrow \mathbf{c} = \mathbf{M}_S\mathbf{P}$$



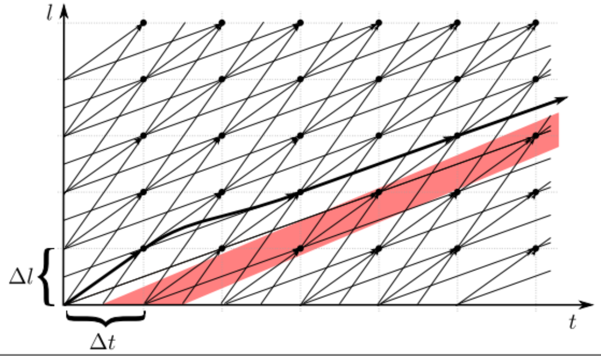
Trajectory Planning

State Lattices Planning

- Originally, the concept of state lattice is based on the configuration space. The following image illustrates the nontemporal state lattice on 2D workspace which considers the curvature.



- Later, the space-time manifold is constructed by combining the configuration space and time, which can be used in the dynamic environment.



State Lattices Planning

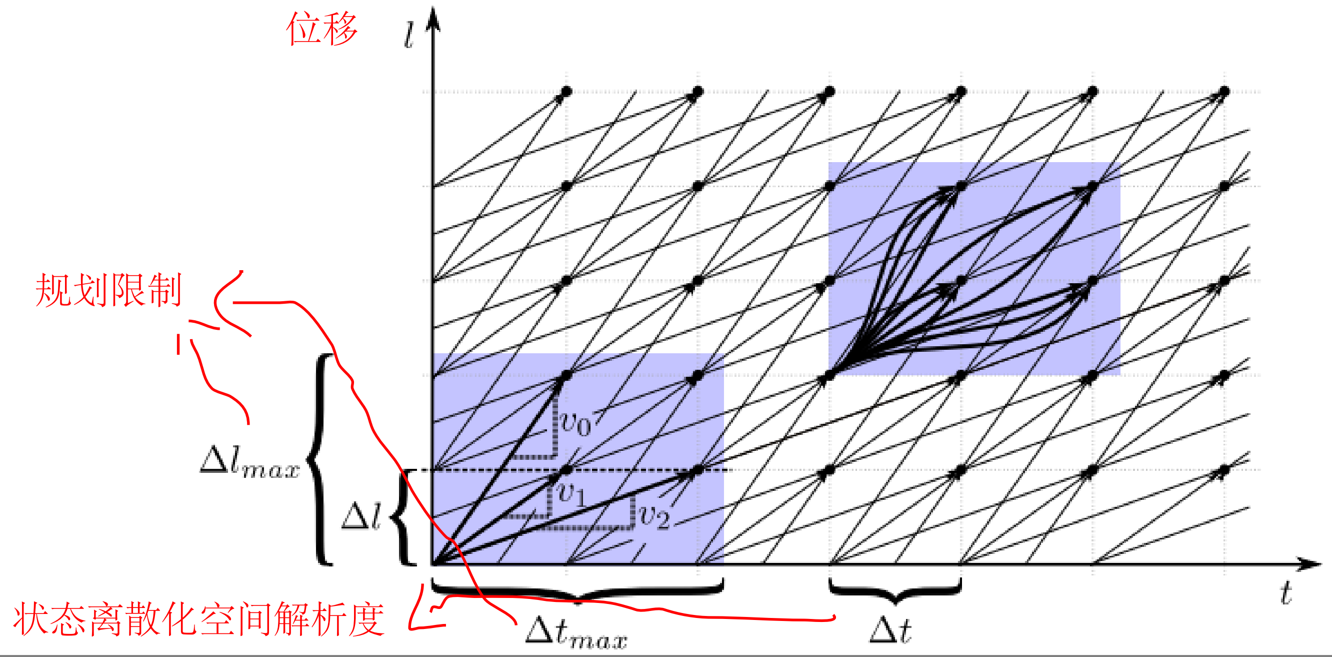
- State Lattice Planning Workflow

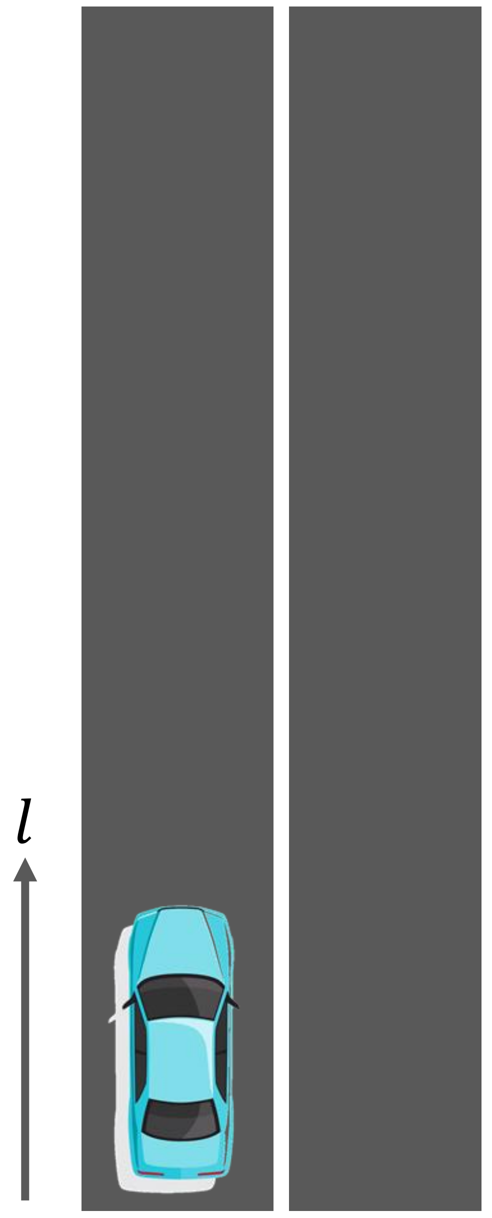
1. Sample the candidate trajectories. 根据离散的空间轨迹进行采样
2. Compute the cost by pre-defined cost function.
   根据定义的代价函数计算每一条轨迹的代价
3. Check the collision and Motion constraints and select the low-cost trajectory as output.
   确认所有轨迹是否符合运动避障

