# Path Planning Probabilistico

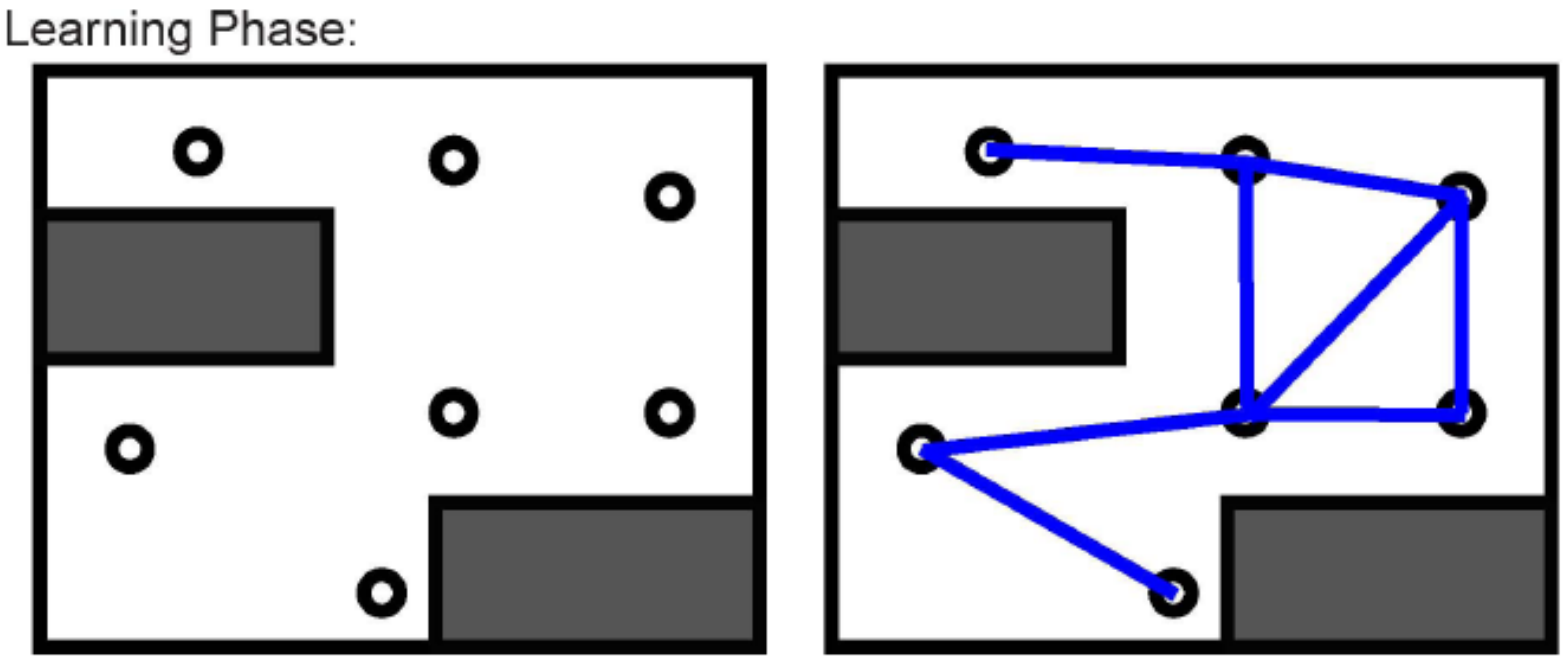
- Continuous state spaces
- Continuous actions

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

# Probabilistic Roadmap

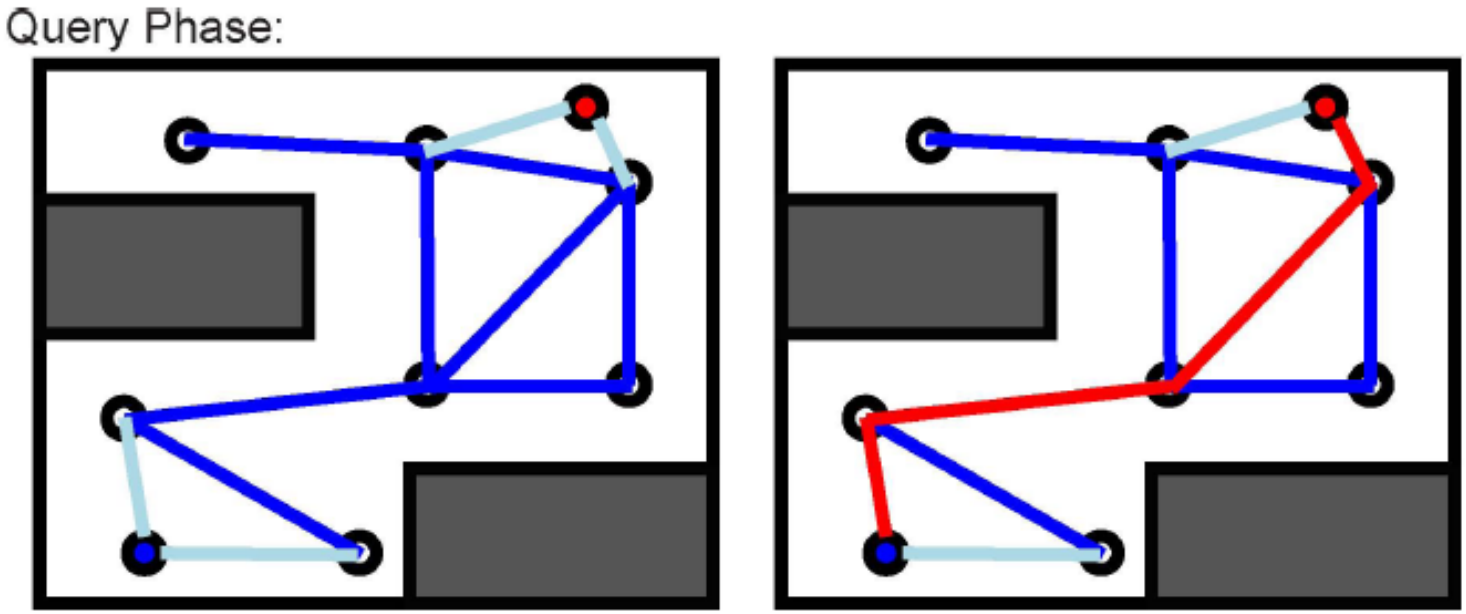
- Separate planning into two stages
  - "Learning" Phase
    - random samples of free configurations (vertices)
    - Attempt to connect pairs of nearby vertices with a local planner
    - if a valid plan is found, add an edge to the graph
  - Query Phase
    - find local connections to graph from initial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

