

1 Journey to Success(or-State Axioms)

1. First, let's review some definitions. What are successor-state axioms?

Successor-state axioms are axioms outlining what preconditions must be true in order to ensure that the state at the next time step will be specified. By definition, it is an axiom that sets the truth value of F^{t+1} (where F is some fluent, or changeable variable in an environment) in one of two ways:

- The action at time t causes F to be true at $t + 1$ (which refers to $ActionCausesF^t$)
- F was already true at time t and the action at time t does not cause it to be false.

It has the following schema: $F^{t+1} \iff ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$.

We use successor-state axioms to ensure that each state we compute is the result of legal action.

2. Consider the following Mini Pacman grid. In this simplified world, the only available actions are *Left*, *Right*, and *Stay*. The only possible states are $Pacman_{(1,1)}$ and $Pacman_{(2,1)}$. If Pacman tries to move into a wall, he will stay in the same state.

Notice that Pacman's state and actions are both fluent, so we can set up successor-state axioms to define how Pacman moves in this world. Write the successor-state axiom corresponding to Figure 3.

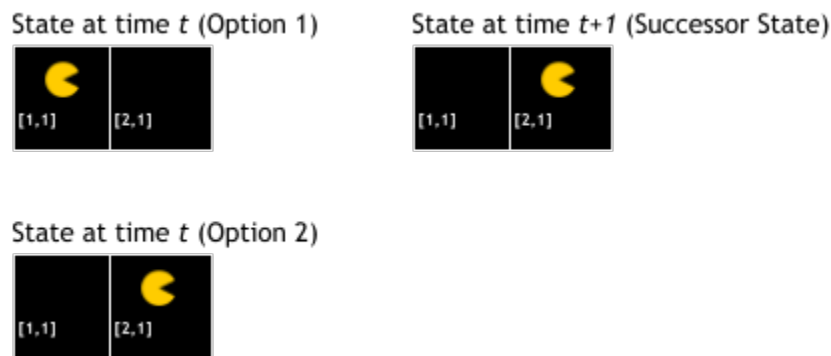


Figure 1: Mini Pacman Grid

Successor-state axiom: $Pacman_{(2,1)}^{t+1} \iff (Pacman_{(1,1)}^t \wedge Right^t) \vee (Pacman_{(2,1)}^t \wedge \neg Left^t)$

F^{t+1} is $Pacman_{(2,1)}^{t+1}$

$ActionCausesF^t$ is $Right^t$

$(F^t \wedge \neg ActionCausesNotF^t)$ is $(Pacman_{(2,1)}^t \wedge \neg Left^t)$

3. Suppose that at time 0, Pacman is somewhere on a 5x5 grid ((1,1) at the bottom left, (5,5) at the top right) with only walls on the borders.

For each of the following, state whether the entailment relation is correct. Explain your reasoning.

- (a) $Up^t \vee Right^t \models \neg Pacman_{(1,1)}^{t+1}$

True, there is no square that would lead to square (1,1) after moving up or right

(b) $\neg Pacman_{(1,1)}^{t+1} \models Up^t \vee Right^t$

False, a counterexample would be starting at square (3,2), and an action left leading to square (2,2)

(c) $Up^0 \wedge Up^2 \wedge Up^3 \models Pacman_{(x,y)}^4 : x \in [1, 5], y \in [4, 5]$

False, this is almost true, however, if Pacman starts at row 1 and the action at step 1 was down, Pacman would end at row 3

(d) $Up^t \wedge Right^t \models \neg Pacman_{(5,5)}^{t+1}$

True

There is no model that fits the action at a time step being both Up and Right. Therefore, for every model that fits this, the right side must also be true
(similar to being vacuously true for implications)

(e) $\neg Pacman_{(5,5)}^{t+1} \models Up^t \wedge Right^t$

False, since there is no model such that the right side is true, and there is at least one model such that the left side is true

(f) $Down^{t+1} \wedge Left^{t+1} \models Up^t \wedge Right^t$

True, since there are no valid models in the left or the right
(similar to $False \implies False$)

2 Compare and Contrast

1. What are some ways to find a plan using a classical planning environment model?

Naive search (BFS), linear planning/non-linear planning, graph plan.

2. What classical planning assumptions are relaxed when using the GraphPlan heuristic? Why is this helpful compared to naive search?

We are assuming we can take multiple non-mutex actions at the same time.