Spatiotemporal State Lattices

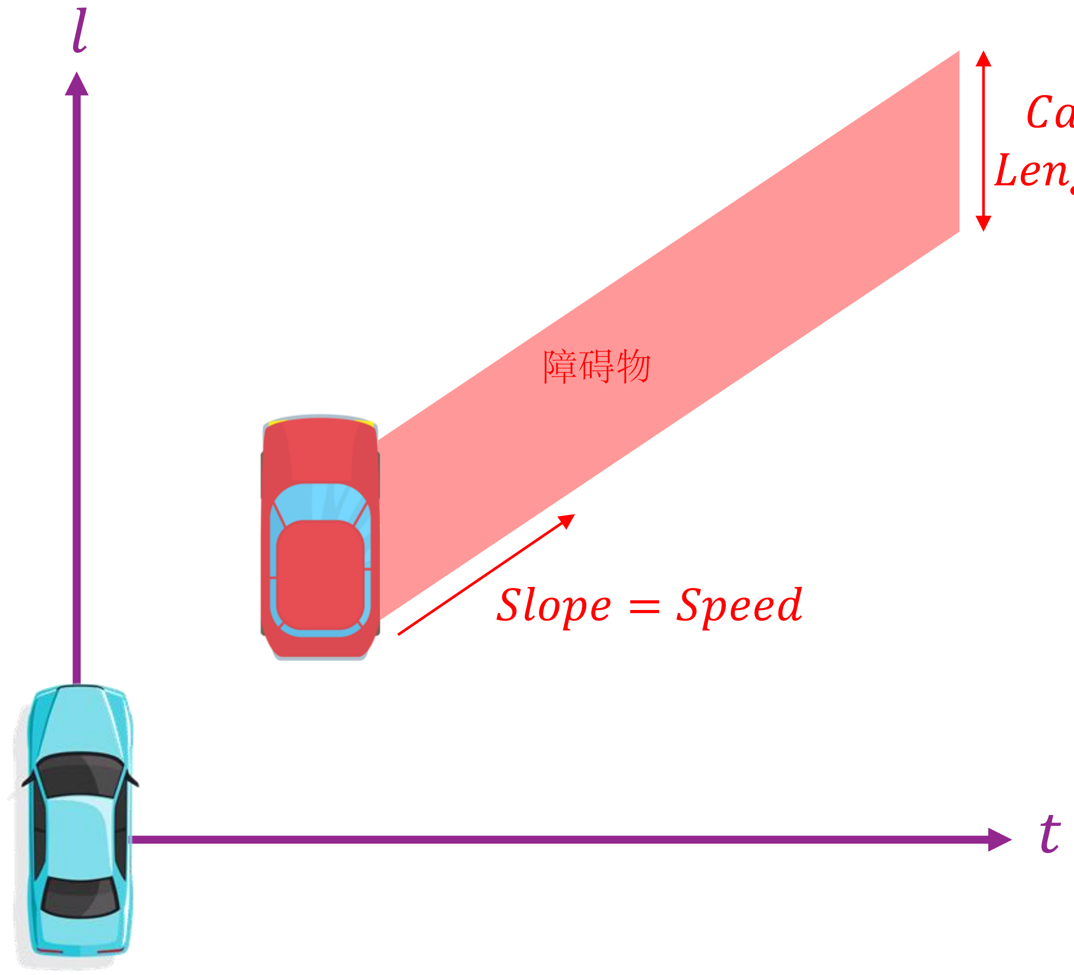
时空状态空间

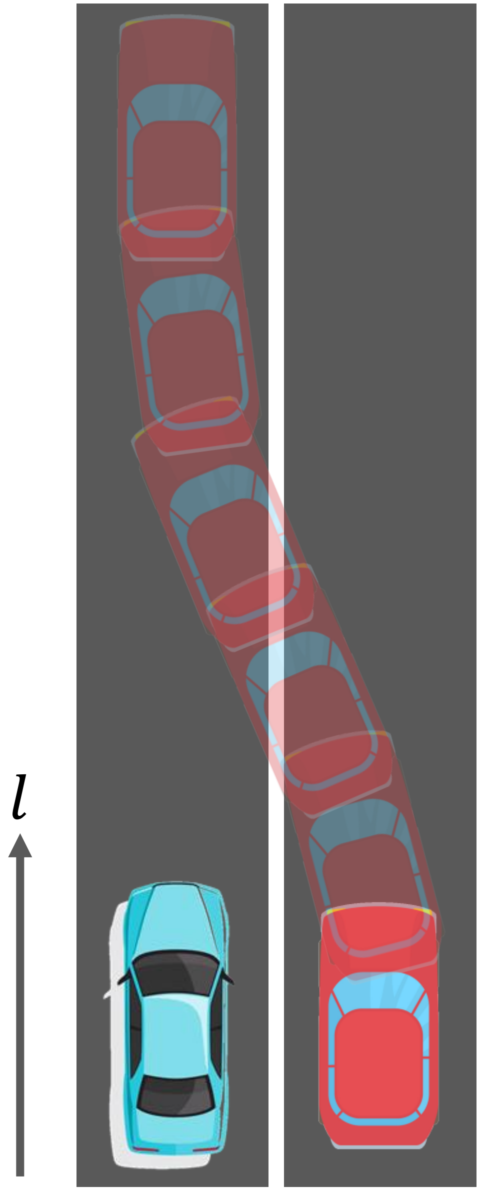
- Consider a spatiotemporal state lattice over a one dimensional workspace.
- $\Delta l, \Delta t$: spatial and temporal resolution.
- $\Delta l_{max}, \Delta t_{max}$: spatial and temporal constraints.



