# Path Planning Probabilistico

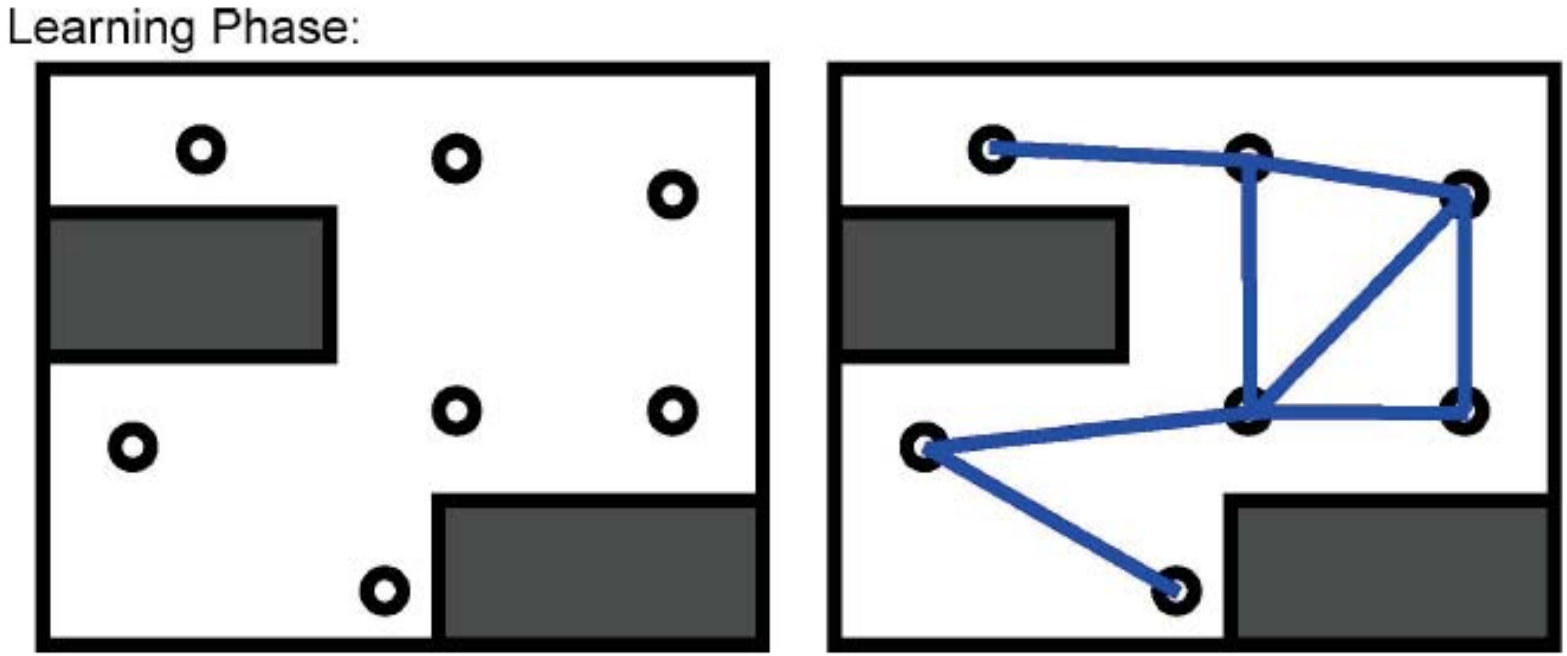
- Continuous state spaces
- Continuous actions

- PRM (Kavraki & many successors)
- RRT (Lavalle & many successors)

# Probabilistic Roadmap

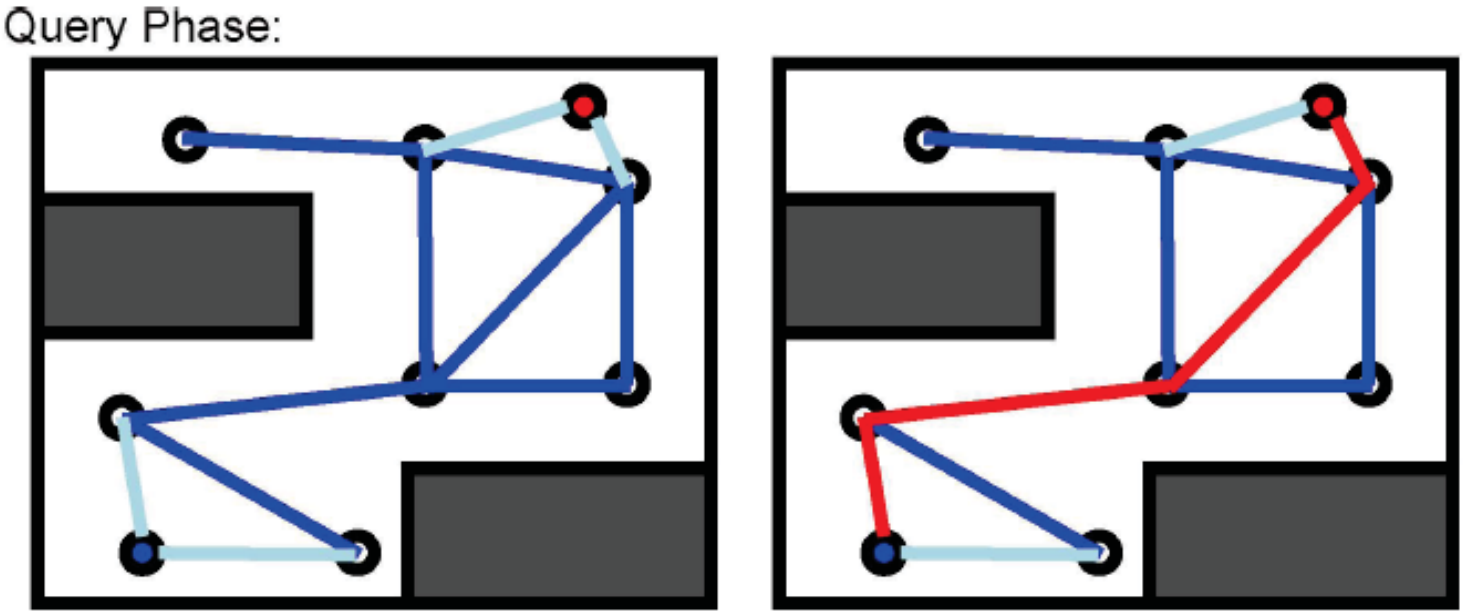
- Separate planning into two stages
  - "Learning" Phase
    - random samples of free configurations (vertices)
    - Attempt to connect pairs of nearby vertices with a local planner
    - if a valid plan is found, add an edge to the graph
  - Query Phase
    - find local connections to graph from initial and goal positions

