CS 4100 5100

Problem Set 1

Instructions

- All submissions must be typed. No exceptions are made to this rule.
- Hand-drawn figures are acceptable only where specified in the question, and provided all labels are clear and legible. If we can't read your text, we can't assign points. Please scan and insert any such figures in the final PDF document.

Academic Integrity

- If you discuss concepts related to this assignment with one or more classmates, all parties must declare collaborators in their individual submissions. Such discussions must be kept at a conceptual level, and no sharing of written answers is permitted. Do not discuss the specific problems within this problem set.
- You may not, as a general rule, use generative AI for problem sets. Any use of GenAI that is solely intended to improve writing clarity or sentence structure is acceptable, but must be accompanied by an appendix containing your original, unedited answers. If you use generative AI in this manner, start a new page titled 'Appendix' at the end of the written submission, and paste your original answers here with the corresponding question numbers clearly indicated.
- Failure to disclose collaborations or use of generative AI will be treated as a violation of academic integrity, and penalties listed in the course syllabus will be enforced.

Reach Out!

If at any point you feel stuck with the assignment, please reach out to the TAs or the instructor, and do so early on! This lets us guide you in the right direction in a timely fashion and will help you make the most of your assignment.

Categorizing AI Environments

Q1 Under what conditions may autonomous robot navigation be considered a **fully observable** environment? What are the agent's percepts, and which algorithms are best-suited to solving this problem in this setting? (2)

Autonomous robot navigation is *fully observable* when the robot's sensors provide complete and accurate information about the environment at every time step—i.e., there is no hidden state. In this case:

- **Percepts:**
  - Range measurements (e.g., LIDAR, sonar) giving obstacle distances
  - Visual input (camera images) for landmark recognition
  - Localization data (e.g., GPS, odometry) for exact pose estimation
  - Preloaded or dynamically updated map of the workspace
- **Suitable algorithms:**
  - **A* Search:** optimal and complete with an admissible, consistent heuristic
  - **Uniform Cost Search (UCS):** handles arbitrary non-negative costs when no heuristic is available
  - **Dijkstra's Algorithm:** special case of UCS for uniform or non-negative grid costs

Q2 Under what conditions may autonomous robot navigation be considered a **partially observable** environment? What are the agent's percepts, and how would you approach this problem in terms of implementing the agent's logic? (2)

Autonomous robot navigation is *partially observable* when the robot's sensors provide only incomplete, noisy, or uncertain information about the environment. This can occur due to occlusions, limited sensor range, or sensor inaccuracies. In this case:

- **Percepts:**
  - Noisy or partial range measurements (e.g., LIDAR with occlusions)
  - Inaccurate localization (e.g., GPS drift, wheel slippage)
  - Limited field of view or resolution in camera images
- **Approach:**
  - Use a **belief state** (a probability distribution over possible states)
  - Implement **probabilistic reasoning** techniques such as:
    - **Bayesian filtering** (e.g., Kalman filter, particle filter)

    - **POMDPs** (Partially Observable Markov Decision Processes) for decision making under uncertainty
  - Continuously update the belief state based on percepts and action outcomes

Q3 Is a fully observable environment always known? Explain with an example. (2)

No, a fully observable environment is not always known. Full observability means the agent can perceive the complete state of the environment at each step, but a known environment implies prior knowledge of the environment's structure and dynamics.

Example: A robot exploring a new maze with complete sensor coverage can observe its surroundings at all times (fully observable), but it has no prior map (unknown). It learns the layout as it explores.

Q4 Is a partially observable environment always unknown? Explain with an example. (2)

No, a partially observable environment is not always unknown. An environment is partially observable if the agent cannot perceive the full state at each step, but it may still be known if the agent has prior knowledge of the environment's structure.

Example: A robot navigating a familiar building with limited-range sensors (e.g., short-range infrared) knows the layout (known), but can only sense nearby walls or obstacles (partially observable).

Q5 Many places now use a large number of drones to create light-shows instead of, or in addition to, fireworks for major holidays. Here's an example from Singapore. Is the use of drones to create a light show always a multi-agent problem? Why/why not? (2)

The use of drones for light shows is not necessarily a multi-agent problem. In most commercial implementations, drones operate on pre-programmed, centrally coordinated flight paths without needing to communicate or react to one another—this is effectively a single-agent system executed in parallel. However, if drones must respond to local information or coordinate with nearby drones (e.g., to avoid collisions or handle dynamic changes), the problem shifts to a multi-agent one. Therefore, whether it's a multi-agent system depends on the control approach: centralized (single-agent) vs. decentralized (multi-agent).

Search[1]

Q6 For a search problem with average branching factor b, the version of BFS from our class notes explores $\mathcal{O}(b^d)$ nodes. Explain how the number of nodes explored changes when the `if w == goal: terminate` condition block is removed; i.e., when expanding a

[1]The RJM questions in the Search section are inspired by Todd Neller's work presented at EAAI 2010. Original publication: **T. Neller**, "Model AI Assignments", AAAI, vol. 24, no. 3, pp. 1919-1921, Jul. 2010.

node, v, each neighbor w is not checked for whether it is the goal state. (2)

If the `if w == goal: terminate` condition is removed from BFS, the algorithm will not stop immediately upon discovering the goal as a neighbor. Instead, it will continue exploring all nodes at the current depth before checking for goal states during dequeue.

