COMS 4733 Homework 2

Jaisel Singh

October 9, 2025

1 Problem 1: Discrete Search Algorithms

1.1 Search Algorithm Comparison

Notation:

- b = branching factor (average number of successors per node)
- d = depth of the shallowest goal node
- m = maximum depth of the search tree

Note: All algorithms use graph search (visited set) on finite state space.

Algorithm	**Complete?**	**Cost-Optimal?**	**Time Complexity**	**Space Complexity**
DFS	Yes	No	$O(b^m)$	$O(bm)$
BFS	Yes	No	$O(b^d)$	$O(b^d)$
Dijkstra	Yes	Yes	$O(b^d)$	$O(b^d)$
A* (admissible h)	Yes	Yes	$O(b^d)$	$O(b^d)$

Table 1: Comparison of search algorithms

1.1.1 Justification for Selected Entries

Dijkstra's Algorithm - Time Complexity: $O(b^d)$

Dijkstra's algorithm explores nodes in order of increasing cost from the start node. In the worst case, it must explore all nodes up to the depth d of the optimal solution before guaranteeing it has found the cheapest path.

Consider a tree structure where:

- At depth 0: 1 node (start)

- At depth 1: b nodes
- At depth 2: b^2 nodes
- $\vdots$
- At depth d: b^d nodes

The total number of nodes explored is:

$$1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

Each node is processed once (removed from the priority queue), and for each of its neighbors, we may perform a priority queue operation. With an efficient priority queue implementation (binary heap), these operations take $O(\log n)$ time where n is the number of nodes in the queue.

Therefore, the overall time complexity is $O(b^d)$ in the context of search algorithms.

Dijkstra's Algorithm - Space Complexity: $O(b^d)$

Dijkstra's algorithm uses a priority queue to store nodes that have been discovered but not yet fully explored. In the worst case, before reaching the goal at depth d, the priority queue may contain all nodes at the frontier of the search.

Since Dijkstra's explores in a breadth-first manner (ordered by cost rather than depth), it must maintain all nodes at the current cost level. In a tree with branching factor b and goal at depth d, the maximum number of nodes in the priority queue at once is proportional to b^d (the number of nodes at depth d).

Additionally, with graph search (using a visited set to avoid cycles), we must store all visited nodes, which is also $O(b^d)$ in the worst case.

Therefore, the space complexity is $O(b^d)$.

1.2 Search Tree Expansion

1.2.1 Task 1: BFS Expansion Order

Using lexicographic ordering (smallest x first, then smallest y) for tie-breaking, the BFS expansion order is:

(0,3) 5	**(1,3)** 8	**(2,3)** 12	**G (3,3)** 14
(0,2) 2	**X (1,2)** -	**(2,2)** 9	**(3,2)** 13
S (0,1) 0	**(1,1)** 3	**(2,1)** 6	**(3,1)** 10
(0,0) 1	**(1,0)** 4	**(2,0)** 7	**(3,0)** 11

Complete expansion sequence:

$$0 : S(0,1) \to 1 : (0,0) \to 2 : (0,2) \to 3 : (1,1) \to 4 : (1,0) \to 5 : (0,3) \to 6 : (2,1)$$
$$\to 7 : (2,0) \to 8 : (1,3) \to 9 : (2,2) \to 10 : (3,1) \to 11 : (3,0) \to 12 : (2,3)$$
$$\to 13 : (3,2) \to 14 : G(3,3)$$

1.2.2 Task 2: Dijkstra's Algorithm

Dijkstra's algorithm expands nodes in the **same order** as BFS for this problem.

Explanation:

BFS and Dijkstra's expand nodes identically because all edges have uniform cost (cost = 1). Dijkstra's algorithm uses a priority queue to explore nodes in order of increasing cumulative cost from the start. Since each move costs 1, a node at distance d from the start has total cost d. This matches BFS exactly, which explores nodes layer-by-layer by distance.

When multiple nodes have the same priority (same cost in Dijkstra's, same depth in BFS), both algorithms apply the lexicographic tie-breaking rule. Since cost equals depth in this uniform-cost scenario, both algorithms break ties identically and produce the same expansion order.

Therefore, Dijkstra's expansion order is identical to the BFS grid shown above.