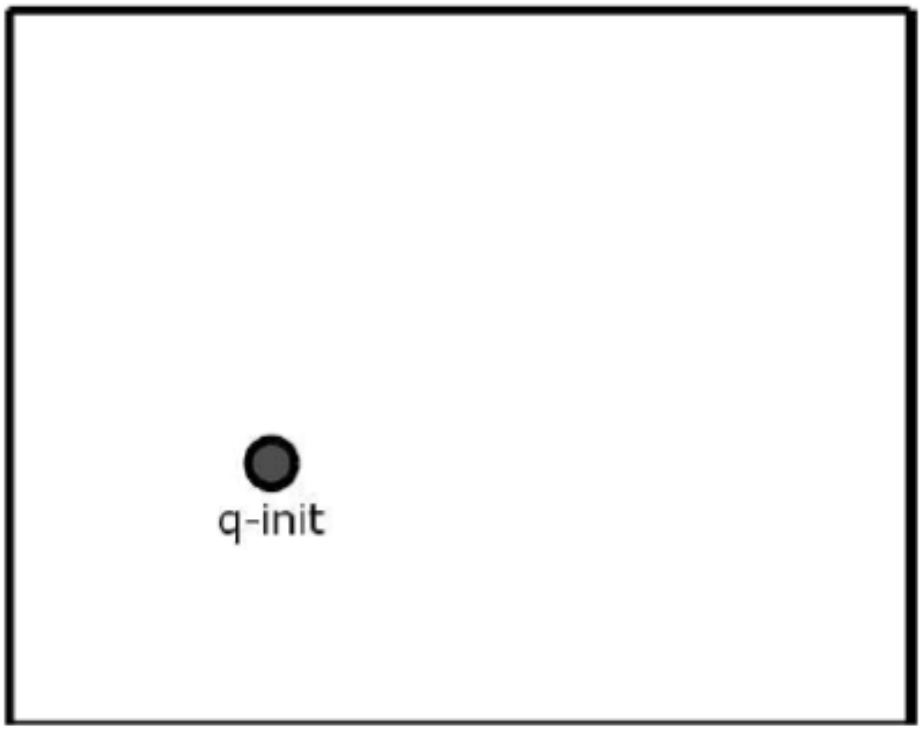
# RRT: Algoritmo

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)

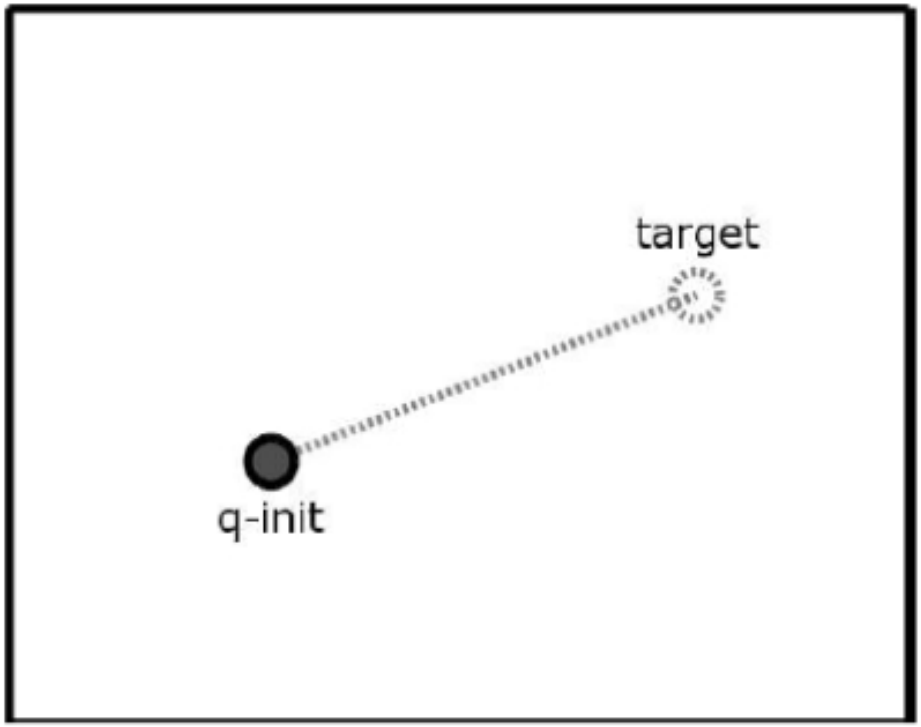
(1) Start with the initial state as the root of a tree



# RRT: Esempio (Esplorazione)

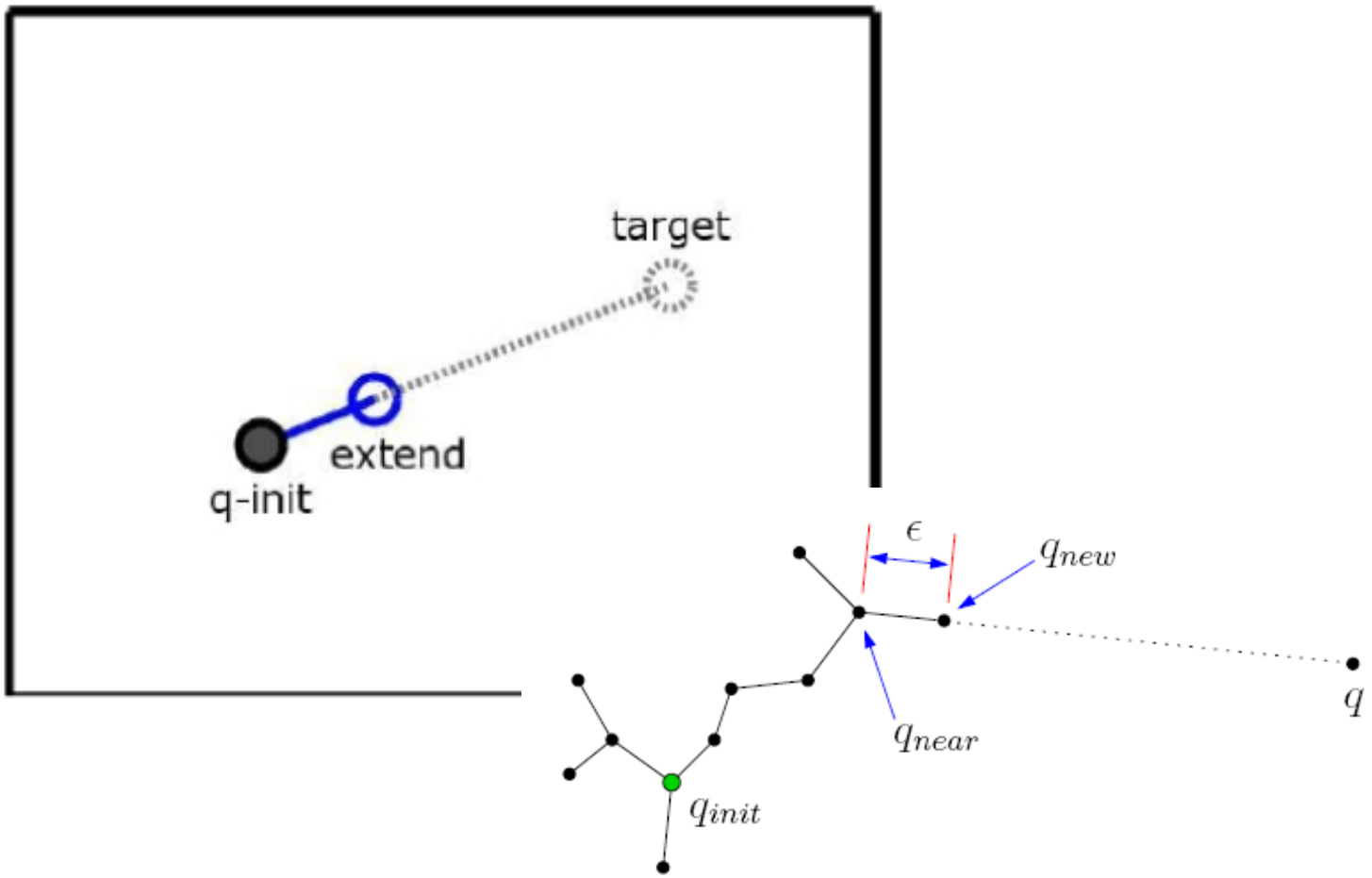
(2) Pick a random state in the environment