This is useful since in this environment, taking multiple steps at a time will allow us to add multiple goals, finishing the search problem much quicker than the traditional one action method

(Also, if we return a plan that requires we take multiple actions at the same time, we can take them in any order with the same effect since they are non-mutex)

3 Vocabulary Check

Define each of the following terms in context of Mutex within GraphPlan:

1. Interference

One action's effect negates a precondition of the other.

2. Inconsistent effects/Inconsistency

One action's effect negates an effect of the other.

3. Competing Needs

One action's precondition is in mutex with a precondition of the other.

4. Negation

Two conditions are the negation of each other

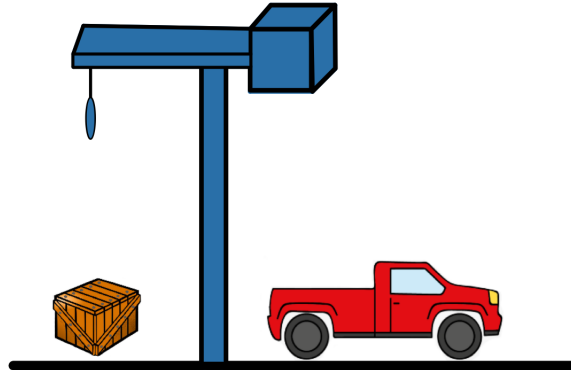
5. Inconsistent Support

Every pair of actions that produces the two conditions are in mutex with each other.

4 Graph Plan - Crate Problem

In the Crane problem, you are given a crane, a package and a truck. The package starts on the left, the truck on the right, and the crane faces the left. The goal of this is to load the package onto the truck and have the crane be facing the left.

The crane can swing between left and right, with or without a payload, and it can pick up the crate if it is on the same side. The crate can only be loaded onto the truck using the crane.



(a) Define the actions and predicates for this scenario.

Predicates: On(Crane, Left), On(Crane, Right), On(Crate, Crane), On(Crate, Ground), On(Crate, Truck)

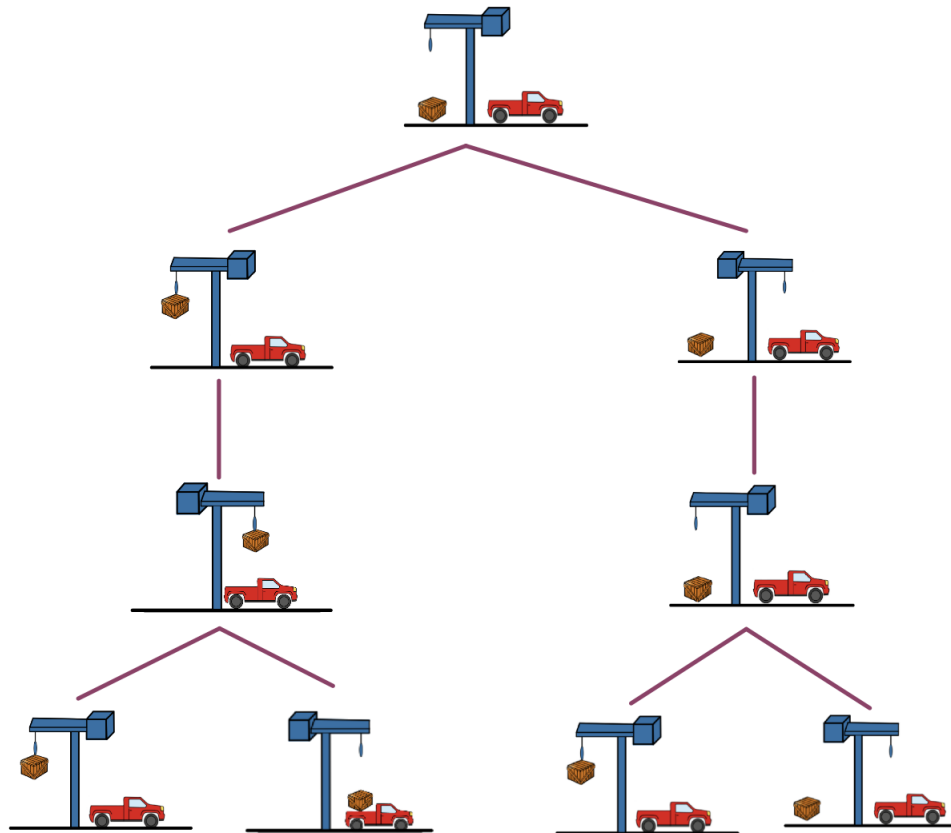
Actions: Pickup : $\begin{cases} \text{Pre: On(Crane, Left), On(Crate, Ground)} \\ \text{Add: On(Crate, Crane)} \\ \text{Del: On(Crate, Ground)} \end{cases}$