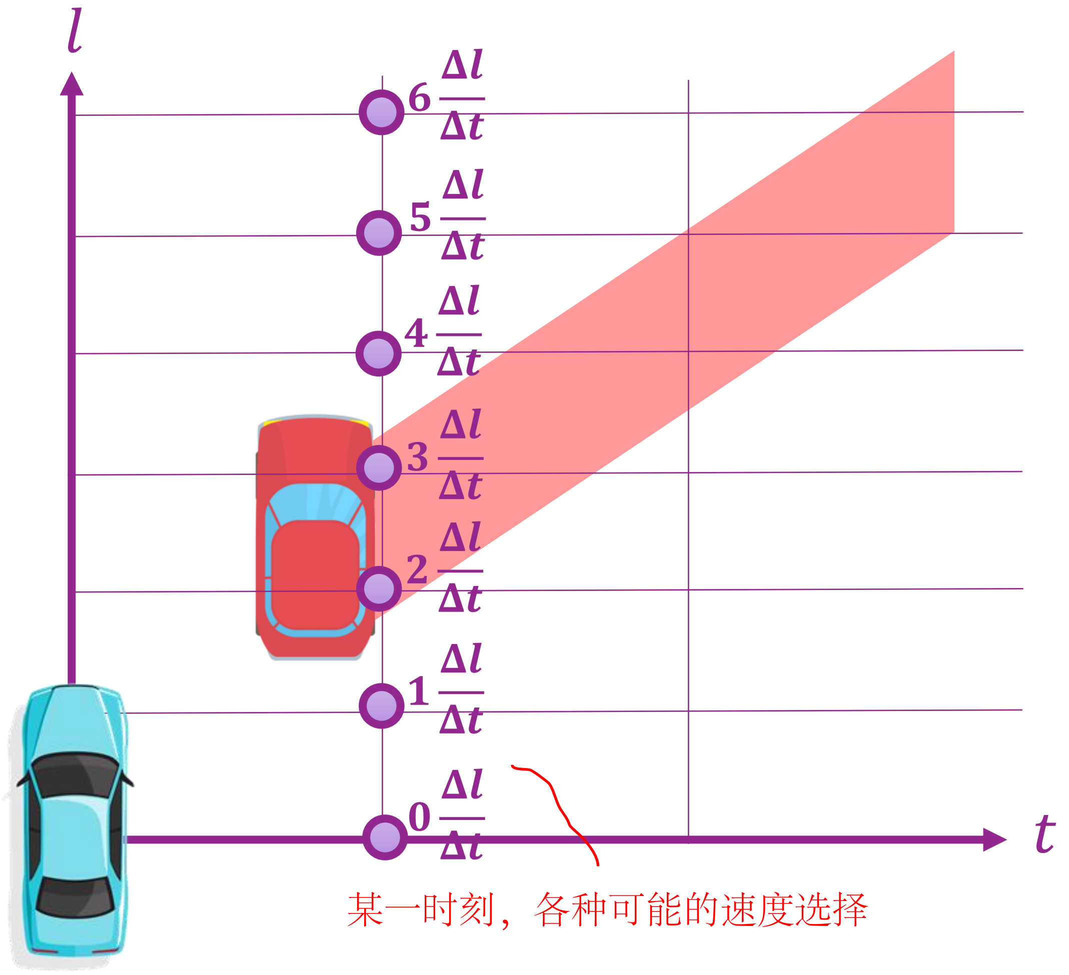


Spatiotemporal State Lattices



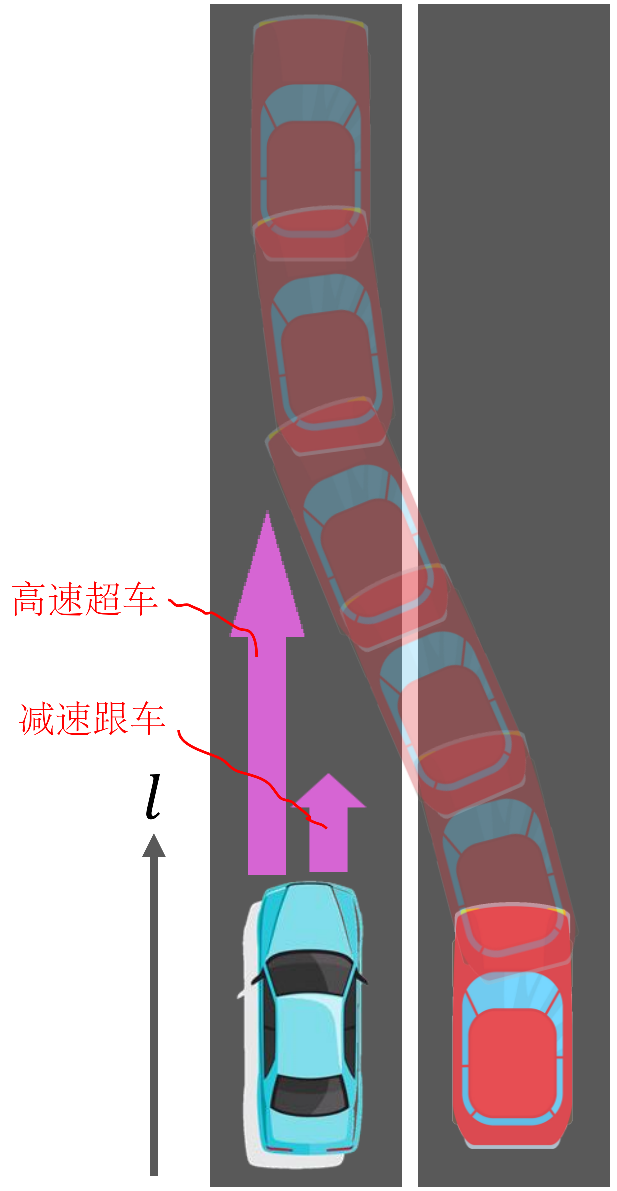


Spatiotemporal State Lattices

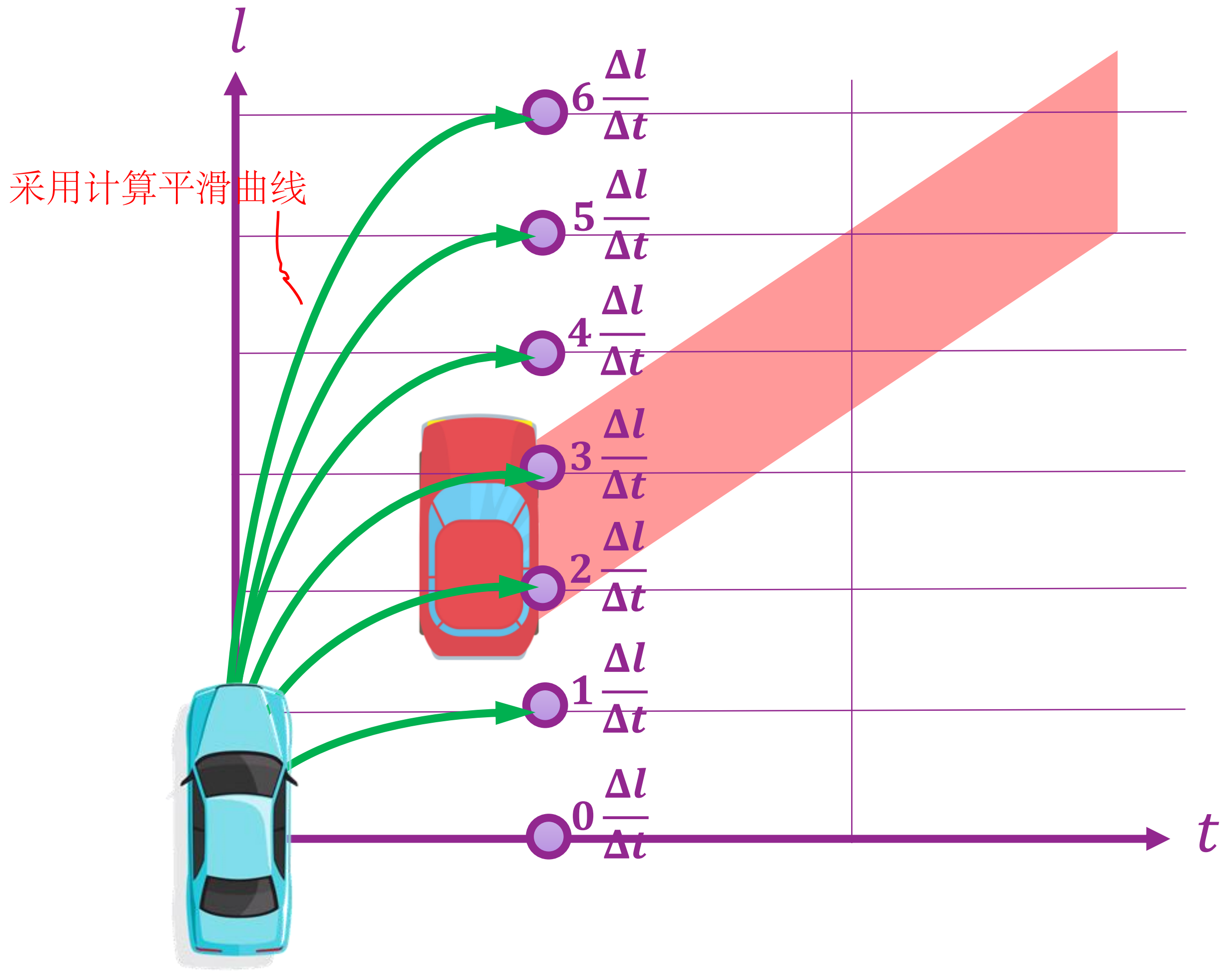


l
$6\frac{\Delta l}{\Delta t}$
$5\frac{\Delta l}{\Delta t}$
$4\frac{\Delta l}{\Delta t}$
$3\frac{\Delta l}{\Delta t}$
$2\frac{\Delta l}{\Delta t}$
$1\frac{\Delta l}{\Delta t}$
$0\frac{\Delta l}{\Delta t}$
t