(3) Find the closest node in the tree

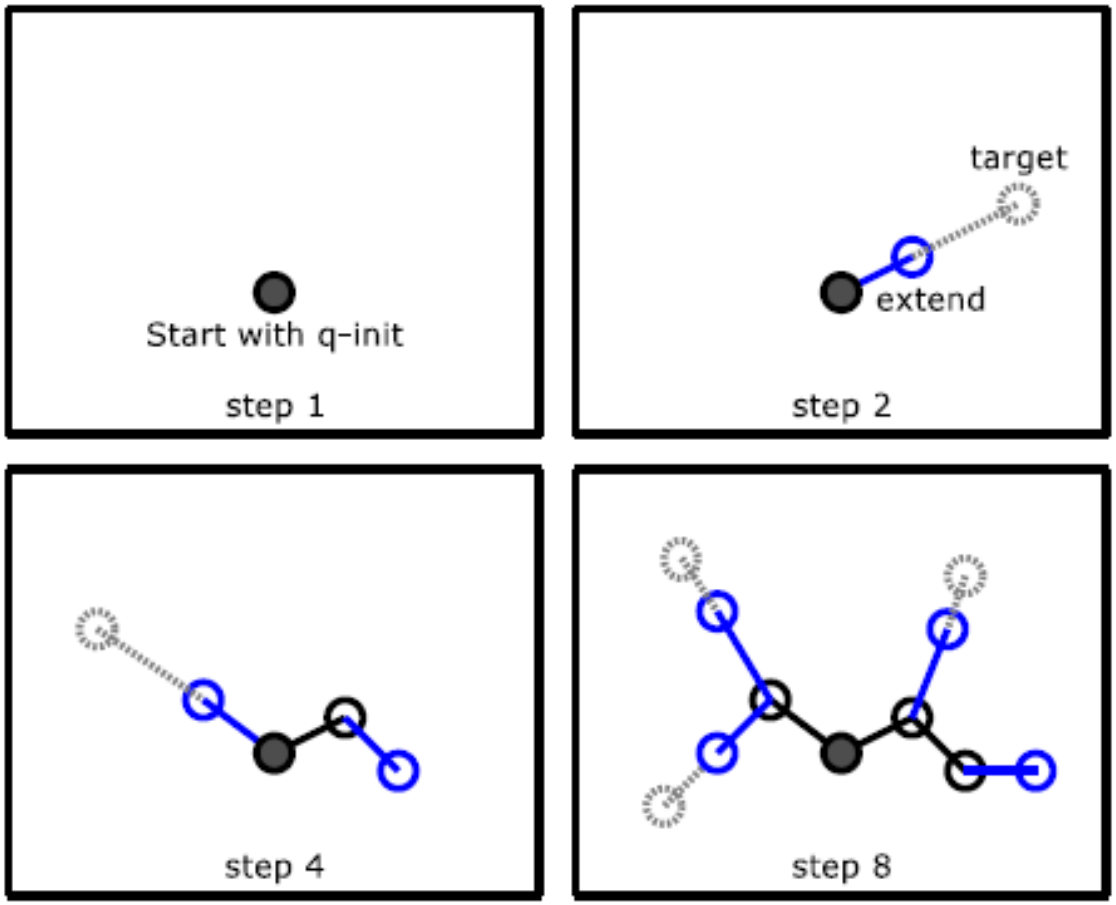


# RRT: Esempio (Ricerca)

(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($q_{init}$)

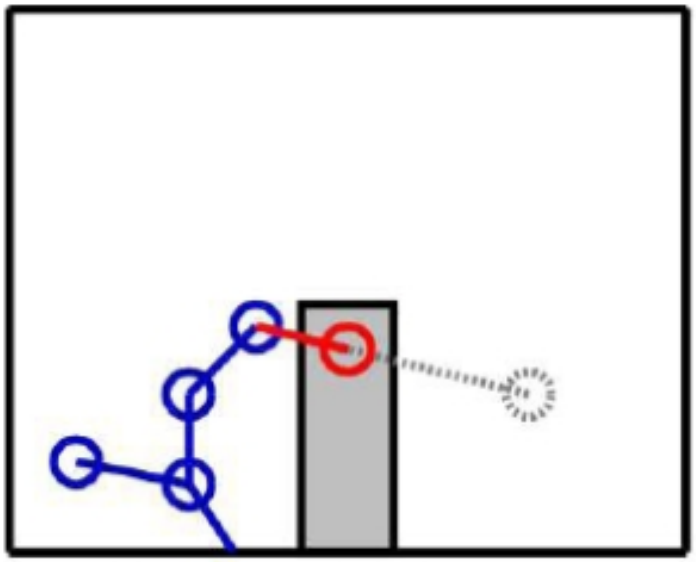
**for** $k$ = 1 **to** $K$
    $q_{rand} \leftarrow$ RAND_CONF()
    $q_{near} \leftarrow$ NEAREST_VERTEX($q_{rand}$, $G$)
    $q_{new} \leftarrow$ NEW_CONF($q_{near}$, $\Delta q$)
    $G$.add_vertex($q_{new}$)
    $G$.add_edge($q_{near}$, $q_{new}$)
**return** $G$

**Algorithm 3**: RRT.

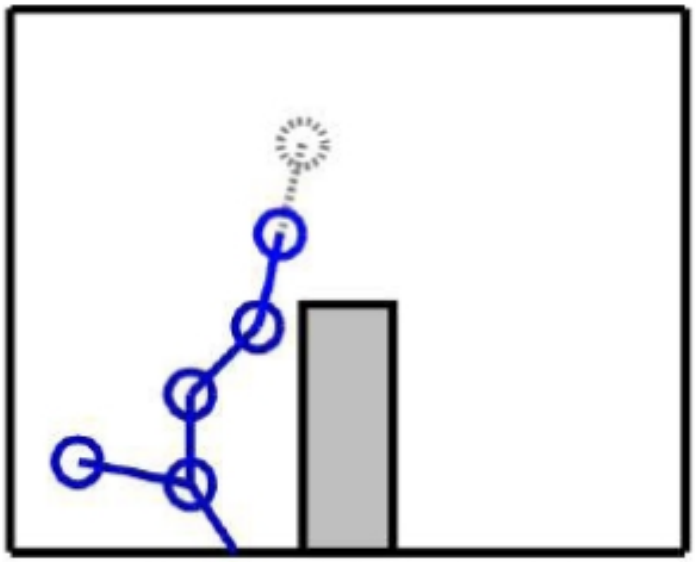
1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$ ;
6  **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7  $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$ ;
8 **return** $G = (V, E)$;