1.2.3 Task 3: A* with Manhattan Distance Heuristic

Using the Manhattan distance heuristic $h(x, y) = |3 - x| + |3 - y|$ and lexicographic tie-breaking, A* expands nodes in the following order:

Order	**Node**	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
0	S (0,1)	0	5	5
1	(0,2)	1	4	5
2	(0,3)	2	3	5
3	(1,1)	1	4	5
4	(1,3)	3	2	5
5	(2,1)	2	3	5
6	(2,2)	3	2	5
7	(2,3)	4	1	5
8	(3,1)	3	2	5
9	(3,2)	4	1	5
10	G (3,3)	5	0	5

Grid with expansion order:

(0,3) 2	**(1,3)** 4	**(2,3)** 7	**G (3,3)** 10
(0,2) 1	**X (1,2)** -	**(2,2)** 6	**(3,2)** 9
S (0,1) 0	**(1,1)** 3	**(2,1)** 5	**(3,1)** 8
(0,0) -	**(1,0)** -	**(2,0)** -	**(3,0)** -

Observations/Notes

- All expanded nodes have $f(n) = 5$, which equals the optimal path cost
- Nodes in row $y = 0$ (bottom row) such as (0,0), (1,0), (2,0), and (3,0) are generated but never expanded because they all have $f = 7 > 5$
- The Manhattan distance heuristic is admissible (never overestimates) and consistent in this 4-connected grid with unit costs
- Because h is admissible and the grid uses 4-connected movement with unit costs, every node on an optimal path satisfies $g + h = 5$ (constant). Therefore, the expansion order among these nodes is determined purely by the lexicographic tie-breaking rule
- A* reaches the goal at step 10, having expanded only 11 nodes (including the start), compared to 15 nodes for BFS

1.3 Heuristic Admissibility

1.3.1 Is Euclidean distance admissible?

Answer: Yes, the Euclidean distance heuristic $h(n) = \sqrt{(x_g - x)^2 + (y_g - y)^2}$ is admissible for this cost model.

Justification:

A heuristic is admissible if it never overestimates the true optimal cost to reach the goal. The Euclidean distance represents the straight-line distance between two points, which is the shortest possible distance in Euclidean space.

With 8-connected movement where diagonal moves cost $\sqrt{2}$ and cardinal moves cost 1, we can analyze the relationship between the heuristic and actual cost:

- Consider moving from $(0, 0)$ to $(3, 3)$:
  - Euclidean distance: $h = \sqrt{(3-0)^2 + (3-0)^2} = \sqrt{18} = 3\sqrt{2} \approx 4.243$
  - Optimal path: Three diagonal moves $(0, 0) \to (1, 1) \to (2, 2) \to (3, 3)$
  - Actual cost: $3\sqrt{2} \approx 4.243$
- In general, for any two points, the Euclidean distance equals the cost of moving in a straight diagonal line (when possible), which is the optimal path in an obstacle-free 8-connected grid
- Since we cannot move "more directly" than the straight line, and diagonal moves have cost exactly $\sqrt{2}$, the Euclidean distance never overestimates the true cost

Therefore, the Euclidean distance heuristic is admissible for this cost model.

1.3.2 Propose a different admissible heuristic

Answer: The Chebyshev distance (L^∞ norm) is an admissible heuristic for this cost model:

$$h(n) = \max(|x_g - x|, |y_g - y|)$$

Justification:

The Chebyshev distance measures the maximum absolute difference in either coordinate, which corresponds to the minimum number of moves required to reach the goal when diagonal moves are allowed.

To show admissibility, we verify that this heuristic never overestimates:

- The Chebyshev distance counts the number of diagonal or cardinal moves needed along the longer dimension
- Example: From $(0, 0)$ to $(3, 3)$:
  - Chebyshev distance: $h = \max(|3 - 0|, |3 - 0|) = 3$
  - Optimal path: Three diagonal moves with cost $3\sqrt{2} \approx 4.24$
  - Since $3 < 4.24$, the heuristic does not overestimate
- More generally, consider moving from (x, y) to (x_g, y_g):
  - Let $\Delta x = |x_g - x|$ and $\Delta y = |y_g - y|$
  - Assume without loss of generality that $\Delta x \geq \Delta y$
  - Optimal strategy: Move diagonally Δy times, then cardinally $(\Delta x - \Delta y)$ times
  - Actual cost: $\Delta y \cdot \sqrt{2} + (\Delta x - \Delta y) \cdot 1 = \Delta x + \Delta y(\sqrt{2} - 1)$
  - Chebyshev distance: $h = \Delta x$
  - Since $\sqrt{2} - 1 \approx 0.414 > 0$, we have $h = \Delta x < \Delta x + \Delta y(\sqrt{2} - 1)$ $=$ actual cost

Therefore, the Chebyshev distance never overestimates the true cost and is admissible.