# Fase di Learning




Configurazione random nello spazio libero, poi connessa da archi collegando tutti i k vicini a distanza minore di n

# Fase di Query




Nella fase di Query si connette goal e posizione iniziale e si cerca lo shortest path (A*, Dijkstra)

E’ probabilisticamente completo: all’aumentare dei punti, la probabilità di non trovare il percorso va a zero

# Rapidly Exploring Random Trees

- RRT
  - Explore continuous spaces efficiently
    - No need for an artificial grid
  - Form the basis for probabilistically complete planner
- Complete planners exist, but are slow
- RRT uses random search and approximation for speed

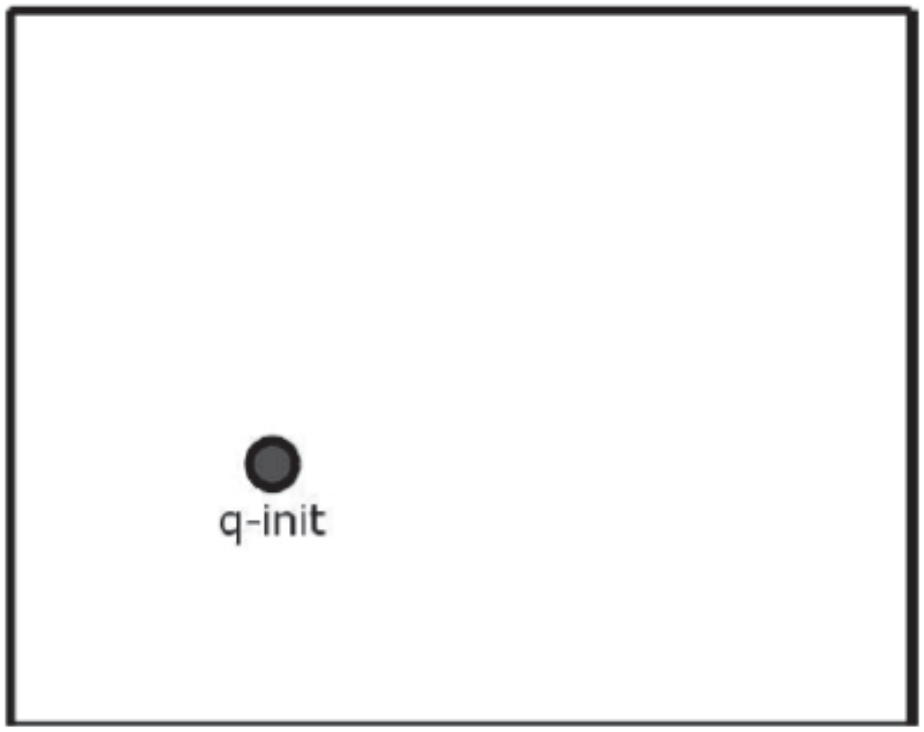
# RRT: Algoritmo

Basic RRT Algorithm:

(1) Initially, start with the initial configuration as root of tree
(2) Pick a random state in the configuration space
(3) Find the closest node in the tree
(4) Extend that node toward the state if possible
(5) Goto (2)