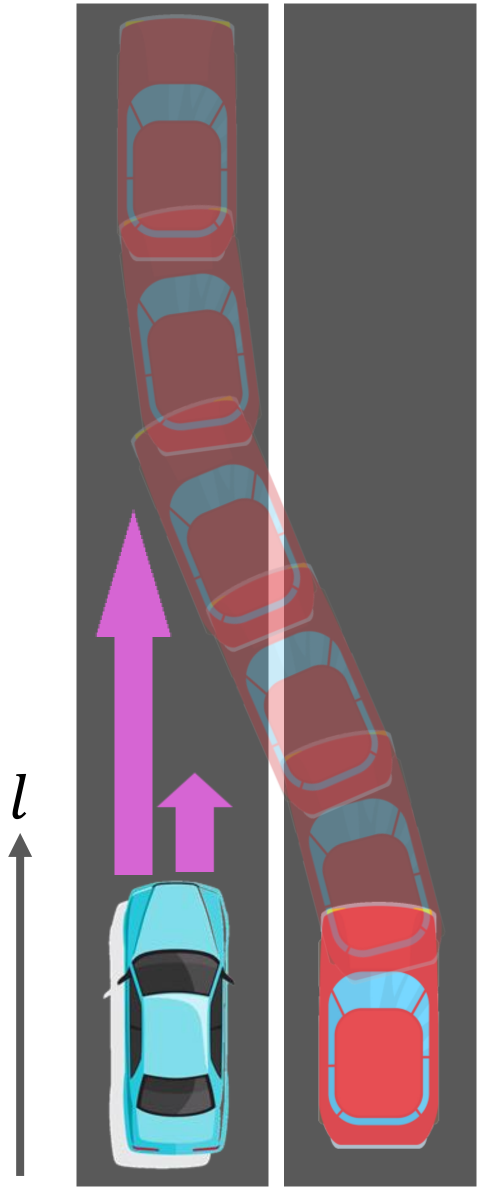
某一时刻，各种可能的速度选择



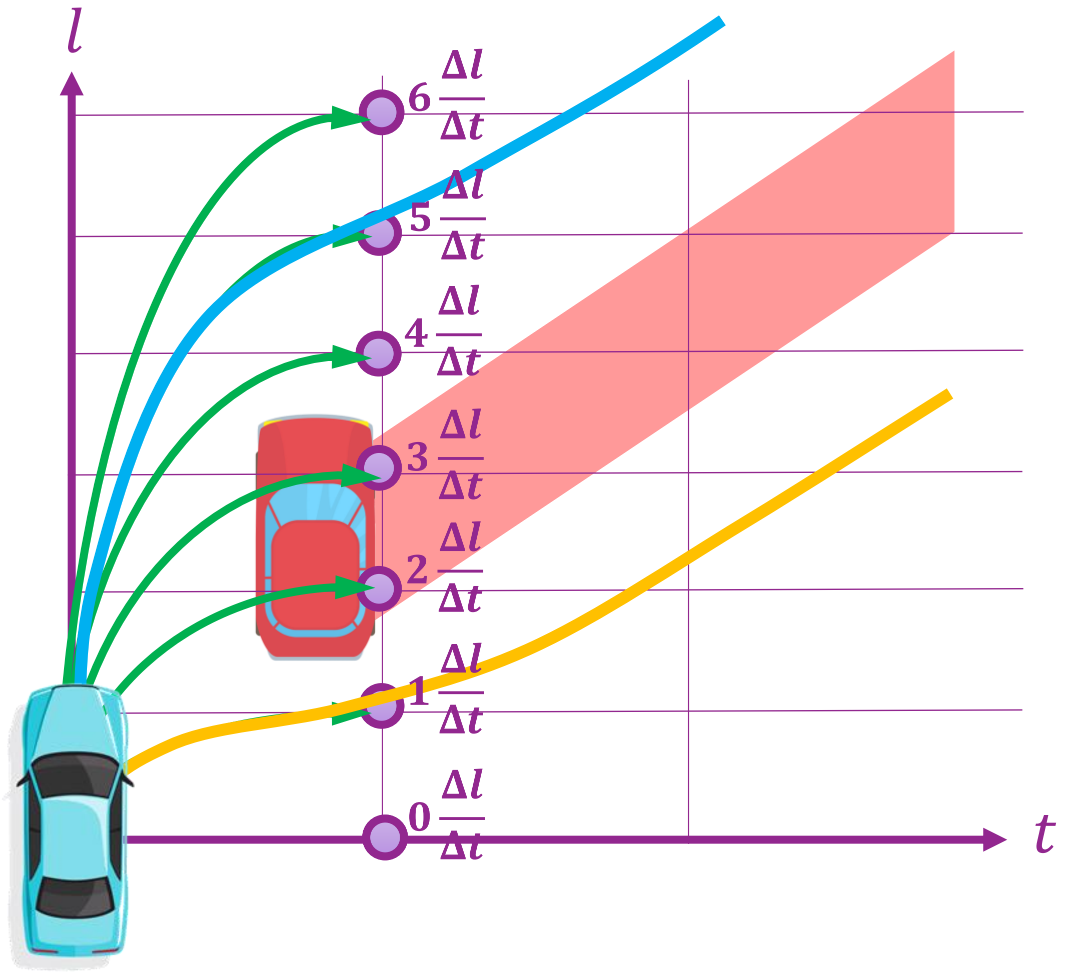
Spatiotemporal State Lattices

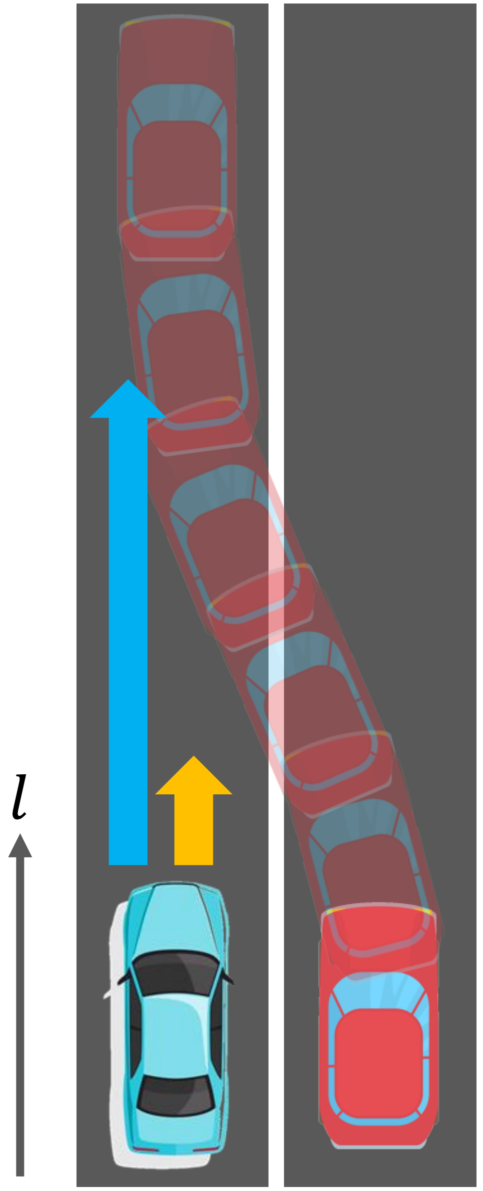


l
t
采用计算平滑曲线
$6\frac{\Delta l}{\Delta t}$
$5\frac{\Delta l}{\Delta t}$
$4\frac{\Delta l}{\Delta t}$
$3\frac{\Delta l}{\Delta t}$
$2\frac{\Delta l}{\Delta t}$
$1\frac{\Delta l}{\Delta t}$
$0\frac{\Delta l}{\Delta t}$

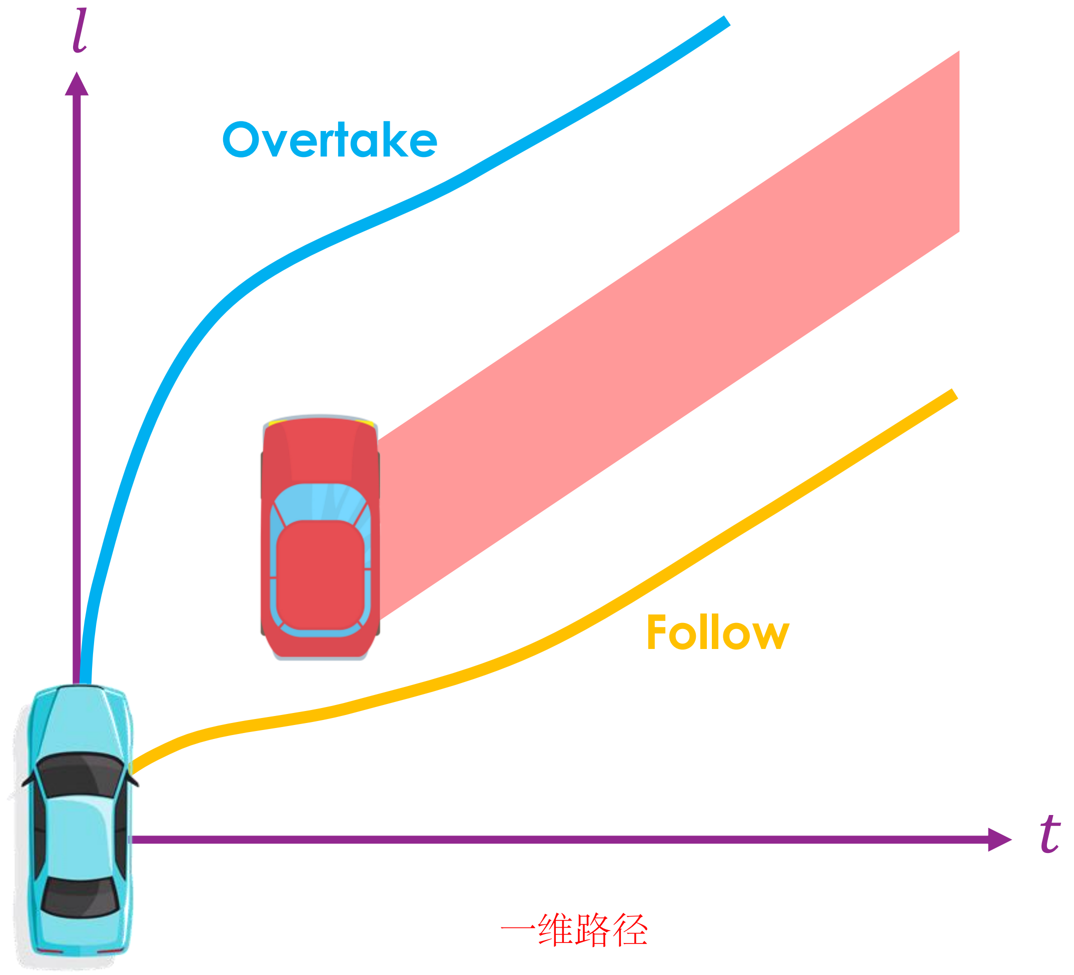


Spatiotemporal State Lattices

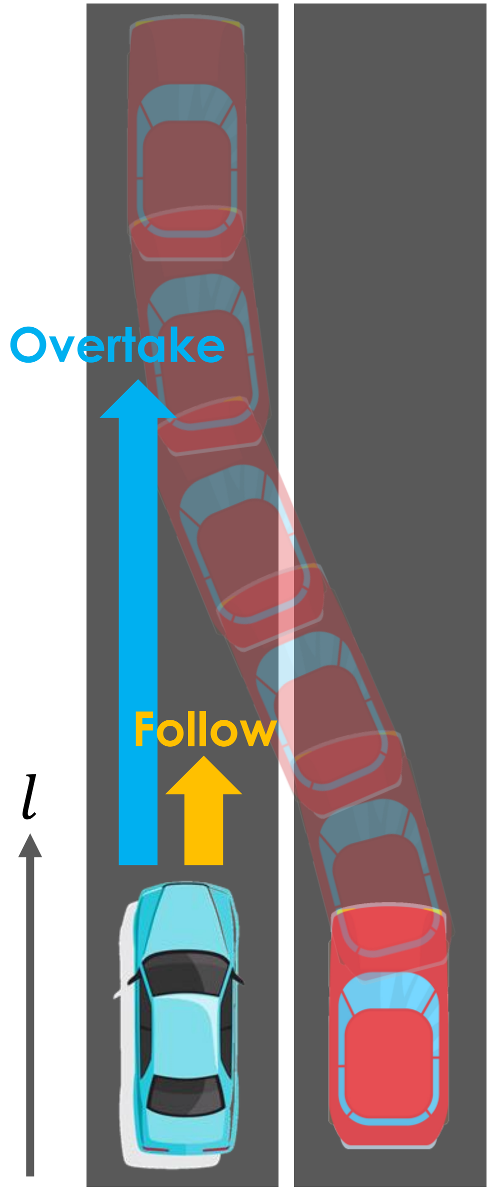

l
$6\frac{\Delta l}{\Delta t}$
$5\frac{\Delta l}{\Delta t}$
$4\frac{\Delta l}{\Delta t}$
$3\frac{\Delta l}{\Delta t}$
$2\frac{\Delta l}{\Delta t}$
$1\frac{\Delta l}{\Delta t}$
$0\frac{\Delta l}{\Delta t}$
t