Alternative: Any scaled Manhattan distance of the form $h(n) = c \cdot (|x_g - x| + |y_g - y|)$ where $c \leq \frac{\sqrt{2}}{2} \approx 0.707$ is also admissible, though it provides a looser bound than Chebyshev distance.

1.4 PRM Construction and Search

1.4.1 Task 1: Discard samples inside obstacle

The obstacle is defined by corners at $(4, 4)$ and $(6, 6)$, forming a rectangle where $4 \leq x \leq 6$ and $4 \leq y \leq 6$.

Checking each sample:

- $(2, 2)$: Outside obstacle

- $(3,7)$: Outside obstacle
- $(5,2)$: Outside obstacle
- $(7,3)$: Outside obstacle
- $(8,5)$: Outside obstacle
- $(5,8)$: Outside obstacle
- $(3,3)$: Outside obstacle
- $(8,8)$: Outside obstacle
- $(5,5)$: **Inside obstacle - discarded**

Result: 8 valid samples remain after discarding $(5,5)$.

1.4.2 Task 2: Connect samples within radius $r = 2.5$

Computing Euclidean distances between all pairs of valid samples (including start and goal), we connect pairs with distance ≤ 2.5:

Edges created:

- Start $(1,1)$ to $(2,2)$: $d = \sqrt{2} \approx 1.41$
- $(2,2)$ to $(3,3)$: $d = \sqrt{2} \approx 1.41$
- $(3,3)$ to $(5,2)$: $d = \sqrt{5} \approx 2.24$
- $(3,7)$ to $(5,8)$: $d = \sqrt{5} \approx 2.24$
- $(5,2)$ to $(7,3)$: $d = \sqrt{5} \approx 2.24$
- $(7,3)$ to $(8,5)$: $d = \sqrt{5} \approx 2.24$
- $(8,8)$ to Goal $(9,9)$: $d = \sqrt{2} \approx 1.41$

1.4.3 Task 3: Resulting roadmap

The constructed PRM graph has two disconnected components:

- **Left component:** Start $(1,1) \to (2,2) \to (3,3) \to (5,2) \to (7,3) \to (8,5)$ and $(3,7) \to (5,8)$
- **Right component:** $(8,8) \to$ Goal $(9,9)$

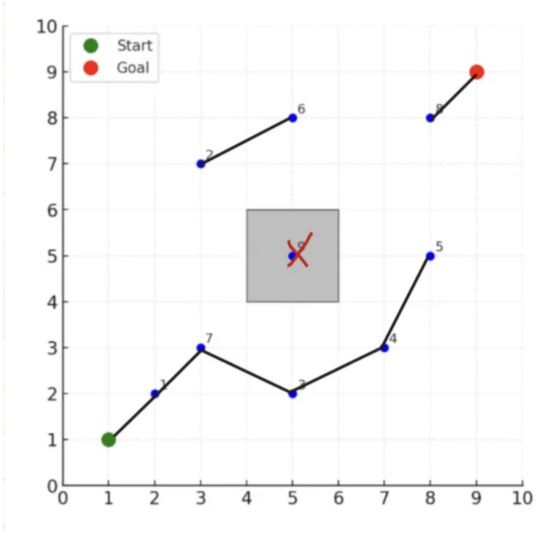

Figure 1: Constructed PRM roadmap showing valid samples (blue), start (green), goal (red), obstacle (gray), and edges connecting nodes within radius $r = 2.5$.

1.4.4 Task 4: Search algorithm

Running Dijkstra's algorithm from Start $(1, 1)$:

The algorithm explores all nodes reachable from the start but cannot reach the goal because the graph is disconnected. The search terminates without finding a path.

1.4.5 Task 5: Results and connectivity analysis

Path length: No path exists.

Number of nodes expanded: Dijkstra's expands all nodes in the connected component containing the start: Start, $(2, 2)$, $(3, 3)$, $(3, 7)$, $(5, 2)$, $(5, 8)$, $(7, 3)$, $(8, 5)$ = 8 nodes.