Swing Right : $\begin{cases} \text{Pre: On(Crane, Left)} \\ \text{Add: On(Crane, Right)} \\ \text{Del: On(Crane, Left)} \end{cases}$

Swing Left : $\begin{cases} \text{Pre: On(Crane, Right)} \\ \text{Add: On(Crane, Left)} \\ \text{Del: On(Crane, Right)} \end{cases}$

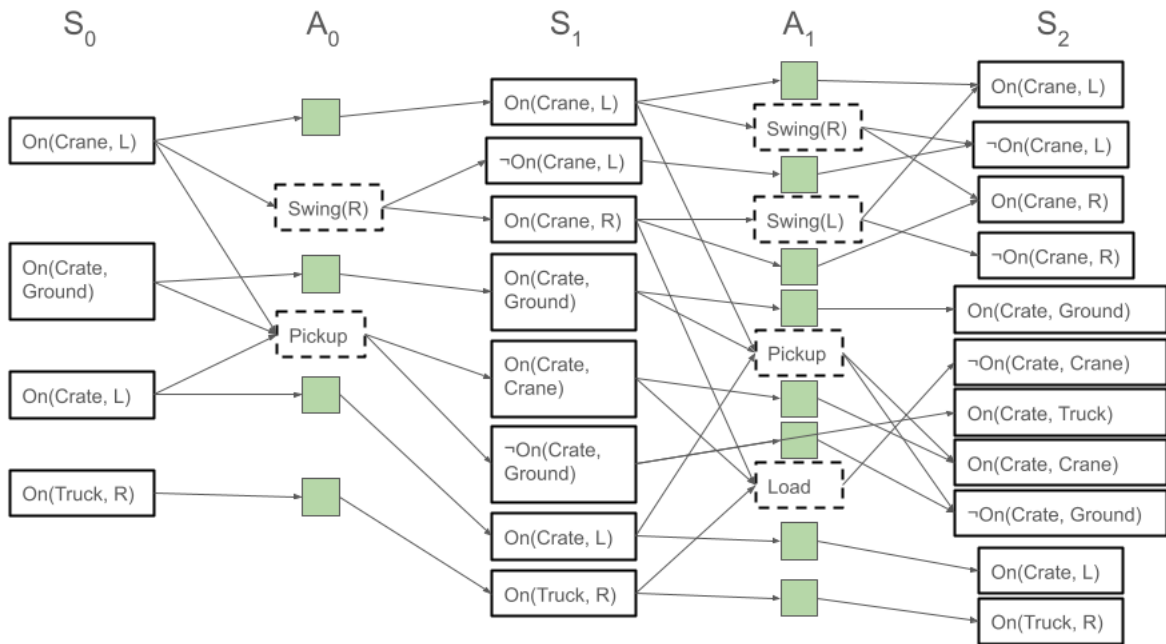
Load : $\begin{cases} \text{Pre: On(Crate, Crane), On(Crane, Right)} \\ \text{Add: On(Crate, Truck)} \\ \text{Del: On(Crate, Crane)} \end{cases}$

(b) Draw the state graph for the first 3 moves. You may use pictures instead of propositions.



(c) Draw the first two levels of the Graph Plan graph.

In the following diagram, the green squares in the action levels represent no-op's.



(d) Based on the above graph, list two actions that are mutex via interference

In the level A_0 , Swing(R) and Pickup interfere with each other. Any solution where one action's effect is the negation of a precondition of the other is correct.