Spatiotemporal State Lattices



一维路径

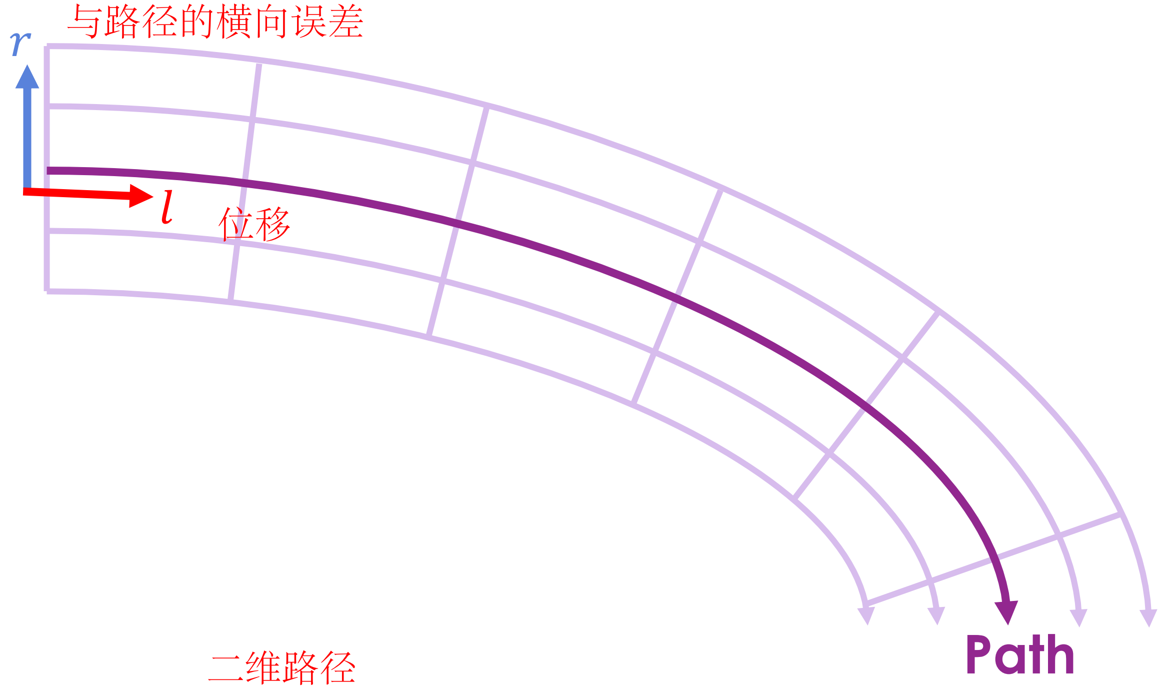


Frenet Coordinate

将车的位置投影到参考位置上

- Frenet coordinate projects the vehicle position onto the reference path.
- ***l*** represents the distance along the path and ***r*** represents the bias distance from the path.

明确计算出路径坐标系
与标准坐标系的转换



二维路径

Frenet Coordinate (Cont.)

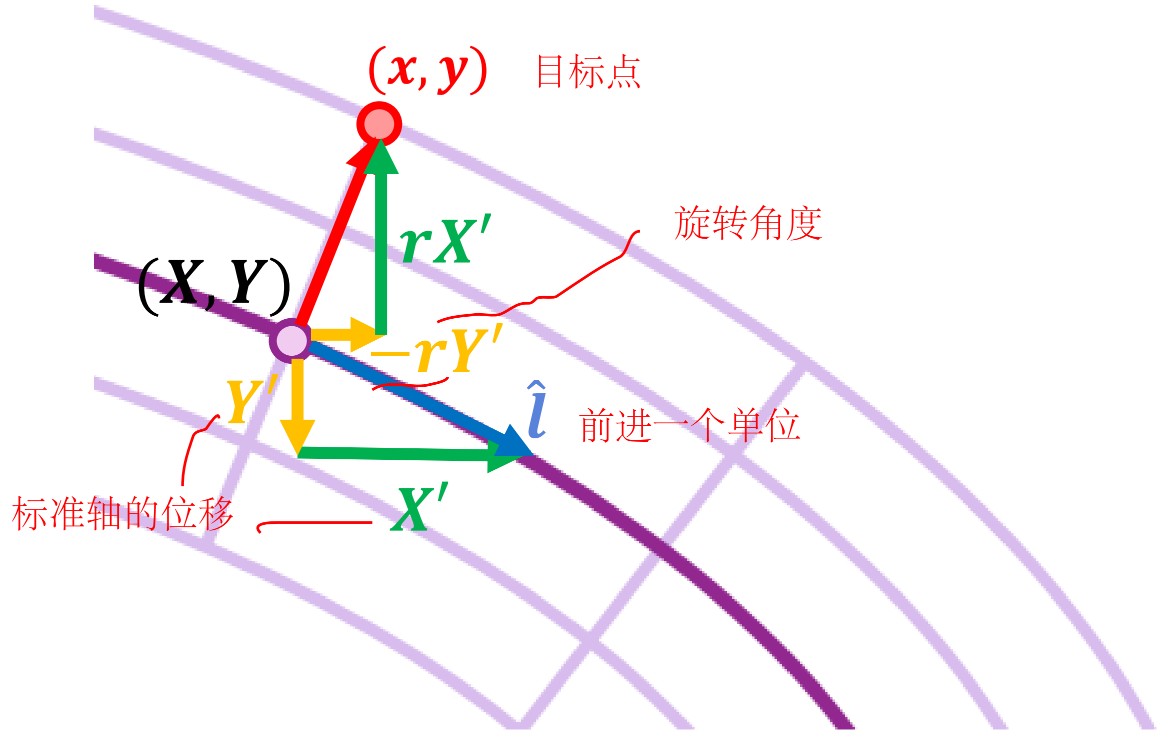
给定

- (l, r) : reparameterization parameters along the path. 路径坐标上的参数
- $(X_s(l), Y_s(l))$: points on the path.
- $(x(t), y(t))$: Standard Cartesian coordinate.

计算出

- Transform Function

$$x(t) = X(l) - rY'(l)$$
$$y(t) = Y(l) + rX'(l)$$



Frenet Coordinate (Cont.)

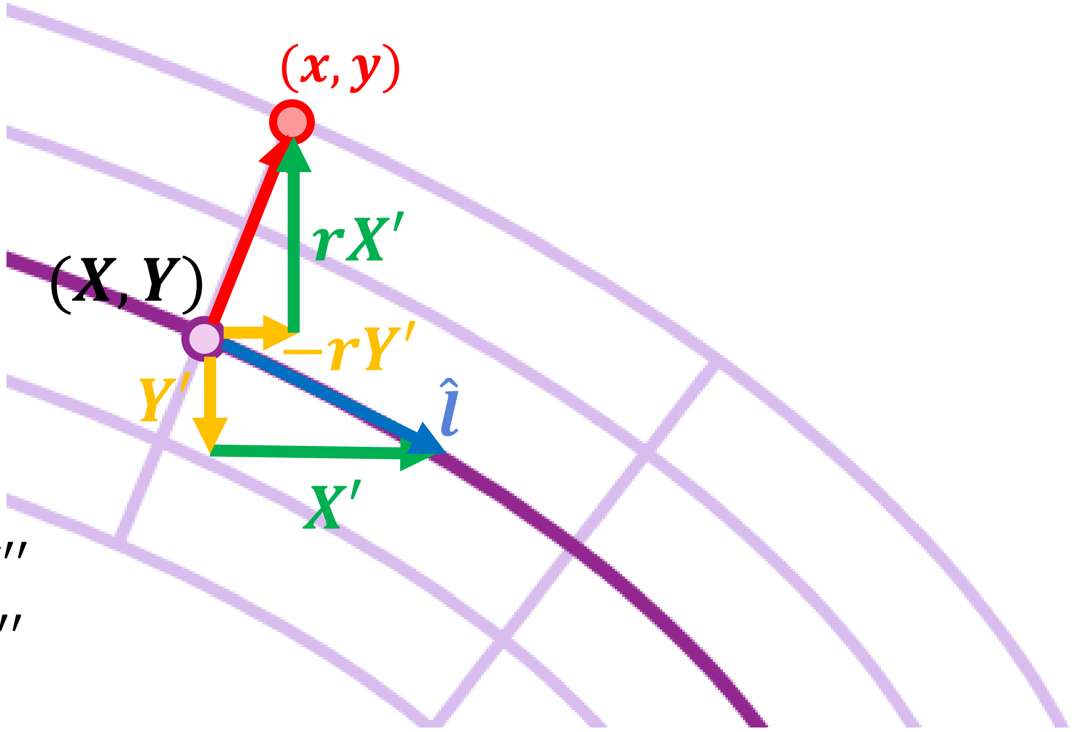
- (l, r) : reparameterization parameters along the path.
- $(X_s(l), Y_s(l))$: points on the path.
- $(x(t), y(t))$: Standard Cartesian coordinate.