# RRT: Esempio (inizio)

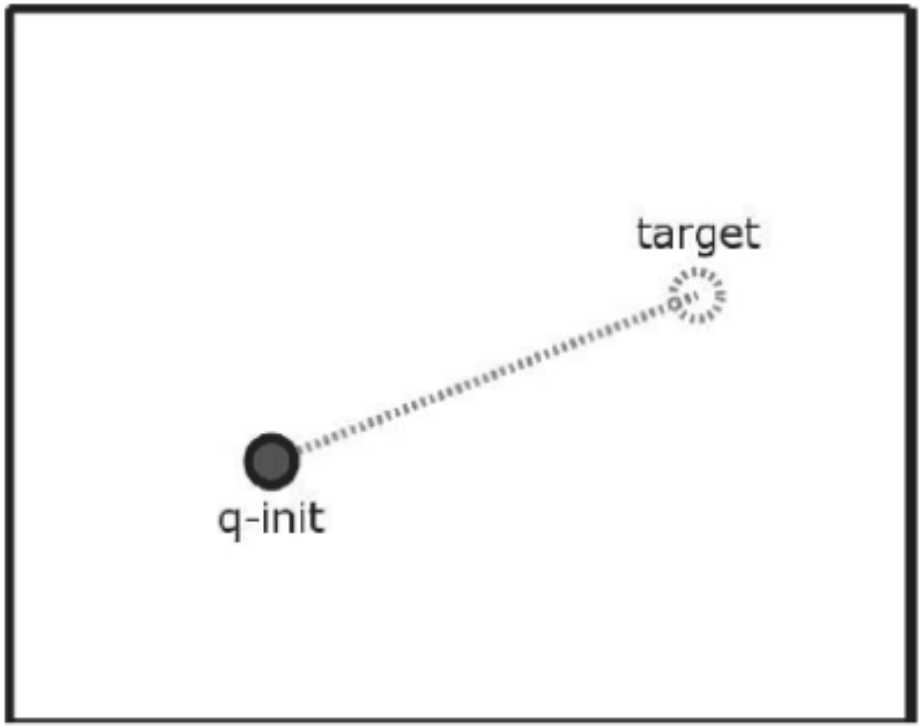
(1) Start with the initial state as the root of a tree



# RRT: Esempio (Esplorazione)

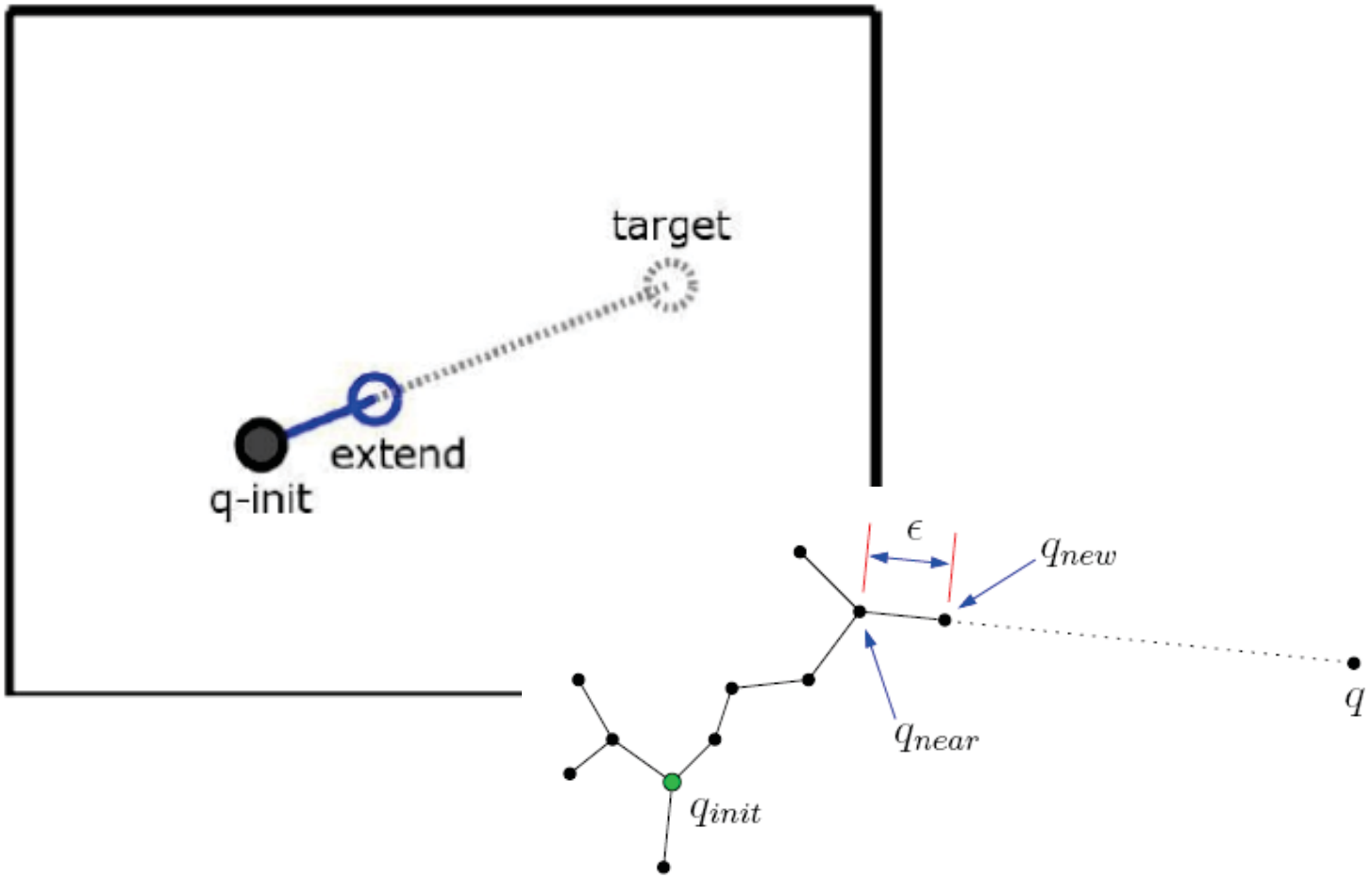
(2) Pick a random state in the environment