- (e) Based on the above graph, list two actions that are mutex via inconsistent effects

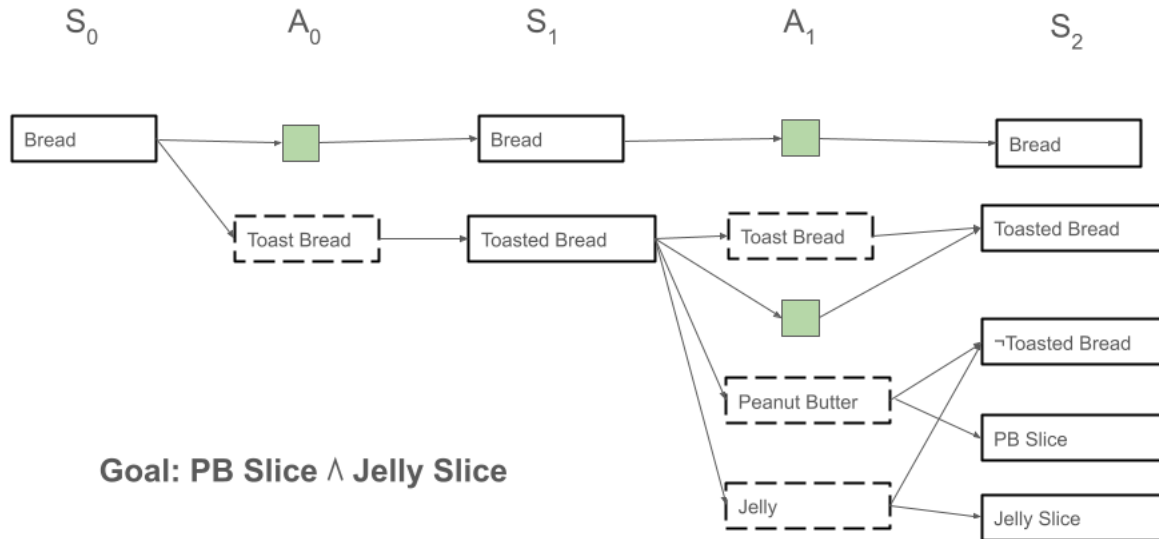
In the level A_0 , Swing(R) and $\text{No-op(On(Crane, L))}$ have inconsistent effects because their effects are negations of each other. Any solution where the two actions point to states that are negations of each other is correct.

- (f) Based on the above graph, list two conditions that are mutex via inconsistent support

In the level S_1 , On(Crate, Ground) and On(Crate, Crane) are in mutex. The only pair of actions that can produce them at A_0 is Pickup and $\text{No-op(On(Crate, Ground))}$. These actions are in mutex because they have inconsistent effects - they produce conditions that are negations of each other.

5 Estimating Plan Cost

Imagine you are trying to make a peanut butter sandwich. You have unlimited plain slices of bread which can be toasted and then applied peanut butter/jelly to. The goal conditions are $\text{PB Slice} \wedge \text{Jelly Slice}$. Assume that each action has a cost of 1. The planning graph for this scenario is given below:



- (a) What arrows would be removed by converting this graph into Π^+ , the relaxed planning graph?

Relaxing the graph removes all the delete edges so it would remove all the arrows to $\neg\text{Toasted Bread}$

- (b) Compute h^{max} and h^{add} . Which of these heuristics, if any, are admissible?

$h^{max} = 2$ since that is the maximum cost of achieving a single subgoal. It is admissible because the total cost of achieving a conjunction of goals can never be less than the maximum cost of achieving any one subgoal. $h^{sum} = 4$ since that is the sum of the costs of achieving each subgoal from the start. This is not admissible because it is very possible for there to be shared work which reduces the total cost.

- (c) What is the optimal plan based on this relaxed graph?

```

RELAXED-PLAN( $s, G$ )
  build the relaxed planning graph  $F_0 \subseteq F_1 \subseteq \dots \subseteq F_m$  until  $G \subseteq F_m$ 
   $lev(p) \leftarrow$  the first layer in which  $p$  appears
   $Goals_i \leftarrow \{g \in G : lev(g) = i\}$  for each  $i$ 
  for  $i = m$  down to 1 do
    for each  $g \in Goals_i$  do
      choose an  $a$  with  $g \in ADD(a)$  and all preconditions at level  $< i$ 
      add  $a$  to  $RP_{i-1}$ , and add each  $p \in PRE(a)$  to  $Goals_{lev(p)}$ 
  return  $RP$ 

```

Because the relaxed plan does not take into account mutexes, it returns a plan of cost 3, namely to toast bread, apply peanut butter, and then apply jelly.

6 Sampling-based motion planning with RRT

6.1 Scenario

A warehouse uses small wheeled robots to carry items from a pick station to a dispatch chute. The robot can drive in any direction from a standstill, so we never have to plan which way it is facing. Only the position of its center matters, and a single pair of numbers $\mathbf{q} = (x, y)$ completely describes it.