# RRT: Ostacoli

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



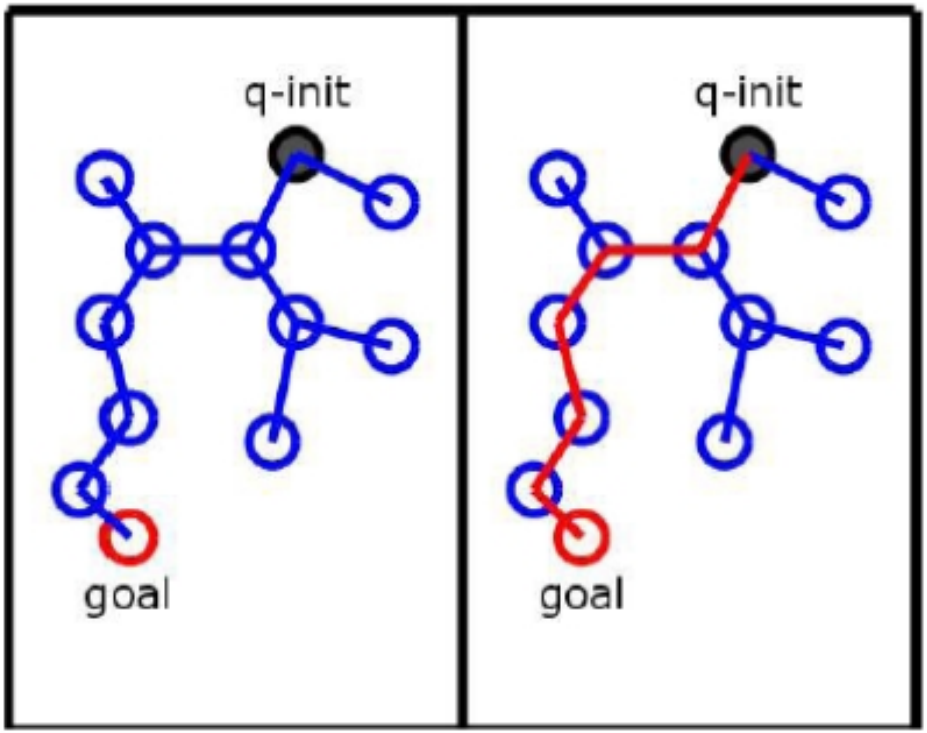
Ignore invalid extension



Record valid extension

# 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

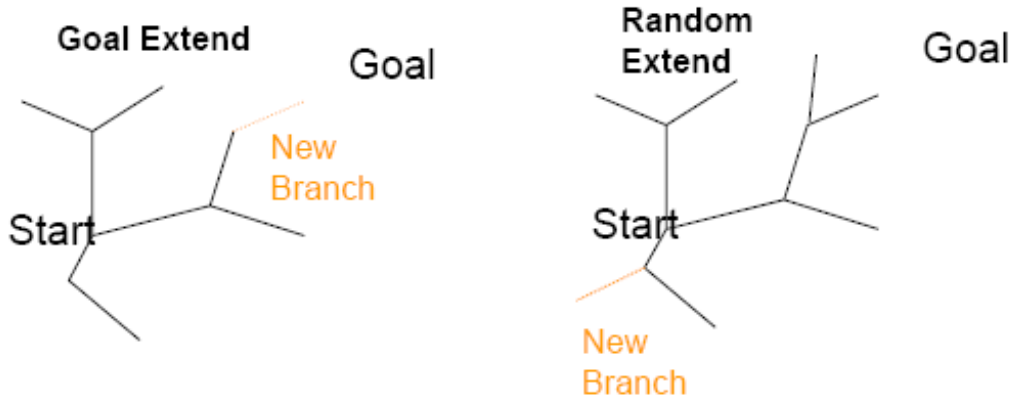
# RRT Orientato al Gol

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

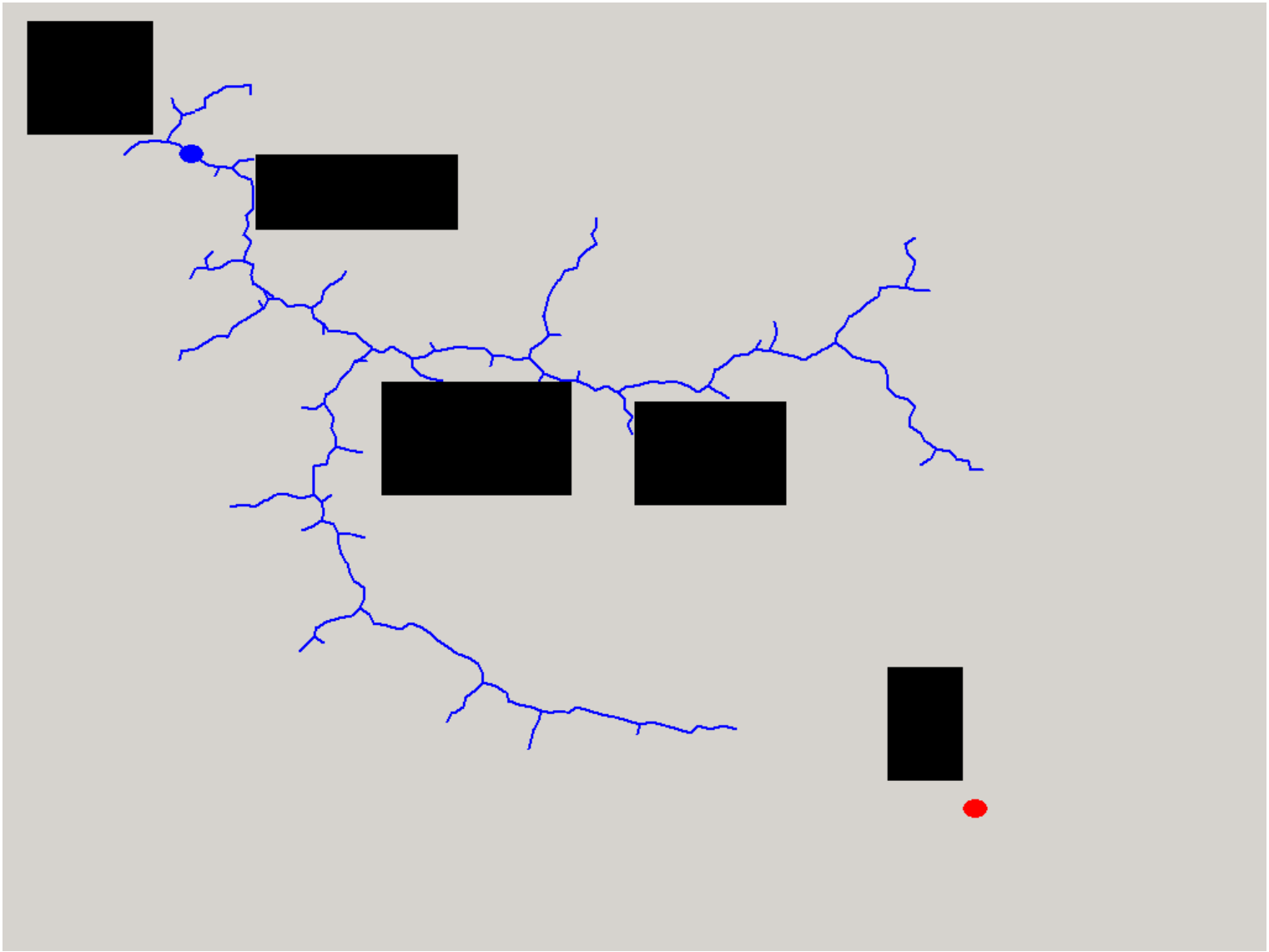
# RRT per Planning

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 / Replanning

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

- When failure, what to do?