(3) Find the closest node in the tree

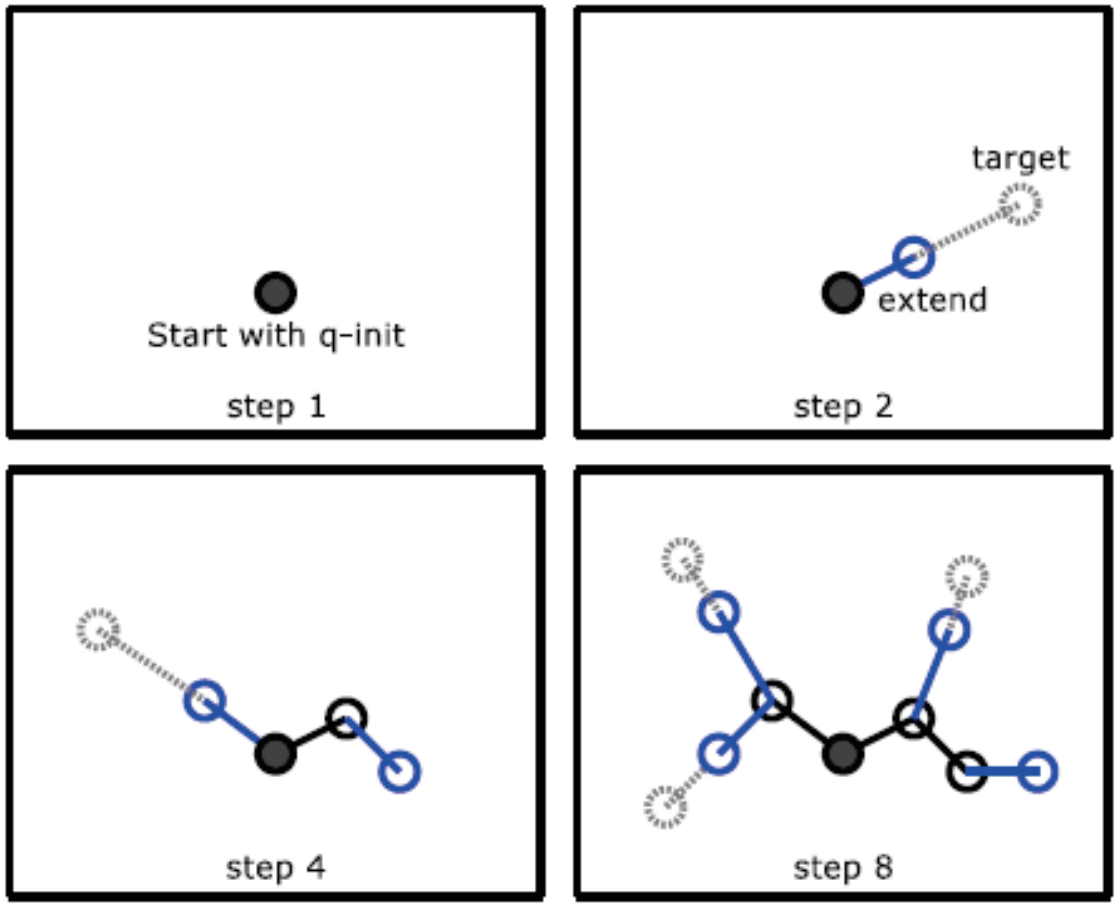


# RRT: Esempio (Ricerca)

(4) Extend that node toward the target



# RRT: Esempio (Ricerca)



# RRT: Algoritmo

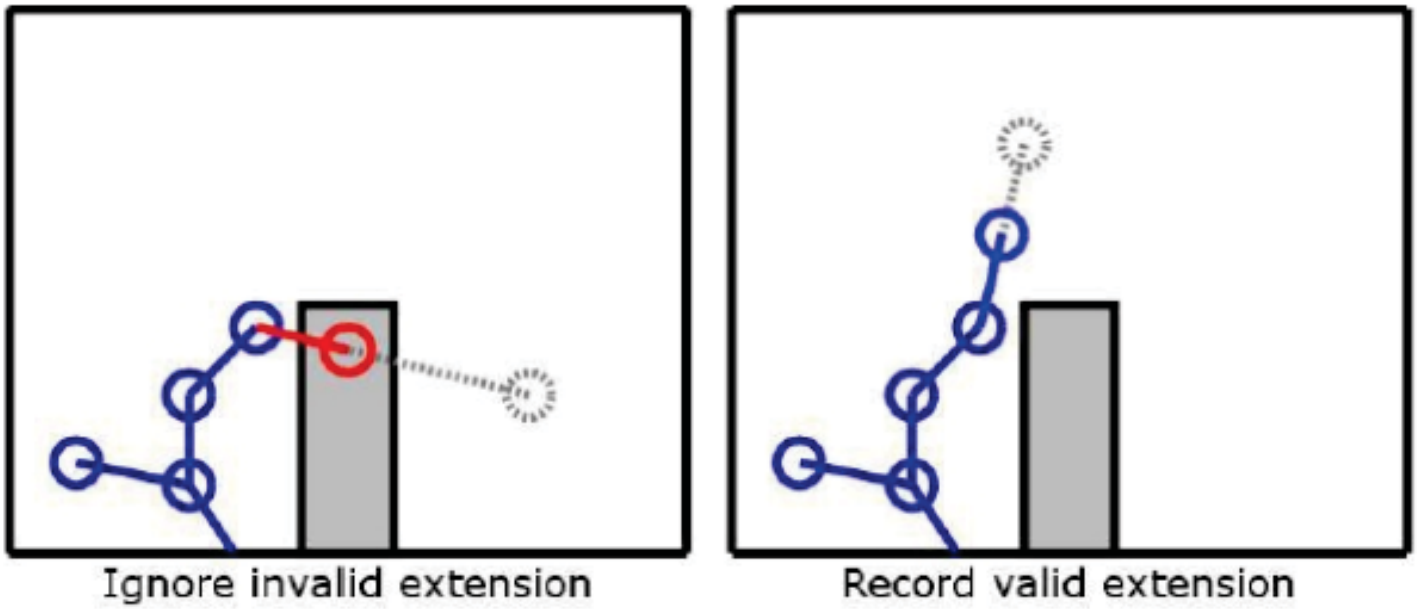
**Algorithm** BuildRRT
Input: Initial configuration $q_{init}$, number of vertices in RRT $K$, incremental distance $\Delta q$)
Output: RRT graph $G$