The robot is treated as a point. It is really a disc of radius 0.4 m, but rather than track that radius during planning we grow each obstacle outward by 0.4 m on every side before planning starts. A path that keeps the centre point out of the grown obstacles will keep the real robot from touching the real obstacles.

The bay is 12×10 metres and holds two obstacles. One is attached to the south wall and the other to the north wall. You can see a visualization below.

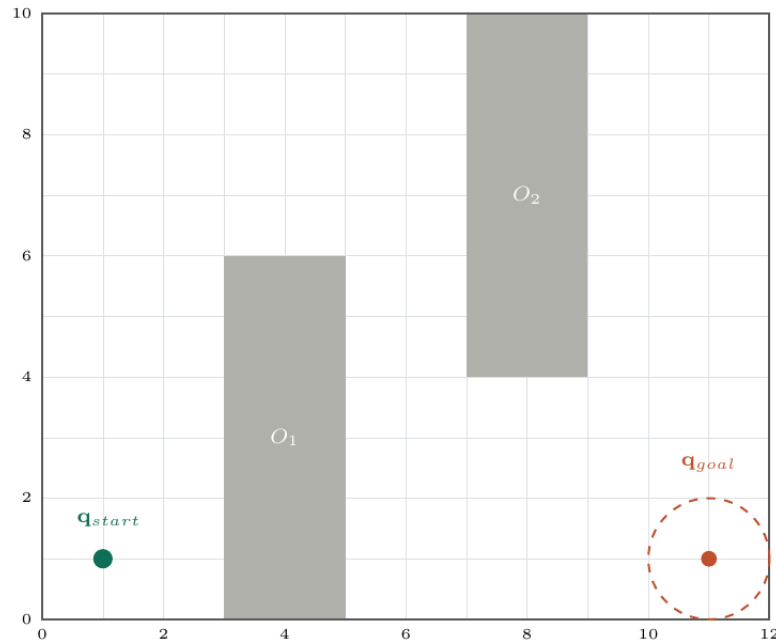
6.2 Problem specification

Symbol	Value	Meaning
$\mathcal{C}$	$[0, 12] \times [0, 10]$	configuration space, the bay floor
O_1	$[3, 5] \times [0, 6]$	obstacle on the south wall
O_2	$[7, 9] \times [4, 10]$	obstacle on the north wall
$\mathcal{C}_{obs}$	$O_1 \cup O_2$	obstacle region
$\mathcal{C}_{free}$	$\mathcal{C} \setminus \mathcal{C}_{obs}$	free space
$\mathbf{q}_{start}$	$(1, 1)$	pick station
$\mathbf{q}_{goal}$	$(11, 1)$	dispatch chute
r	1.0	goal radius: the chute accepts a tote within this distance
ε	2.0	step size: the controller accepts waypoints at most this far apart
p	0.10	goal bias

Two further choices complete the specification:

- **Distance metric:** Euclidean, $\|\mathbf{q}_a - \mathbf{q}_b\| = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$.
- **Local planner:** the straight segment between two configurations.

Grid lines below are one meter apart; the dashed circle is the goal region.



6.3 The algorithm

Each iteration performs six steps.

1. SAMPLE with prob. p : $q_{\text{rand}} = q_{\text{goal}}$; else $q_{\text{rand}} = \text{uniform}(\mathcal{C})$
 2. NEAREST $q_{\text{near}} = \text{argmin over all nodes of } \text{dist}(\text{node}, q_{\text{rand}})$
 3. STEER $d = \text{dist}(q_{\text{near}}, q_{\text{rand}})$
 if $d \leq \epsilon$: $q_{\text{new}} = q_{\text{rand}}$
 else: $q_{\text{new}} = q_{\text{near}} + \epsilon \cdot (q_{\text{rand}} - q_{\text{near}}) / d$
 4. COLLISION if $\text{collides}(q_{\text{near}}, q_{\text{new}})$: discard, go to 1
 5. ADD append q_{new} with $\text{parent}(q_{\text{new}}) = q_{\text{near}}$
 6. GOAL TEST if $\text{dist}(q_{\text{new}}, q_{\text{goal}}) \leq r$: extract path and stop
- EXTRACT follow parent pointers from the final node to the root, reverse

6.4 Configuration space and terminology

- (a) What is the dimension of $\mathcal{C}$ here, and what would it be if the tote were carried by a six-joint robot arm instead? Which parts of the algorithm above would have to change?