- Differential Transform Function

$$\dot{x}(t) = \dot{l}X'(l) - \dot{r}Y(l) - r\dot{l}Y'(l)$$
$$\dot{y}(t) = \dot{l}Y'(l) + \dot{r}X(l) + r\dot{l}X'(l)$$

and

$$\ddot{x}(t) = \ddot{l}X' + \ddot{l}^2X'' - \ddot{r}Y - \left(2\dot{r}\dot{l} + r\ddot{l}\right)Y' - \dot{r}\dot{l}^2Y''$$
$$\ddot{y}(t) = \ddot{l}Y' + \ddot{l}^2Y'' - \ddot{r}\mathrm{X} - \left(2\dot{r}\dot{l} + r\ddot{l}\right)\mathrm{X}' - \dot{r}\dot{l}^2\mathrm{X}''$$



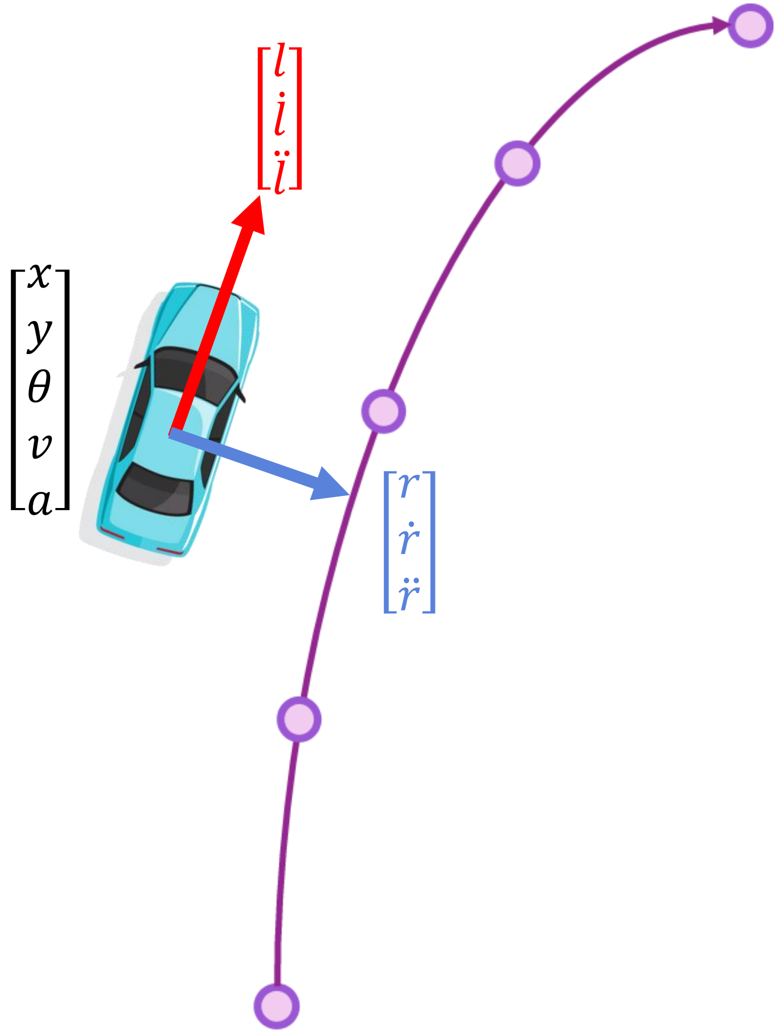
Trajectory Generation

- Vehicle States $[x, y, \theta, v, a]$

纵向状态

- Longitude States
  - l : Longitude distance
  - $\dot{l} = \frac{dl}{dt}$: Longitude speed
  - $\ddot{l} = \frac{d^2 l}{dt^2}$: Longitude acceleration

- Lateral States
  - r : Lateral offset
  - $\dot{r} = \frac{dr}{dt}$: Lateral speed
  - $\ddot{r} = \frac{d^2 r}{dt^2}$: Lateral acceleration



$\begin{bmatrix} l \\ \dot{l} \\ \ddot{l} \end{bmatrix}$
$\begin{bmatrix} x \\ y \\ \theta \\ v \\ a \end{bmatrix}$
$\begin{bmatrix} r \\ \dot{r} \\ \ddot{r} \end{bmatrix}$

Trajectory Generation (Cont.)