Explanation: The PRM fails to find a path because the roadmap is disconnected. The obstacle at $(4, 4)$ to $(6, 6)$ combined with the connection radius $r = 2.5$ creates a gap that prevents connectivity between the left and right sides of the workspace. The samples near the obstacle are too far apart to bridge this gap.

Suggestions to improve connectivity:

1. **Increase connection radius:** Setting $r \geq 3.0$ would allow connec-

tions like $(5,8)$ to $(8,8)$ (distance $= 3.0$), bridging the gap around the obstacle

2. **Add more samples:** Generating additional random samples, particularly in the region around the obstacle (e.g., near $(7,7)$ or $(3,5)$), would increase the likelihood of creating bridging connections

3. **Use both approaches:** Combining more samples with a larger connection radius provides the most robust solution for ensuring connectivity

2 Problem 2: Rapidly-Exploring Random Trees (RRT)

2.1 Hand-Simulated RRT Growth

This section presents a hand-simulated construction of a Rapidly-Exploring Random Tree (RRT) for the workspace defined in Problem 1(d). The simulation is carried out for ten iterations, starting from the initial configuration $q_{\text{start}} = (1,1)$ and using the following parameters:

- Step size: $\eta = 1.5$
- Goal bias: $p_{\text{goal}} = 0.10$
- Collision-check resolution: $\delta = 0.25$

Random samples q_{rand} are drawn uniformly from the free configuration space, excluding the rectangular obstacle with corners at $(4,4)$ and $(6,6)$. At each iteration, the nearest node q_{near} is identified, an extension is attempted toward q_{rand} up to distance η, and the resulting segment is collision-checked before being added to the tree.

Iter	Sample Type	q_{rand}	q_{near}	q_{new}	Collision / Edge
1	Uniform	(8.18, 9.59)	(1.00, 1.00)	(1.96, 2.15)	No / Yes
2	Uniform	(5.62, 7.21)	(1.96, 2.15)	(2.84, 3.37)	No / Yes
3	Uniform	(0.54, 7.03)	(2.84, 3.37)	(2.04, 4.64)	No / Yes
4	Uniform	(7.10, 3.22)	(2.84, 3.37)	(4.34, 3.31)	No / Yes
5	Uniform	(9.48, 9.93)	(4.34, 3.31)	(5.26, 4.50)	**Yes / No**
6	Uniform	(1.83, 1.35)	(1.96, 2.15)	(1.83, 1.35)	No / Yes
7	Uniform	(8.32, 1.63)	(4.34, 3.31)	(5.72, 2.73)	No / Yes
8	Uniform	(3.12, 4.95)	(2.04, 4.64)	(3.12, 4.95)	No / Yes
9	Uniform	(0.41, 5.46)	(2.04, 4.64)	(0.70, 5.31)	No / Yes
10	Uniform	(4.33, 7.34)	(3.12, 4.95)	(3.80, 6.29)	No / Yes

Table 2: Hand-simulated RRT growth over 10 iterations. The edge in iteration 5 was discarded due to a collision with the obstacle.

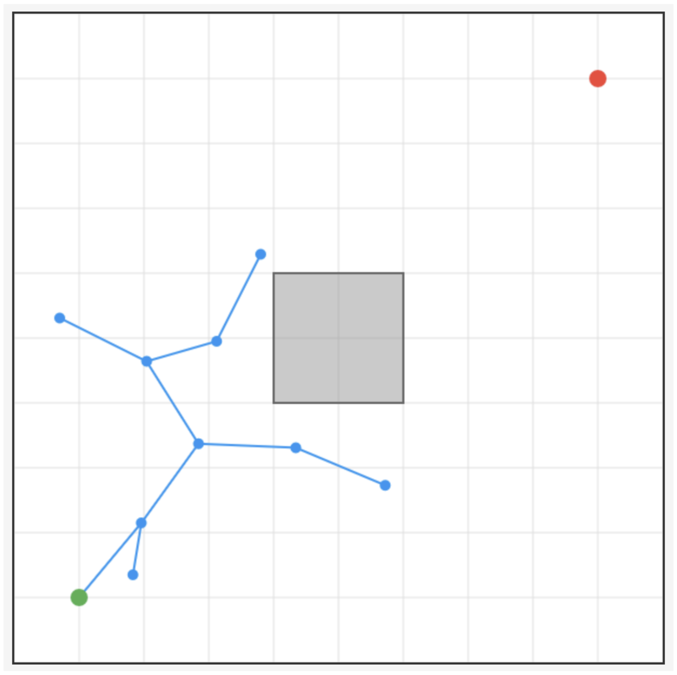

Figure 2: RRT after 10 iterations. Nodes are shown in blue, valid edges in black, the obstacle in gray, and start/goal in green/red.