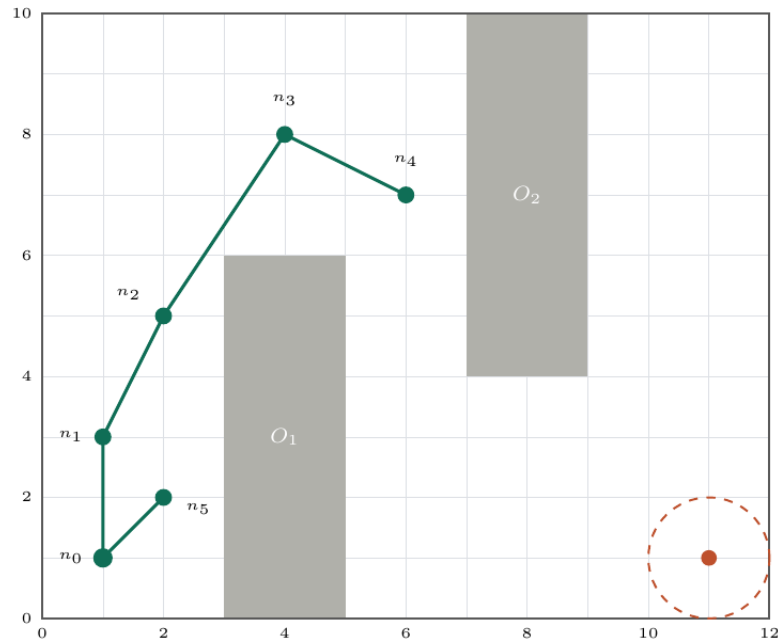
Two, since a configuration is (x, y) . A six-joint arm has $\mathcal{C} \subseteq \mathbb{R}^6$, one dimension per joint angle. No line of the algorithm changes. Only the distance metric and the collision checker are replaced; the search structure is identical. This is precisely why sampling-based planning scales to high dimensions.

- (b) In one sentence each, state the role of ϵ and the role of the goal radius r . Why is confusing the two a bug rather than a matter of taste?

ε bounds how far the tree may extend in a single iteration, so it controls the resolution of the search and here reflects a controller limit. r defines when the query counts as solved, and here reflects the physical tolerance of the chute. Shrinking ε makes the tree finer, whereas shrinking r makes termination stricter; they act on different parts of the loop and are not interchangeable.

6.5 Nearest neighbour and steering

The following tree, called Tree A, has been grown in the bay described above. All six nodes and all five edges are collision-free; you may assume this.



node	n_0	n_1	n_2	n_3	n_4	n_5
(x, y)	(1, 1)	(1, 3)	(2, 5)	(4, 8)	(6, 7)	(2, 2)
parent	—	n_0	n_1	n_2	n_3	n_0

- (a) The sampler returns $\mathbf{q}_{rand} = (6, 3)$. Compute the distance from $\mathbf{q}_{rand}$ to all six nodes, identify $\mathbf{q}_{near}$, and state the runner-up together with its margin of loss.

$$\begin{aligned}
 n_0 : \sqrt{25 + 4} &= 5.385, & n_1 : \sqrt{25 + 0} &= 5.000, & n_2 : \sqrt{16 + 4} &= 4.472, \\
 n_3 : \sqrt{4 + 25} &= 5.385, & n_4 : \sqrt{0 + 16} &= \mathbf{4.000}, & n_5 : \sqrt{16 + 1} &= 4.123.
 \end{aligned}$$

$\mathbf{q}_{near} = n_4 = (6, 7)$. Runner-up is n_5 at 4.123, losing by only 0.123; a sketch would not settle this, the distances must be computed.

- (b) Using your $\mathbf{q}_{near}$ from part (a), compute $\mathbf{q}_{new}$. State which branch of step 3 fires.

$d = 4.000 > \varepsilon = 2.0$, so the partial-step branch fires:

$$\hat{u} = \frac{(6, 3) - (6, 7)}{4.000} = (0, -1), \quad \mathbf{q}_{new} = (6, 7) + 2.0(0, -1) = (6, 5).$$

- (c) Now suppose instead $\mathbf{q}_{near} = n_3 = (4, 8)$ and $\mathbf{q}_{rand} = (5, 9)$. Compute d and $\mathbf{q}_{new}$. How far did the tree actually grow, and what does this say about the claim “every edge has length ε ”?