$G$.init($q_{init}$)

**for** $k = 1$ **to** $K$
  $q_{rand} \leftarrow$ RAND_CONF()
  $q_{near} \leftarrow$ NEAREST_VERTEX($q_{rand}$, $G$)
  $q_{new} \leftarrow$ NEW_CONF($q_{near}$, $\Delta q$)
  $G$.add_vertex($q_{new}$)
  $G$.add_edge($q_{near}$, $q_{new}$)
**return** $G$

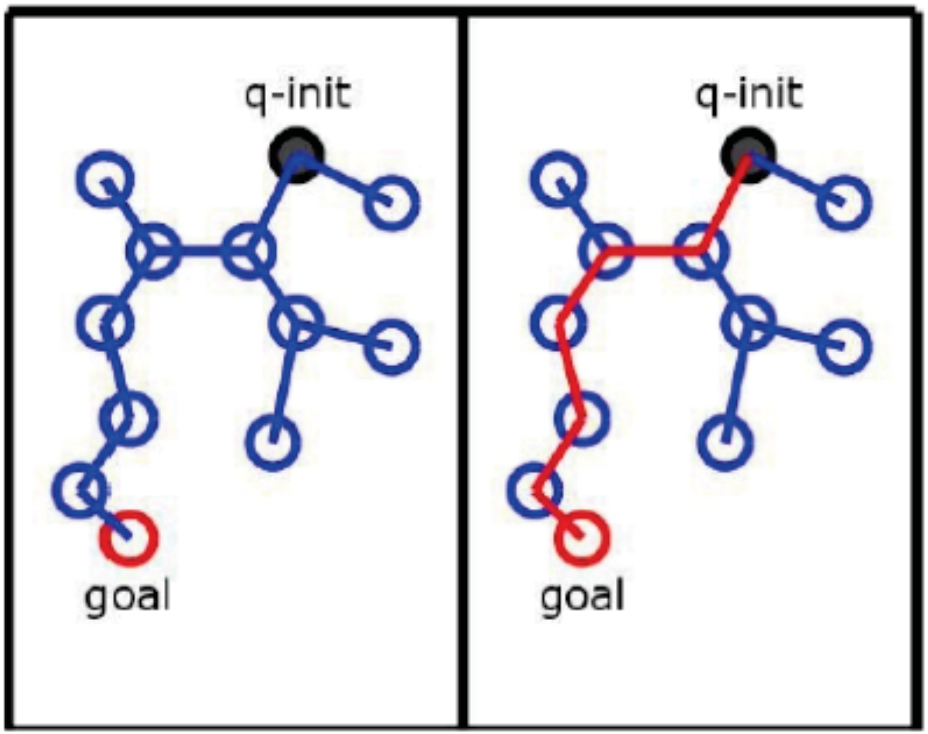
# RRT: Ostacoli

- Ignore extensions which hit obstacles
- Resulting tree contains *only* valid paths



# RRT per Planning

- Once a node of the tree is a *goal*, the plan is the path back up the tree



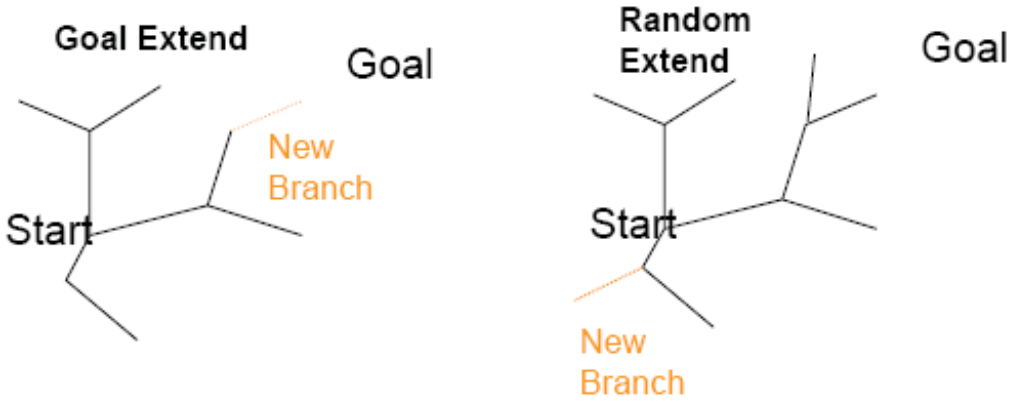
# RRT Orientato al Gol

1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - Random configuration with probability 1-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