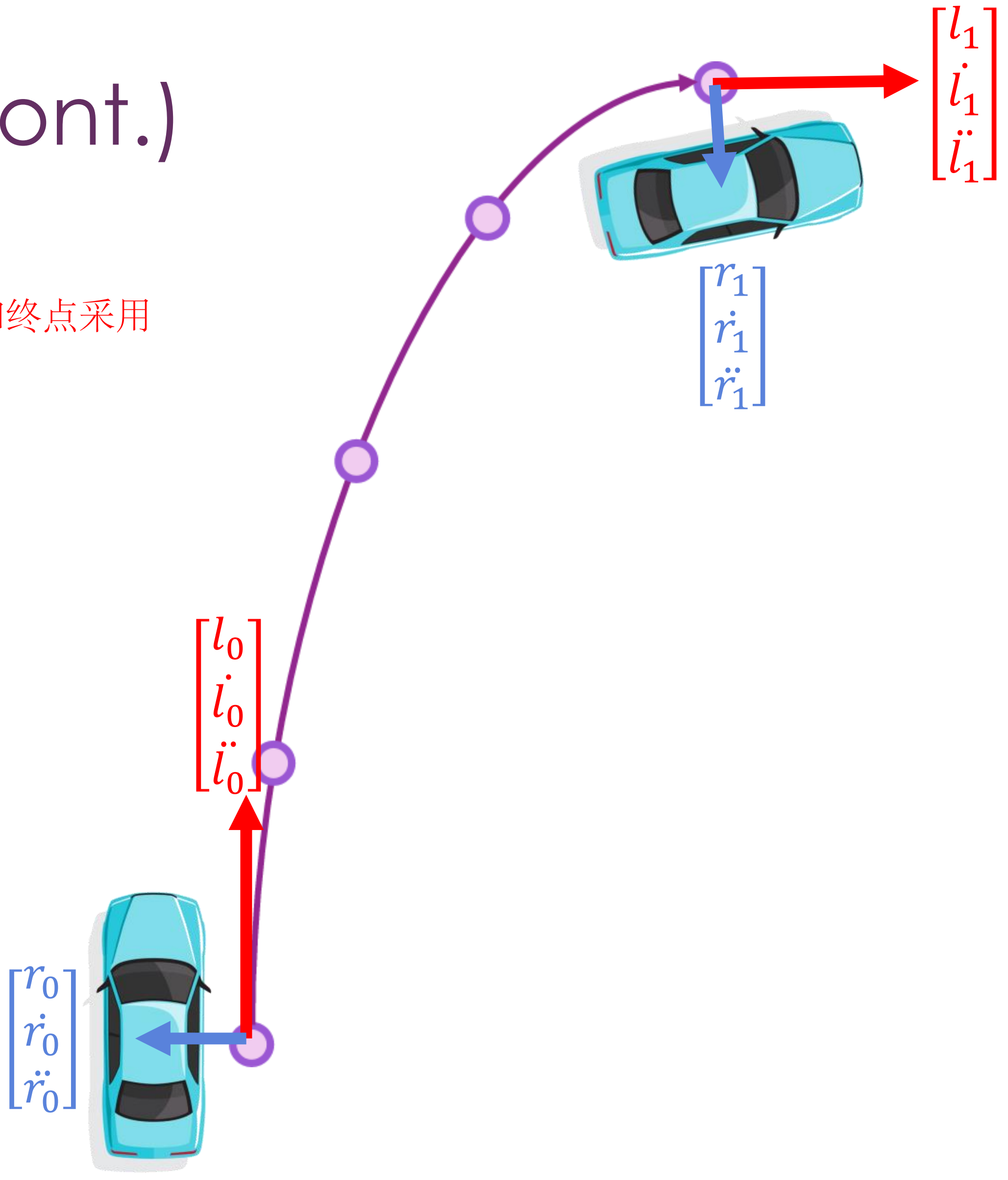
- Given initial state at time t_0

  $\{(r_0, \dot{r}_0, \ddot{r}_0), (l_0, \dot{l}_0, \ddot{l}_0)\}$

根据时间的起点和终点采用

- Given initial state at time t_1

  $\{(r_1, \dot{r}_1, \ddot{r}_1), (l_1, \dot{l}_1, \ddot{l}_1)\}$



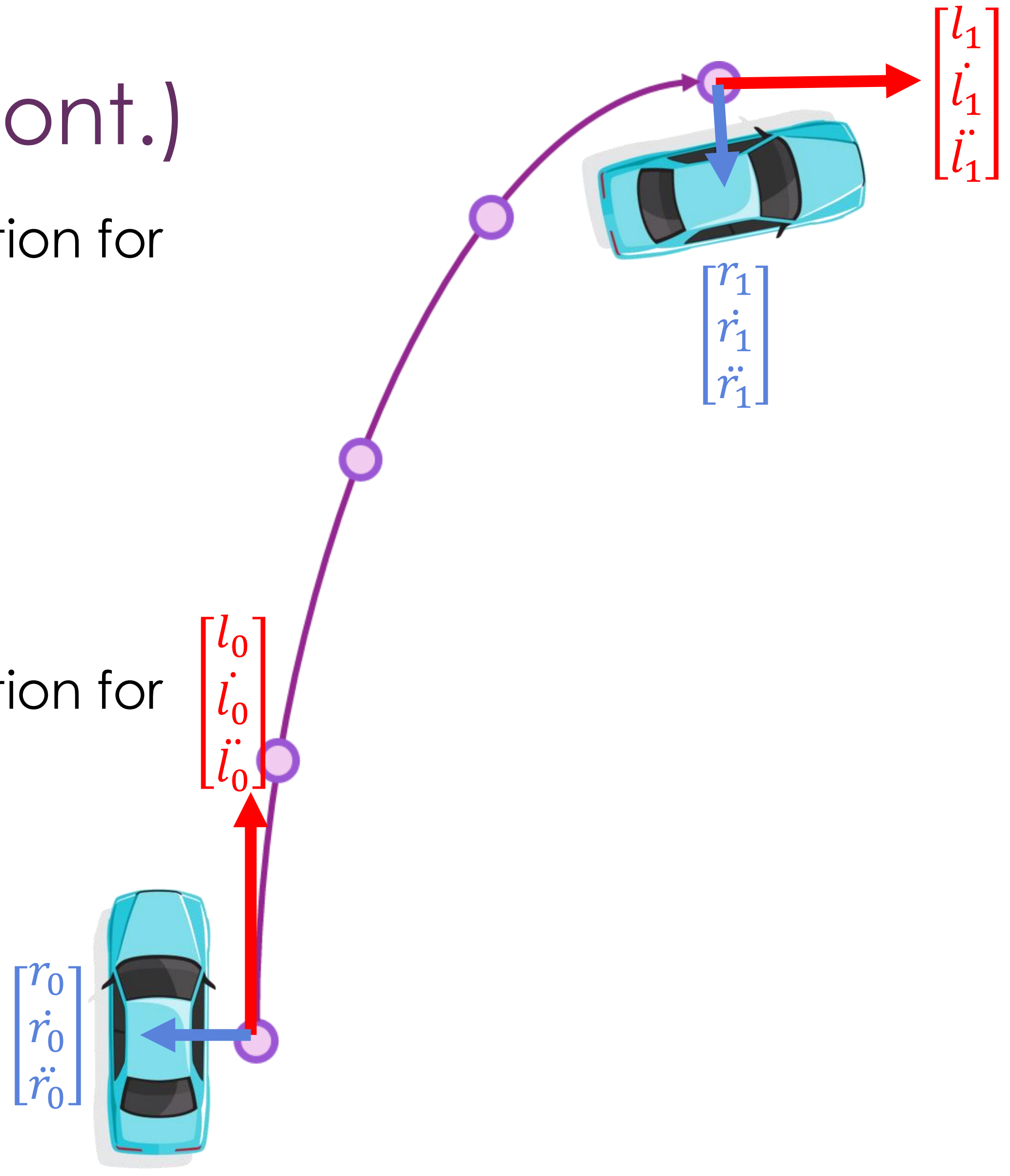
$\begin{bmatrix} l_1 \\ \dot{l}_1 \\ \ddot{l}_1 \end{bmatrix}$
$\begin{bmatrix} r_1 \\ \dot{r}_1 \\ \ddot{r}_1 \end{bmatrix}$
$\begin{bmatrix} l_0 \\ \dot{l}_0 \\ \ddot{l}_0 \end{bmatrix}$
$\begin{bmatrix} r_0 \\ \dot{r}_0 \\ \ddot{r}_0 \end{bmatrix}$

Trajectory Generation (Cont.)

终点作为边界条件

- Set boundary condition of curve function for fitting the longitude trajectory **l(t)**.
  - $l(t_0) = l_0\,, l(t_1) = l_1$
  - $\dot{l}(t_0) = \dot{l_0}\,, \dot{l}(t_1) = \dot{l_1}$
  - $\ddot{l}(t_0) = \ddot{l_0}\,, \ddot{l}(t_1) = \ddot{l_1}$

- Set boundary condition of curve function for fitting the lateral trajectory **r(t)**.
  - $r(t_0) = r_0\,, r(t_1) = r_1$
  - $\dot{r}(t_0) = \dot{r_0}\,, \dot{r}(t_1) = \dot{r_1}$
  - $\ddot{r}(t_0) = \ddot{r_0}\,, \ddot{r}(t_1) = \ddot{r_1}$



$\begin{bmatrix} l_1 \\ \dot{l_1} \\ \ddot{l_1} \end{bmatrix}$
$\begin{bmatrix} r_1 \\ \dot{r_1} \\ \ddot{r_1} \end{bmatrix}$
$\begin{bmatrix} l_0 \\ \dot{l_0} \\ \ddot{l_0} \end{bmatrix}$
$\begin{bmatrix} r_0 \\ \dot{r_0} \\ \ddot{r_0} \end{bmatrix}$

Trajectory Generation (Cont.)

- For time point ***t***
  - ✓ Get the longitude and lateral value from curve function