$d = \sqrt{1+1} = 1.414 \leq \varepsilon$, so the first branch fires and $\mathbf{q}_{new} = \mathbf{q}_{rand} = (5, 9)$ exactly. The tree grew 1.414, not 2.0.

The claim is false. ε is an upper bound on edge length, not the edge length: $\|\mathbf{q}_{new} - \mathbf{q}_{near}\| = \min(\varepsilon, d)$. Edges are shorter whenever the sample lands within ε of the tree, which happens more and more often as the tree becomes dense.

6.6 Collision checking

The parts below start from $\mathbf{q}_{near} = (6, 5)$, the node produced in Section 6.5, in the same bay.

- (a) $\mathbf{q}_{rand} = (10, 5)$. Compute $\mathbf{q}_{new}$ and decide whether the extension is accepted. If it is rejected, name the obstacle and state exactly what the tree looks like after this iteration.

$d = 4.000 > \varepsilon$, $\hat{u} = (1, 0)$, so $\mathbf{q}_{new} = (6, 5) + 2(1, 0) = (8, 5)$.

Here $x = 8 \in [7, 9]$ and $y = 5 \in [4, 10]$, so $\mathbf{q}_{new} \in O_2$. **Rejected.** Nothing is added and the tree is unchanged; the iteration is simply wasted. This is the geometric point of the environment: having cleared O_1 you cannot travel straight right, because O_2 hangs down into the way.

- (b) $\mathbf{q}_{rand} = (7, 2)$. Compute $\mathbf{q}_{new}$ and decide whether the extension is accepted.

$d = \sqrt{1+9} = 3.162 > \varepsilon$, so

$$\hat{u} = \frac{(1, -3)}{3.162} = (0.316, -0.949), \quad \mathbf{q}_{new} = (6 + 0.632, 5 - 1.898) = (6.63, 3.10).$$

The endpoint has $x = 6.63 \notin [3, 5] \cup [7, 9]$, and the whole segment stays in $6 \leq x \leq 6.63$, clear of both walls. **Accepted;** add with parent $(6, 5)$.

6.7 Goal test and the goal bias

In each part the goal bias fires, so $\mathbf{q}_{rand} = \mathbf{q}_{goal} = (11, 1)$ exactly. The bay and the parameters are unchanged.

- (a) $\mathbf{q}_{near} = (10, 2)$. Compute d , $\mathbf{q}_{new}$ and $\|\mathbf{q}_{new} - \mathbf{q}_{goal}\|$. Does the algorithm terminate this iteration?

$d = \sqrt{1+1} = 1.414 \leq \varepsilon$, so the first branch of step 3 fires and $\mathbf{q}_{new} = \mathbf{q}_{rand} = (11, 1)$. Then $\|\mathbf{q}_{new} - \mathbf{q}_{goal}\| = 0 \leq 1.0$, and the segment from $(10, 2)$ to $(11, 1)$ is clear of both walls. **Yes**, it terminates, landing on the goal exactly.

- (b) $\mathbf{q}_{near} = (10, 4.5)$. Repeat the computation. Does it terminate?

$d = \sqrt{1+12.25} = 3.640 > \varepsilon$, so

$$\hat{u} = \frac{(1, -3.5)}{3.640} = (0.275, -0.962), \quad \mathbf{q}_{new} = (10.55, 2.58).$$

$\|\mathbf{q}_{new} - \mathbf{q}_{goal}\| = \sqrt{0.55^2 + 1.58^2} = 1.643 > 1.0$. **No**. The node is added and the loop continues.

6.8 Path extraction

A later run in the same bay terminates with the tree tabulated below. Only the parent table is given.

node	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
x	1.00	1.00	2.00	2.00	4.00	3.00	6.00	6.00	6.63	8.43	10.23
y	1.00	3.00	5.00	2.00	8.00	7.00	7.00	5.00	3.10	2.23	1.37
parent	—	n_0	n_1	n_0	n_2	n_2	n_4	n_6	n_7	n_8	n_9