# RRT: Planning / 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: Execution extended RRT

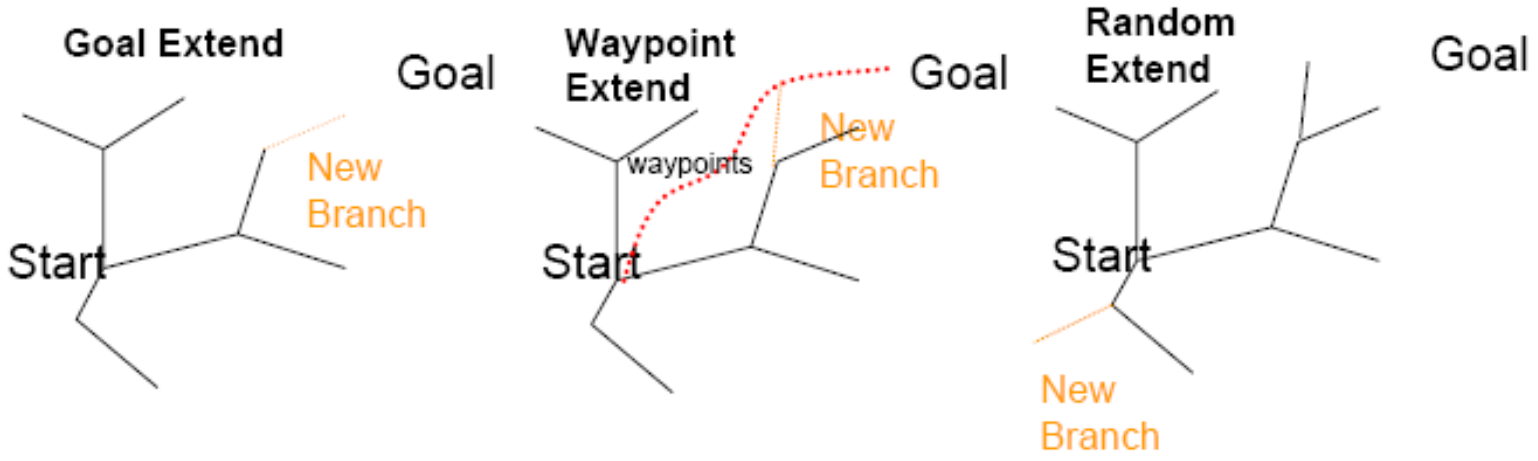
*Save past path as waypoints*

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

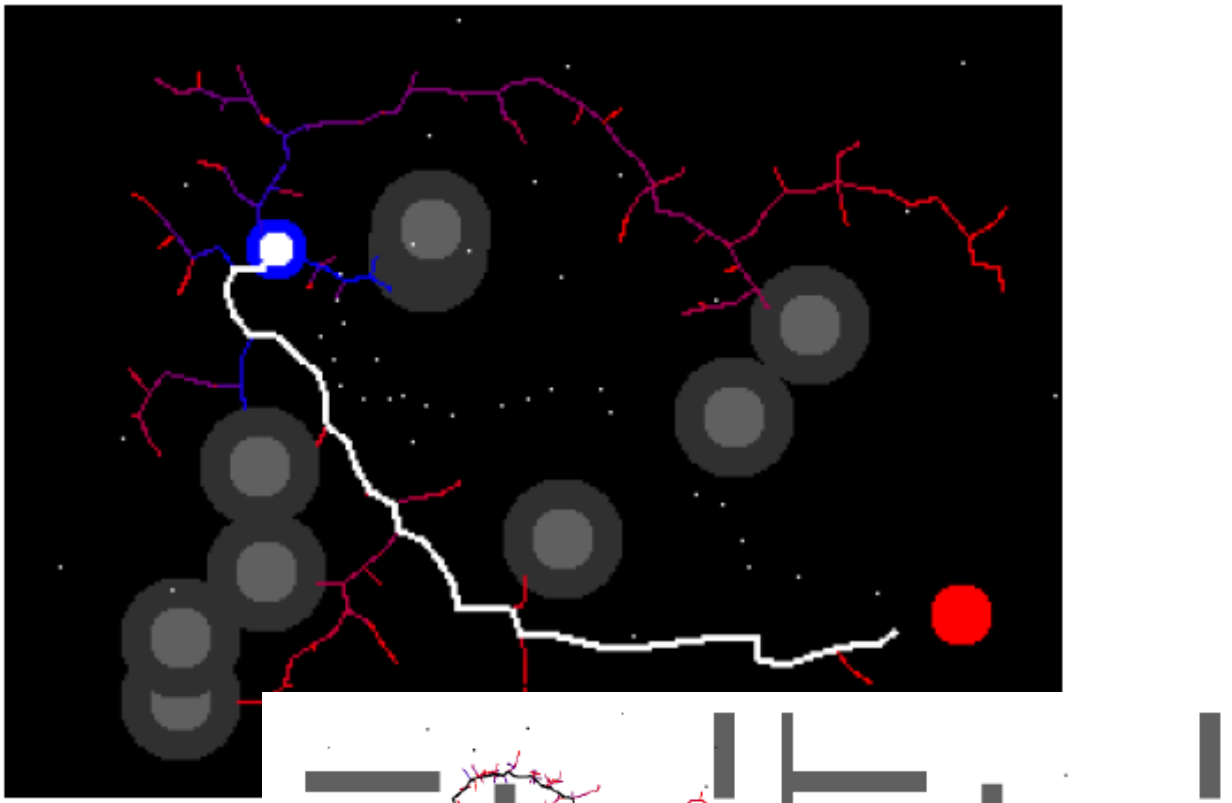
Bruce & Veloso, Real-time randomized path planning for robot navigation, 2003

# ERRT: replanning

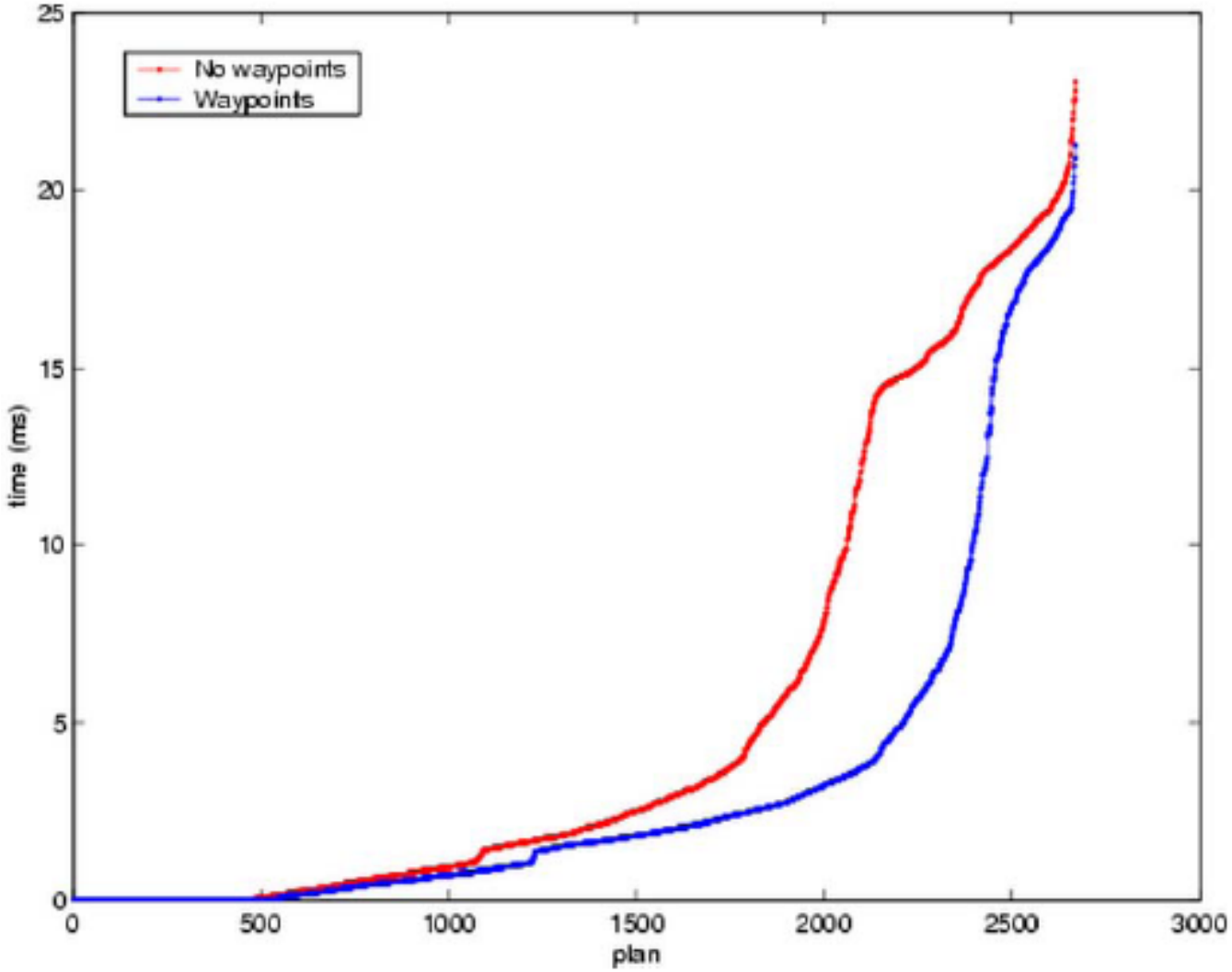
Probability *p* : Extend closest node in tree towards goal
Probability *r* : Extend closest node in tree towards random cache point
Probability *1-p-r* : Extend closest node towards a random point