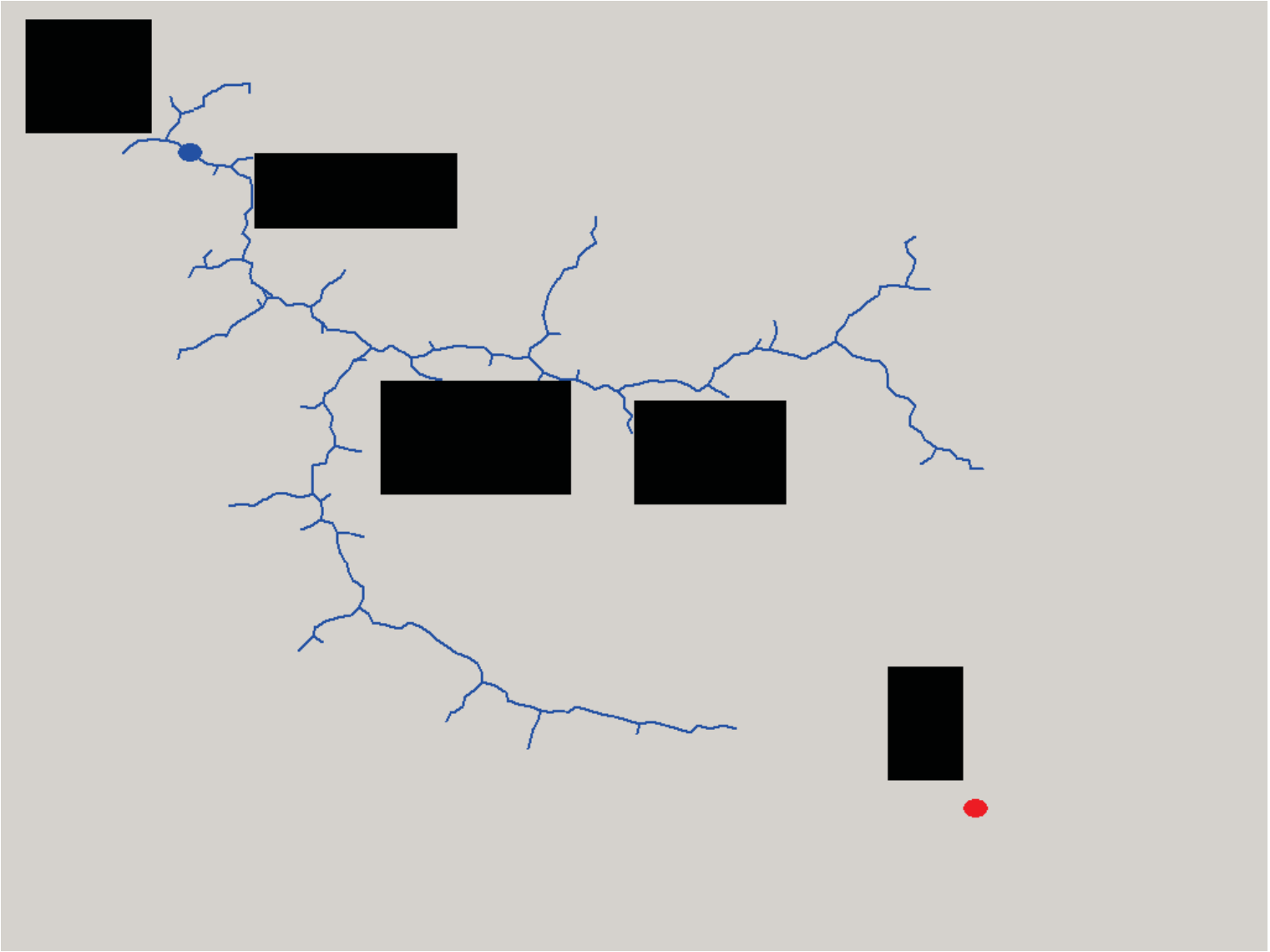
# RRT per Planning

Probability *p* : Extend *closest node* in tree towards goal

Probability *1-p* : Extend closest node towards a random point



# RRT



# RRT: Algoritmo

```
function RRTPlan (env:environment,initial:state,
                  goal:state):rrt-tree
    var nearest,extended,target:state;
    var tree:rrt-tree;
    nearest := initial;
    rrt-tree := initial;
    while(Distance (nearest,goal) < threshold)
        target = ChooseTarget (goal);
        nearest = Nearest (tree,target);
        extended = Extend (env,nearest,target);
        if extended ≠ EmptyState then
            AddNode (tree,extended);
    return tree;

function ChooseTarget (goal:state):state
    var p:real;
    p = UniformRandom in [0.0 .. 1.0];
    if 0 < p < GoalProb then
        return goal;
    else if GoalProb < p < 1 then
        return RandomState();

function Nearest (tree:rrt-tree,target:state):state
    var nearest:state;
    nearest := EmptyState;
    foreach state s in current-tree
        if Distance (s,target) <
        Distance (nearest,target) then
            nearest := s;
    return nearest;
```

# RRT: Planning and Replanning

- Environments and planning
  - Value of p?
- Dynamic environments

- When failure, what to do?

# RRT: planning and replanning

Plain RRT planner has little bias toward plan quality, and replanning is naive (all previous information is thrown out before replanning).