Discussion. The tree expands systematically from the start toward unexplored regions of the workspace, demonstrating the RRT's characteristic space-filling behavior. Only one attempted extension (Iteration 5) intersects the obstacle and is therefore rejected. After ten iterations, the tree contains nine valid edges and ten nodes, covering much of the lower-left to mid-right workspace.

The simulated results highlight how local collision constraints restrict branch expansion near the obstacle region and how random sampling ensures broad workspace coverage even with a small number of iterations.

2.2 Goal bias and step size analysis

2.2.1 Question 1: Effect of increasing p_{goal}

(i) Speed of reaching the goal region:

Increasing p_{goal} **increases the speed** of reaching the goal. A higher goal bias means the algorithm samples the goal location more frequently, which directly pulls the tree toward the goal. This creates more edges pointing in the goal direction, accelerating convergence. In our simulation with $p_{\text{goal}} = 0.10$, no goal-biased samples occurred in 10 iterations. If we had set $p_{\text{goal}} = 0.50$, approximately 5 of the 10 samples would have been at the goal location $(9, 9)$, creating direct paths toward the target.

(ii) Diversity/coverage of explored space:

Increasing p_{goal} **decreases diversity and coverage** of the explored space. A higher goal bias creates a strong directional preference toward the goal, meaning the tree explores less of the overall workspace. The tree becomes more focused on reaching the goal and less exploratory of alternative regions. This reduces the coverage of the free space, which can be problematic if indirect paths are necessary.

When can a large p_{goal} hurt performance?

A large goal bias can hurt performance in several scenarios:

- **Narrow passages or complex obstacles:** When obstacles directly block the path between start and goal, an aggressive goal bias causes the tree to repeatedly attempt the same blocked direction, wasting samples without making progress toward finding alternative routes.

- **Exploration-dependent environments:** When the workspace requires exploring around obstacles before approaching the goal, a high p_{goal} prevents the tree from discovering these necessary detours. In our example, if the obstacle were larger or positioned directly between start and goal, we would need to explore sideways or around before heading toward $(9, 9)$.

- **Local minima:** The tree may get trapped in local configurations where all direct extensions toward the goal collide, and insufficient exploration prevents finding alternative paths.

The optimal p_{goal} balances exploitation (moving toward the goal) with exploration (discovering diverse paths through the free space).

2.2.2 Question 2: Trade-offs in choosing η (step size)

The step size η controls how far the tree extends toward each random sample. Choosing η involves important trade-offs:

If η is much larger than typical obstacle feature sizes:

- **Problem:** Large steps may "jump over" narrow passages or small gaps in obstacles, causing the tree to miss valid paths that require precision navigation
- The tree cannot navigate through tight spaces because each extension is too coarse
- May fail to find feasible paths even when they exist
- **Example:** If $\eta = 5$ in our 10×10 workspace, a single step from $(1, 1)$ could reach $(6, 1)$, potentially jumping through the obstacle boundary without the collision checker detecting intermediate points properly

If η is much smaller than obstacle feature sizes:

- **Problem:** Very slow progress and inefficient exploration of the workspace
- Takes many iterations to make meaningful distance toward the goal
- Computational cost increases dramatically as more nodes are needed
- **Example:** If $\eta = 0.1$, traversing the diagonal distance from $(1, 1)$ to $(9, 9)$ (approximately 11.3 units) would require over 100 iterations just for the direct path, not accounting for obstacle avoidance

Trade-off and rule of thumb:

The optimal η should be proportional to the smallest obstacle features or clearances that must be navigated:

- **Rule of thumb:** Set $\eta \approx 0.5$ to 2 times the minimum clearance or gap width
- Too large: miss narrow passages
- Too small: inefficient, slow convergence
- For our workspace with a 2×2 obstacle in a 10×10 space, $\eta = 1.5$ provides a reasonable balance between exploration efficiency and the ability to navigate around the obstacle

The step size should be large enough to make efficient progress but small enough to respect the geometric constraints of the environment.

2.3 Collision Checking Resolution