# 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

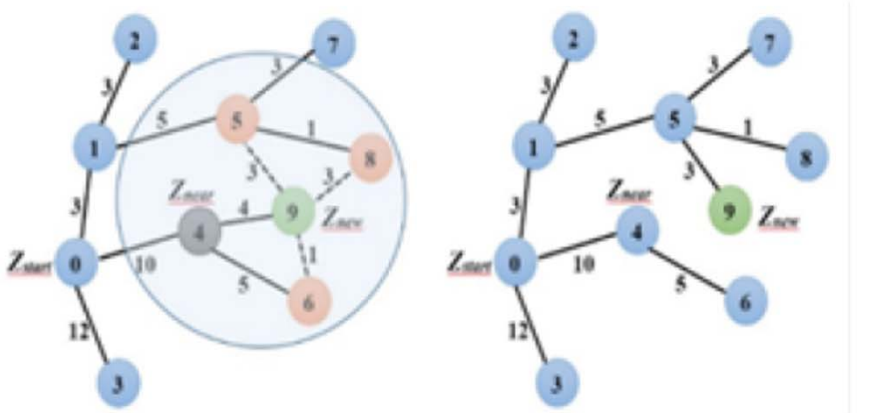
# RRT*

- Ricerca dei vicini e riscrittura dell'albero
  - Ricerca:
    - miglior nodo parent per il nuovo nodo prima della sua inserzione nell'albero (nell'area di una sfera di raggio r).
  - Riscrittura:
    - rigenera l'albero nel raggio di area k per maintenere l'albero con costo minimo

S. Karaman and E. Frazzoli, "Sampling-Based Algorithms for Optimal Motion Planning," The International Journal of Robotics Research, 30(7), 2011 pp. 846–894.

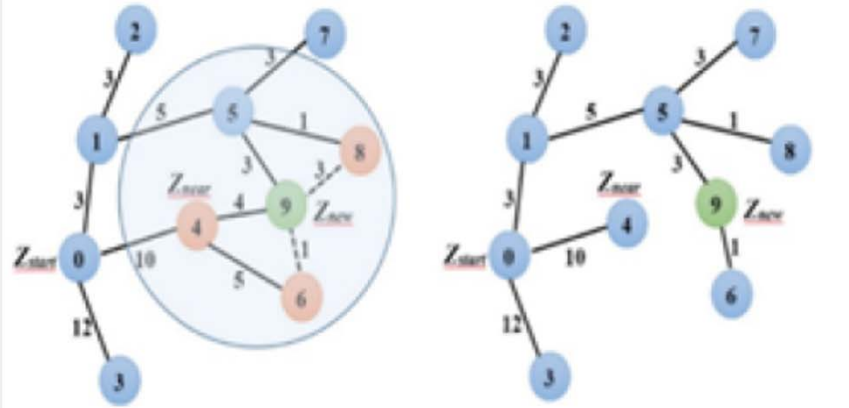
# RRT*

- Ricerca dei vicini e riscrittura dell'albero



(a) Find near

(b) Select best parent.



(c) Check cost for rewiring.

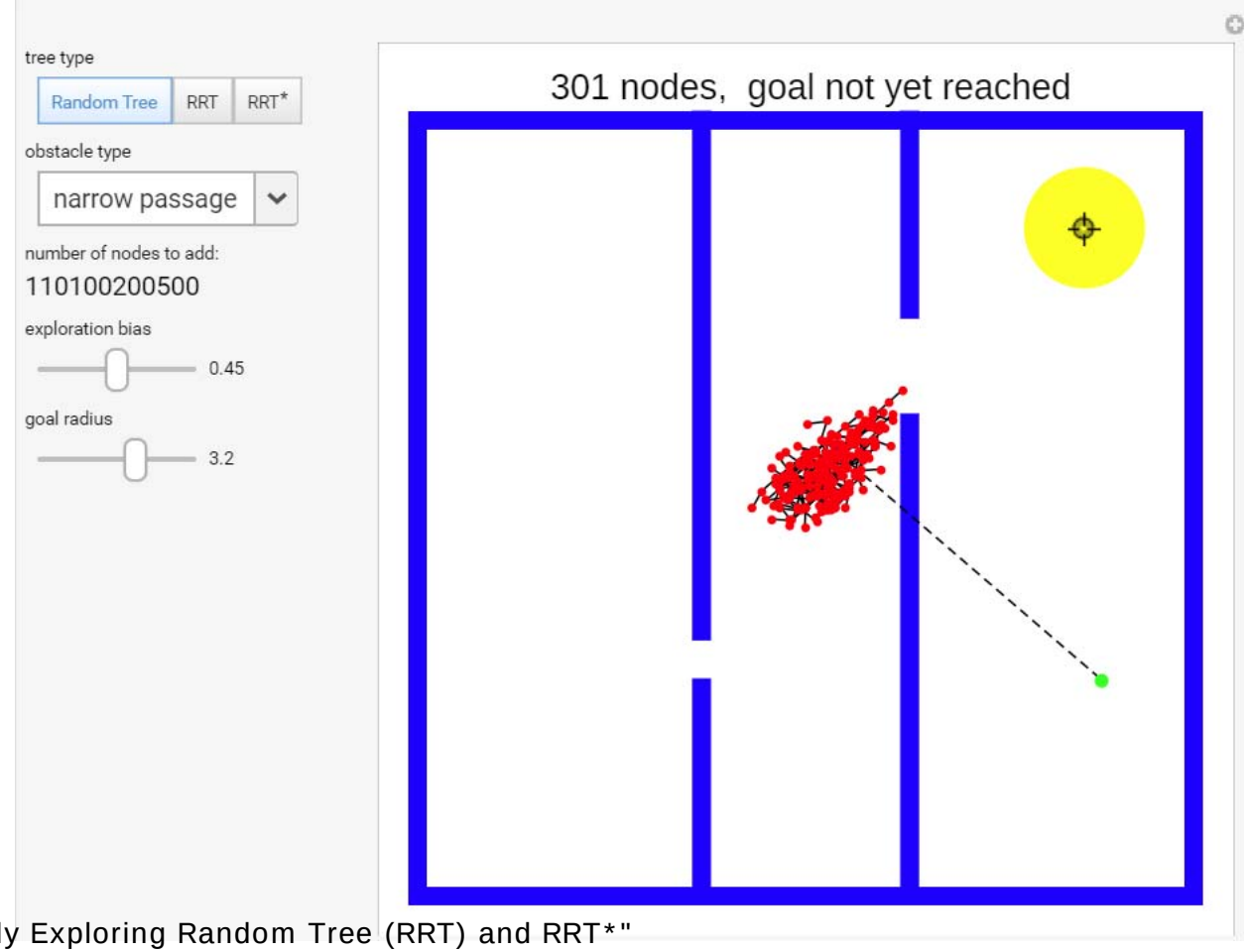
(d) Rewired tree.