Waypoints:

- Previously successful plans can guide new search
- Biases can be encoded by modifying the target point distribution

Waypoint Cache:

- When a plan is found, store nodes into a bin with random replacement
- During target point selection, choose a random item from waypoint cache with probability $q$

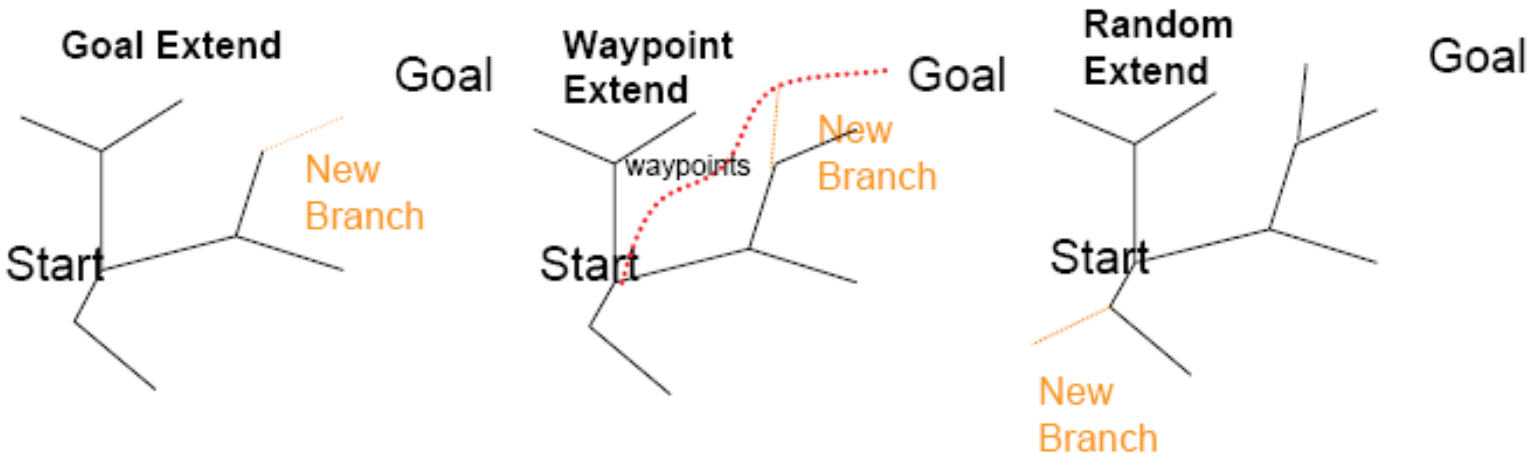
# ERRT: RRT esteso

*Save past path as waypoints*

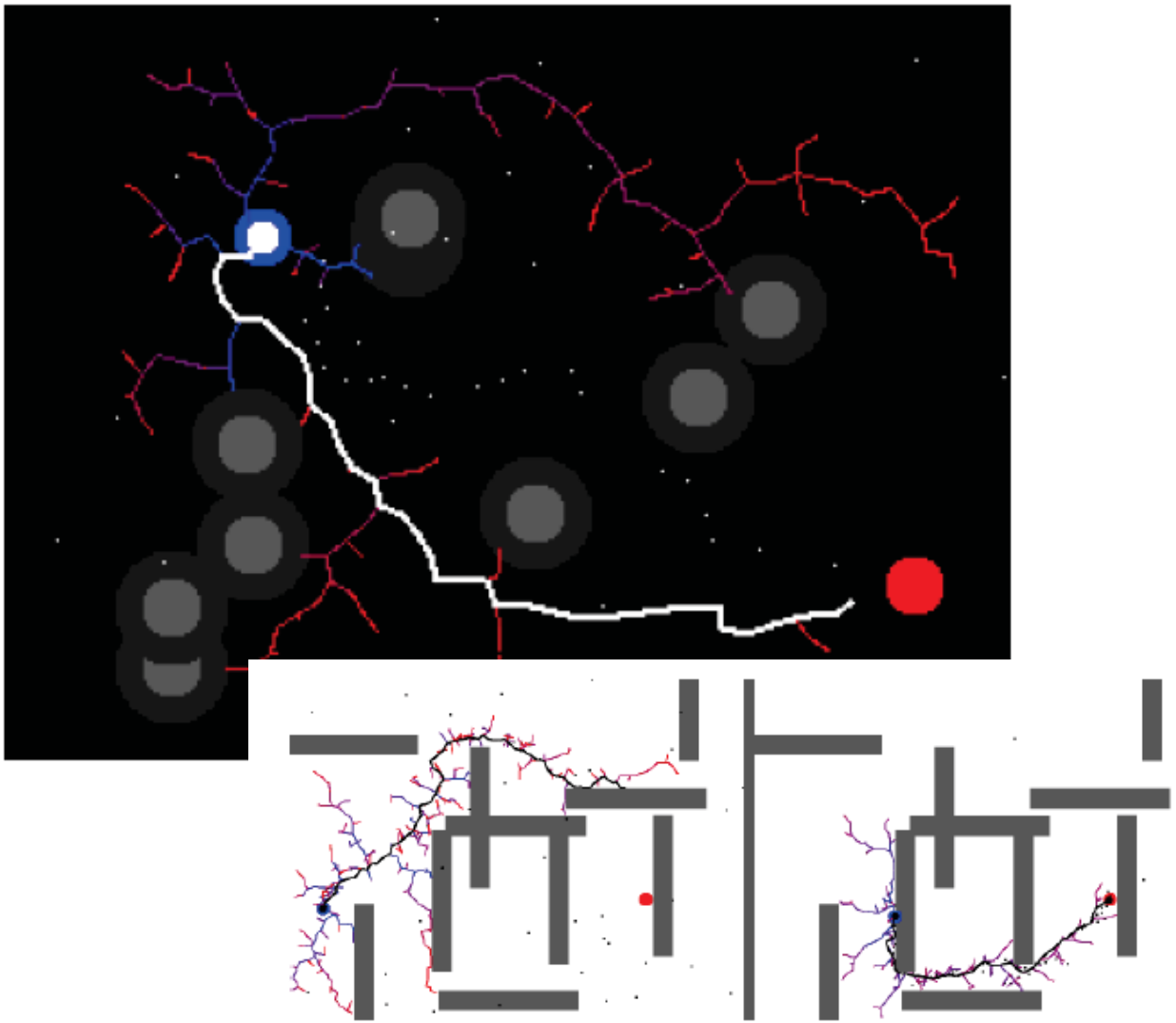
1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - **Random item from waypoint cache with probability q**
   - Random configuration with probability 1-q-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