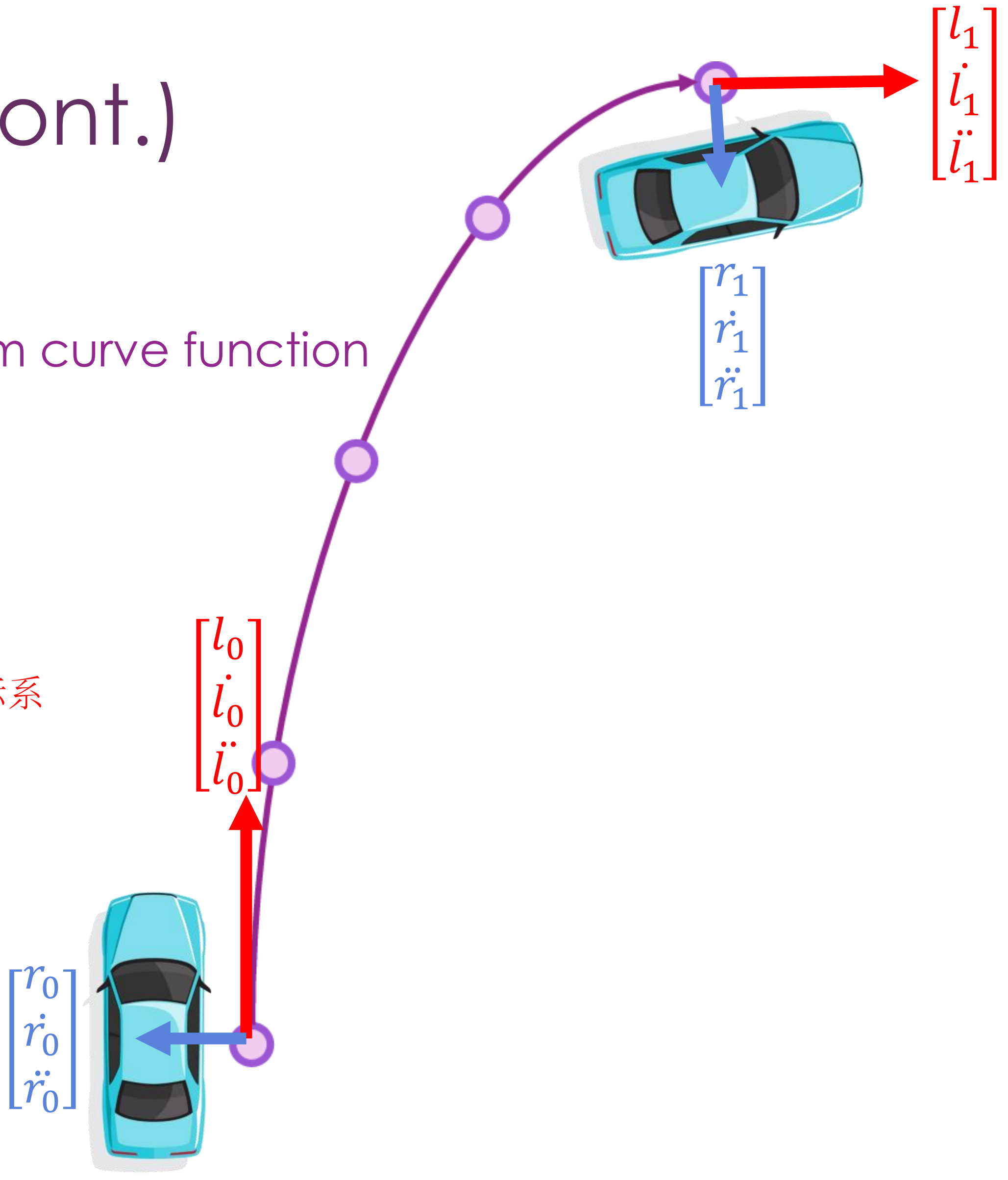
$$t \Rightarrow l = l(t), r = r(t)$$

横向与纵向位移

  - ✓ Transform to standard coordinate

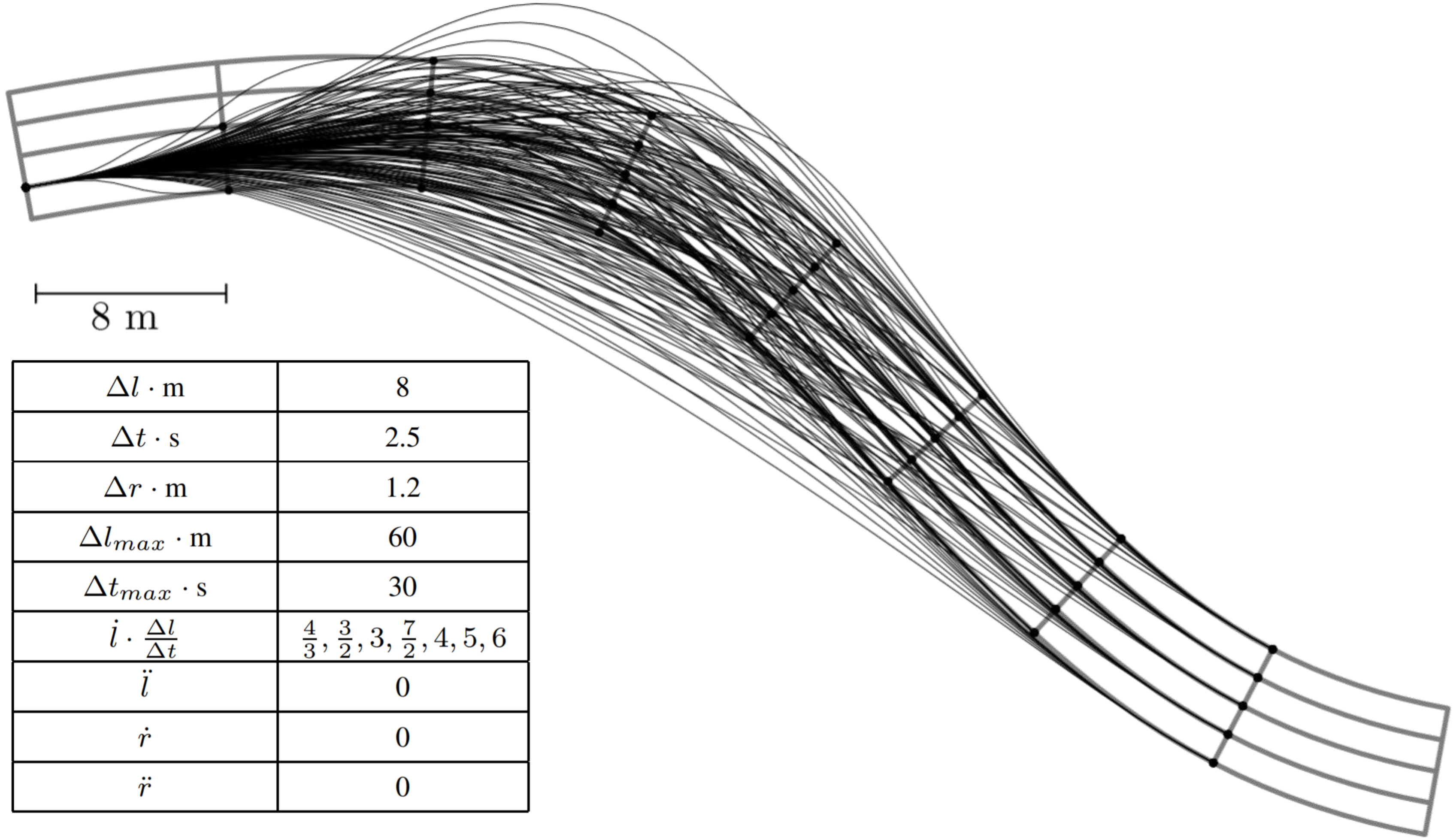
标准坐标系

$$\begin{bmatrix} l \\ r \\ t \end{bmatrix} \xrightarrow[\substack{\dot{x}(t) = \dot{l}X'(l) - \dot{r}Y(l) - r\dot{l}Y'(l) \\ \dot{y}(t) = \dot{l}Y'(l) + \dot{r}X(l) + r\dot{l}X'(l)}]{\substack{x(t) = X(l) - rY'(l) \\ y(t) = Y(l) + rX'(l)}} \begin{bmatrix} x \\ y \\ t \end{bmatrix}$$



$\begin{bmatrix} l_1 \\ \dot{l}_1 \\ \ddot{l}_1 \end{bmatrix}$
$\begin{bmatrix} r_1 \\ \dot{r}_1 \\ \ddot{r}_1 \end{bmatrix}$
$\begin{bmatrix} l_0 \\ \dot{l}_0 \\ \ddot{l}_0 \end{bmatrix}$
$\begin{bmatrix} r_0 \\ \dot{r}_0 \\ \ddot{r}_0 \end{bmatrix}$

Planning on 2D Road

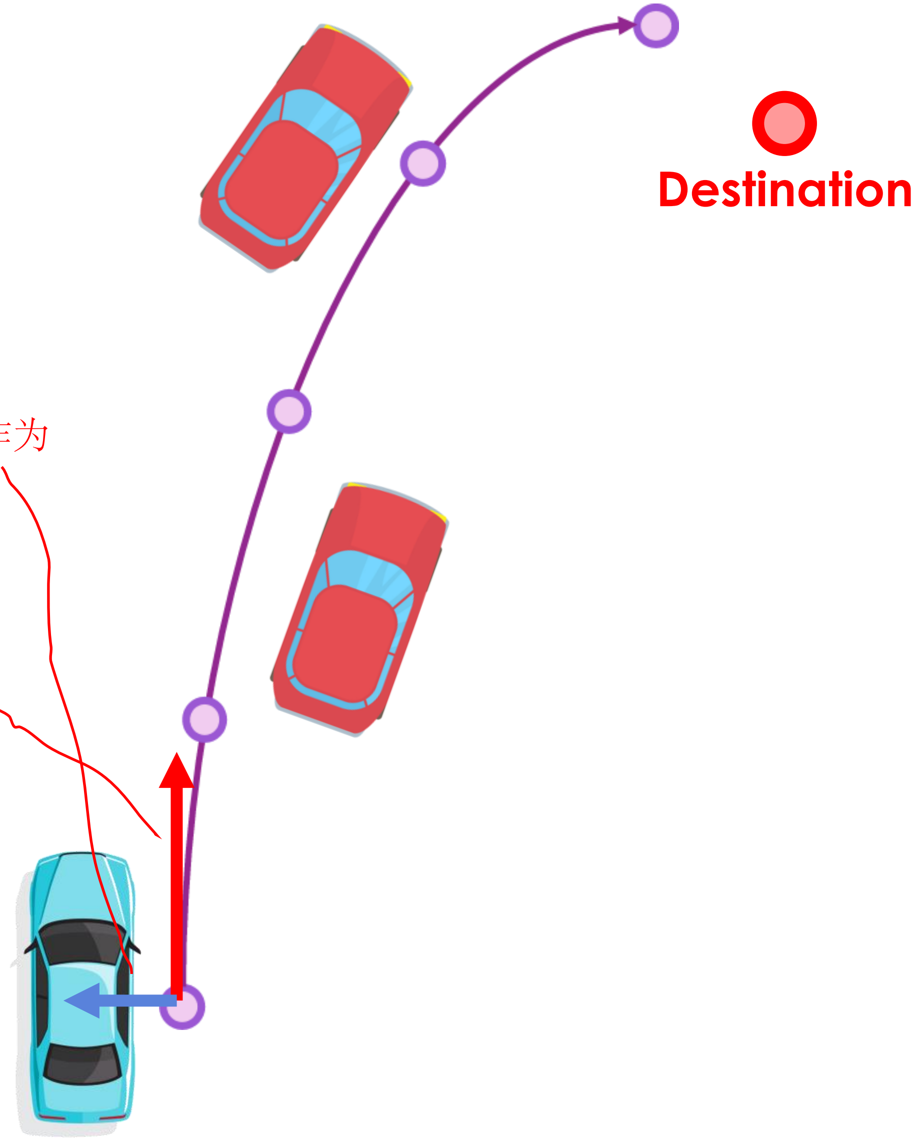


$\Delta l \cdot$ m	8
$\Delta t \cdot$ s	2.5
$\Delta r \cdot$ m	1.2
$\Delta l_{max} \cdot$ m	60
$\Delta t_{max} \cdot$ s	30
$\dot{l} \cdot \frac{\Delta l}{\Delta t}$	$\frac{4}{3}, \frac{3}{2}, 3, \frac{7}{2}, 4, 5, 6$
$\ddot{l}$	0
$\dot{r}$	0
$\ddot{r}$	0

Cost Function Design

- Achieve Destination 轨迹终点与目标点的距离
  - Objective achievement cost
- Follow the path 遵守规则
  - Lateral offset cost 跟随中心行进，使用横向位移作为
- Avoid Collision 避免碰撞
  - Collision cost 增加与障碍物的距离作为代价
- Comfortable 保证舒适性
  - Longitude jerk cost 纵向加速度微分作为代价 希望是等速或等加速度
  - Lateral acceleration cost

横向加速度尽量为0


Destination