**Algorithm 6**: RRT*.

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 $\quad$ **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $\quad\quad X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V)) / \text{card}(V))^{1/d}, \eta\})$;
8 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$;
9 $\quad\quad x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
10 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
11 $\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
12 $\quad\quad\quad\quad x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
13 $\quad\quad E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
14 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
15 $\quad\quad\quad$ **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** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
16 $\quad\quad\quad E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
17 **return** $G = (V, E)$;

# RT, RRT, RRT*

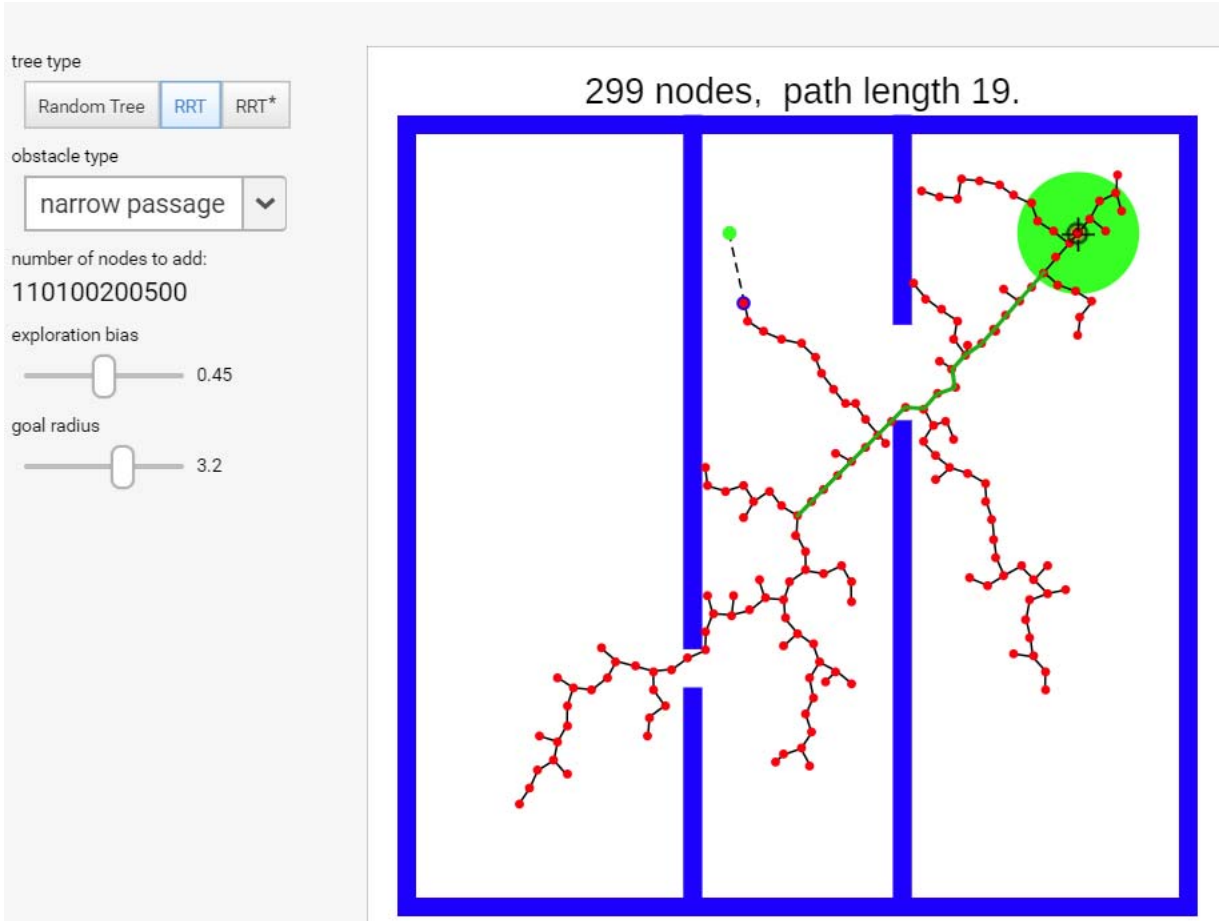
- Simulatore: Ramdom Tree




Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

# RT, RRT, RRT*

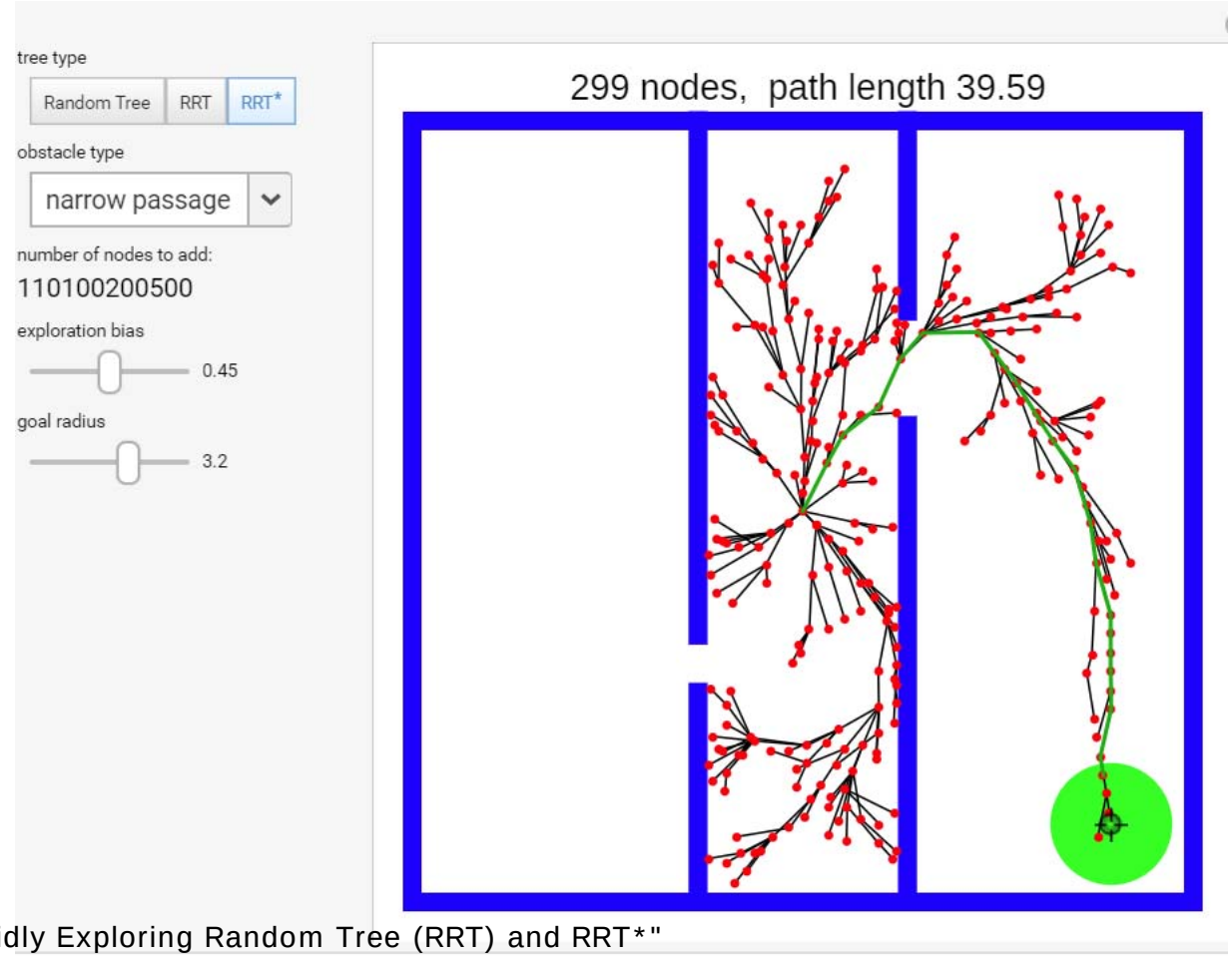
- Simulatore: RRT




Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

# RT, RRT, RRT*

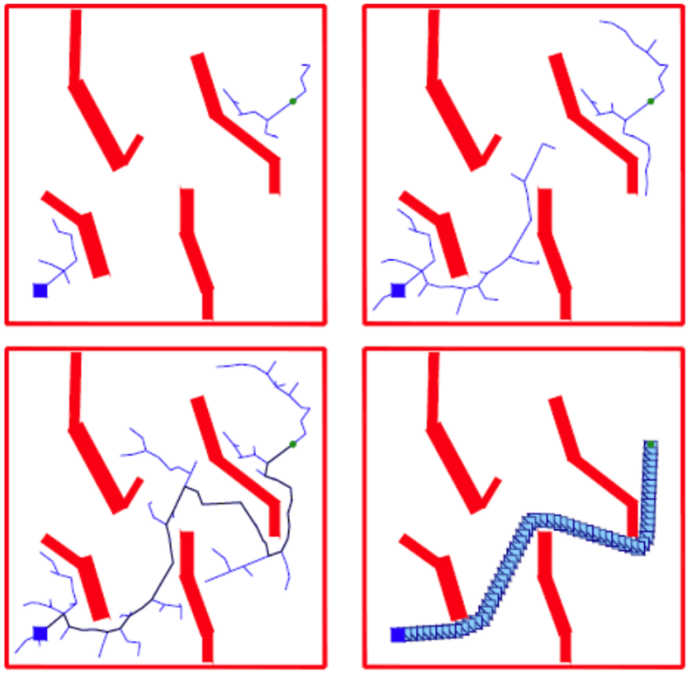
- Simulatore: RRT*




Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

# RRT-Connect

- Ricerca dal init e goal
- Extend e Connect

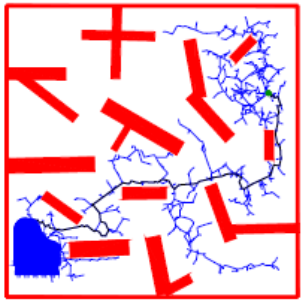


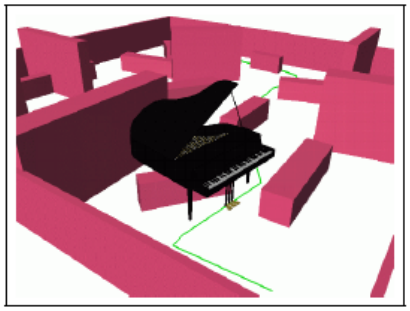
CONNECT($\mathcal{T}, q$)
1 **repeat**
2 $S \leftarrow$ EXTEND($\mathcal{T}, q$);
3 **until not** ($S =$ *Advanced*)
4 Return $S$;

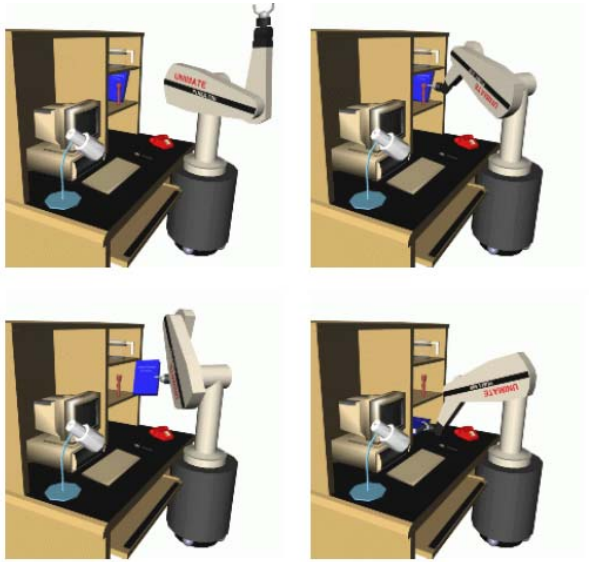
RRT_CONNECT_PLANNER($q_{init}, q_{goal}$)
1 $\mathcal{T}_a$.init($q_{init}$); $\mathcal{T}_b$.init($q_{goal}$);
2 **for** $k = 1$ **to** $K$ **do**
3 $q_{rand} \leftarrow$ RANDOM_CONFIG();
4 **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
5 **if** (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
6 Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
7 SWAP($\mathcal{T}_a, \mathcal{T}_b$);
8 Return *Failure*

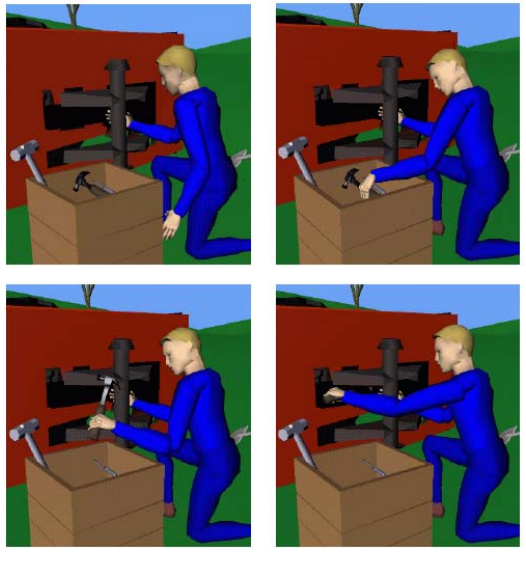
RRT-Connect: An Efficient Approach to Single-Query Path Planning. Kuffner and LaValle 2000.

# RRT-Connect: Applicazioni









RRT-Connect: An Efficient Approach to Single-Query Path Planning. Kuffner and LaValle 2000.