# ERRT: replanning

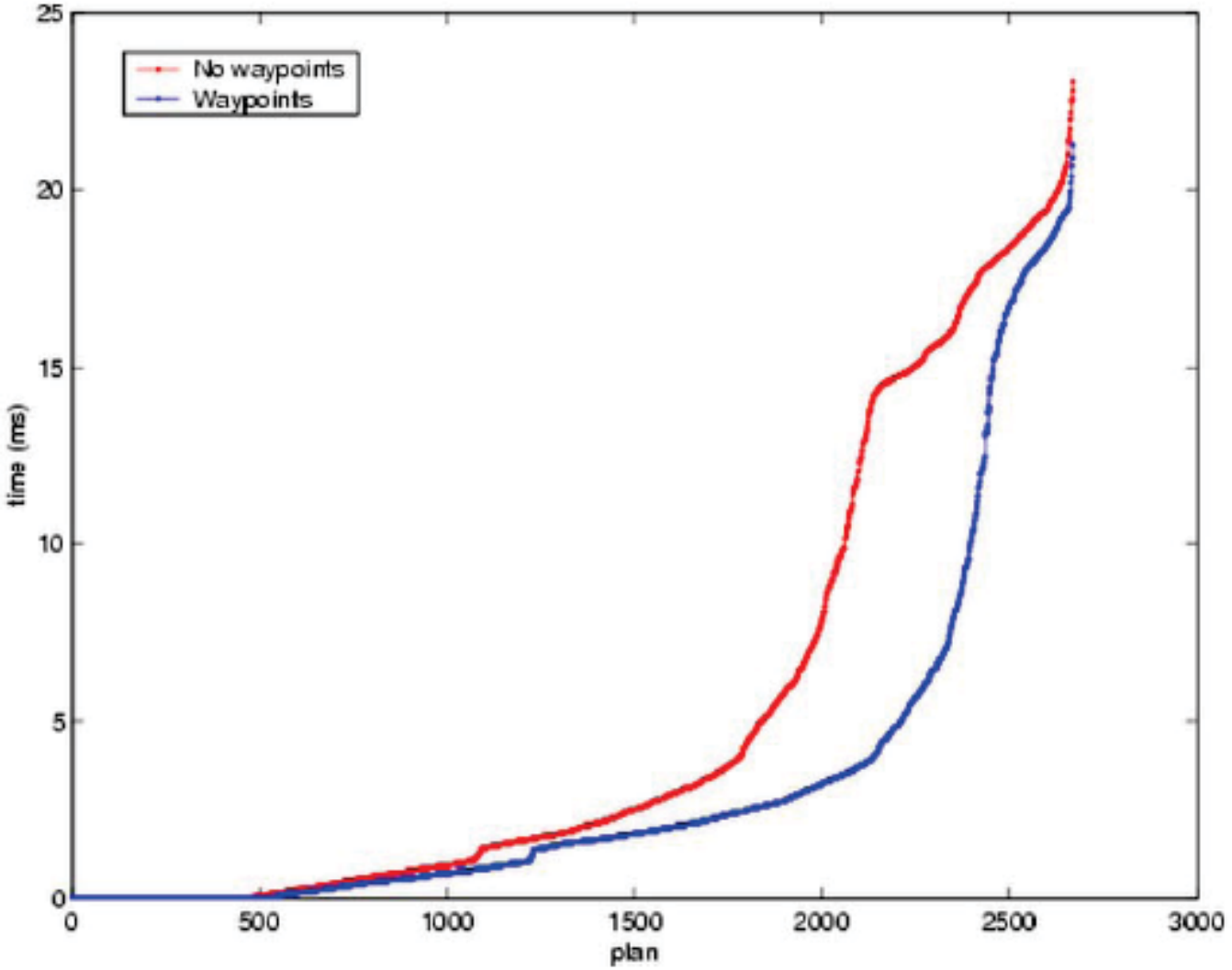
Probability *p* : Extend closest node in tree towards goal
Probability *r* : Extend closest node in tree towards random cache point
Probability *1-p-r* : Extend closest node towards a random point



# ERRT: replanning con waypoint



# Prestazioni



# Discussione

- Planning with RRT
  - High p – few known obstacles
  - Low p – many known obstacles
- Replanning with ERRT
  - High q – small dynamics (no state change)
  - Low q – high dynamics (lots of state change)
  - ERRT – bias to use previous plan; but could be any other bias
- RRT and ERRT – probabilistic convergence

# RRT: Applicazioni