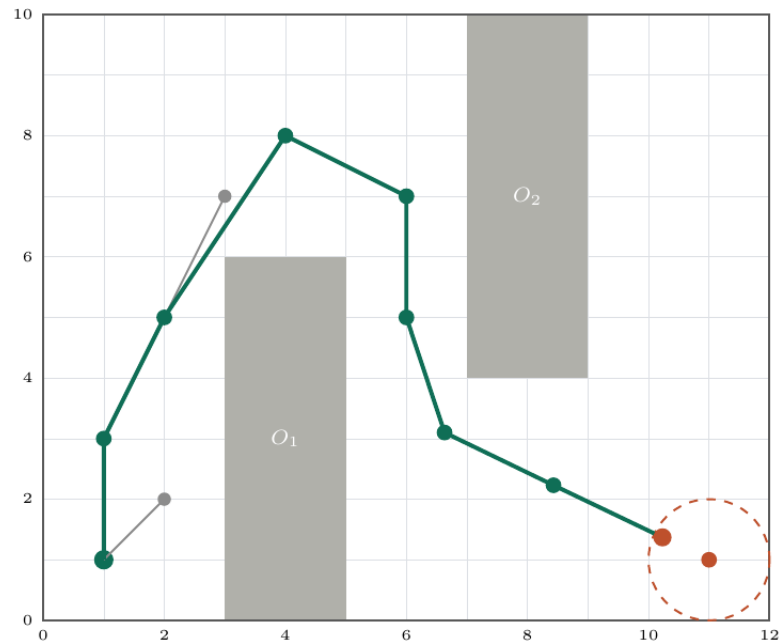
- (a) Extract the solution path by following parent pointers. List the nodes in execution order.

Walking parents from n_{10} gives $10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$. Reversed:

$$n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow n_4 \rightarrow n_6 \rightarrow n_7 \rightarrow n_8 \rightarrow n_9 \rightarrow n_{10},$$

that is $(1, 1) \rightarrow (1, 3) \rightarrow (2, 5) \rightarrow (4, 8) \rightarrow (6, 7) \rightarrow (6, 5) \rightarrow (6.63, 3.10) \rightarrow (8.43, 2.23) \rightarrow (10.23, 1.37)$.

Note that no search is performed at extraction time; a single walk up the parent chain suffices, which is the reason for storing the tree as parent pointers.



(b) Which nodes are not on the path? Is their presence a bug?

$n_3 = (2, 2)$ and $n_5 = (3, 7)$ are dead-end branches. Not a bug. RRT cannot know in advance which branch will reach the goal, so exploratory branches that lead nowhere are the expected cost of the method. Here 11 nodes were built and 9 were used.

6.9 Behavior under changed parameters

All parts refer to the same bay. Answer in two or three sentences each.

(a) The step size is changed to $\varepsilon = 0.1$, and then to $\varepsilon = 8$. Describe what goes wrong in each case.

With $\varepsilon = 0.1$ the frontier advances by at most 0.1 per iteration, so covering the roughly 15-unit corridor needs on the order of 150 successful extensions plus rejections, multiplying the number of nearest-neighbour queries. Curiously the rejection rate near walls falls, since short steps rarely penetrate an obstacle.

With $\varepsilon = 8$ most extensions span the whole room and are rejected for cutting through a wall, so the tree barely grows, and the aisle, which is narrower than 8, becomes nearly impossible to thread.

- (b) The goal bias is set to $p = 0$, and then to $p = 1.0$. In each case, is the algorithm still probabilistically complete, and what does the tree actually do?

With $p = 0$ it remains probabilistically complete, since uniform sampling gives every open set, including the goal region, positive probability. It is impractical here though: the goal region has area $\pi \approx 3.14$ out of 96 free units, so hits are uncommon.

With $p = 1.0$ exploration stops entirely. Every iteration steers from the node nearest the goal straight toward it, so the tree becomes a single chain that drives into the first wall in the way and then stalls: the nearest node never changes and the same rejected extension is retried forever. Completeness is lost.

- (c) RRT is probabilistically complete but not optimal. Define each term precisely, and state which one RRT* changes and how.

Probabilistically complete: if a solution exists, the probability of finding one tends to 1 as the number of samples tends to infinity. It gives no finite-time guarantee and can never report that no solution exists. Not optimal: the returned path may be arbitrarily worse than the shortest one, and running longer does not improve a path already returned.

RRT* changes the second. By rewiring, that is, by checking whether each new node offers a cheaper parent to nearby nodes and vice versa, it becomes asymptotically optimal: the cost of its solution converges to the optimum as samples increase. It does not improve completeness, which was already as strong as sampling permits.