This delays termination by up to one full level, so the number of nodes explored increases from $\mathcal{O}(b^d)$ to $\mathcal{O}(b^{d+1})$, where d is the depth of the shallowest goal. This can significantly increase computation for large b.

Q7 Derive the number of nodes (asymptotic analysis, $\mathcal{O}$-notation) iterative deepening search expands, assuming the depth limit is increased by 1 at every iteration? (2)

In Iterative Deepening Search (IDS), at each depth i (from 0 to d), a depth-limited DFS is performed. The number of nodes at depth i is $\mathcal{O}(b^i)$, and this work is repeated for each iteration up to depth d.

Total nodes expanded:

$$\sum_{i=0}^{d} \mathcal{O}(b^i) = \mathcal{O}(b^d)$$

The deeper levels dominate the sum, so the total asymptotic cost is $\mathcal{O}(b^d)$, same as BFS, but with higher constant overhead due to repeated work.

[LINK] Big-O notation - notes, MITx

In the rest of this section on Search, we will be using Rook Jumping Mazes (RJMs) - a game based on Rook moves in Chess - to understand various search techniques. In an RJM, you are given a square $n \times n$ grid, with each cell containing a number between 1 to $n-1$. From any cell, the player may move either horizontally or vertically, but must take exactly as many steps in the chosen direction as the number on the player's current cell. For instance, in the example on the next page, the agent starts at D1, and the cell has the number 3 on it; therefore the agent can move to either D4 or A1, since they are the only cells that are 3 squares away from D1, and so on. Regardless of the length of the jump (i.e. number of cells passed), this action is considered a single jump (i.e., having a cost of 1) for purposes of our search algorithms. The objective is to get from an assigned start state to a pre-specified goal state.

Consider this example:

	1	2	3	4	5
A	4	G	3	3	2
B	3	3	4	4	4
C	1	1	3	4	4
D	3	3	3	1	2
E	3	1	1	3	3

The cell D1 is the start state and the cell A2 is the goal state. The shortest solution to this RJM is: [D1, A1, A5, C5, C1, C2, D2, A2]. Such a game lends itself well to the various search algorithms we covered in class.

Q8 Can both Breadth-First Search and Depth-First Search be used to solve RJMs? Will both yield the shortest path? Explain your answer. (2)

Yes, both Breadth-First Search (BFS) and Depth-First Search (DFS) can be used to solve Rook Jumping Mazes (RJMs), as they are both complete search algorithms capable of traversing the full state-space.However, only **BFS guarantees the shortest path**, since it explores nodes level-by-level and finds the shallowest solution first when all actions have uniform cost (as is the case in RJMs). **DFS does not guarantee the shortest path**, as it may go deep down an unpromising path before backtracking, possibly finding a longer or suboptimal path first.

Q9 Assuming that each jump has a cost of 1, irrespective of the number of cells the rook passes while making that jump, we can use A* search to find the shortest path. Recall from class that A* search requires a consistent heuristic to guarantee an optimal solution. Show that for this problem, the Manhattan distance from any cell to the goal cell divided by $(n-1)$ is a consistent heuristic, where n is the number of cells in a row/column. In other words, show that

$$H(node, goal, n) = \frac{|node_x - goal_x| + |node_y - goal_y|}{n-1}$$

is a consistent heuristic. Remember that a proof may not rely on a single example, but instead must hold true in general. (5)

[LINK] Here is a good resource on concise and clear proof-writing, which closely mirrors the expectations for most written answers in the course. It is a skill that will serve you

well, especially if you are interested in taking more advanced courses or are considering getting involved in research.

To prove the heuristic

$$H(node, goal, n) = \frac{|node_x - goal_x| + |node_y - goal_y|}{n-1}$$

is consistent, we must show:

$$H(A, G, n) \leq 1 + H(B, G, n)$$

for any node A, its neighbor B, and goal G.

In Rook Jumping Mazes, a move from A to B must be exactly k steps in one direction, where $1 \leq k \leq n-1$. Assume a vertical move: $x_2 = x_1 \pm k$, $y_2 = y_1$.

Then:

$$H(A, G, n) = \frac{|x_1 - x_g| + |y_1 - y_g|}{n-1}$$

$$H(B, G, n) = \frac{|x_1 \pm k - x_g| + |y_1 - y_g|}{n-1}$$

We know:

$$||x_1 - x_g| - k| \leq |x_1 \pm k - x_g| \leq |x_1 - x_g| + k$$

So:

$$H(B, G, n) \geq \frac{|x_1 - x_g| - k + |y_1 - y_g|}{n-1} \Rightarrow H(B, G, n) + \frac{k}{n-1} \geq H(A, G, n)$$

Since $k \leq n-1$, we get:

$$\frac{k}{n-1} \leq 1 \Rightarrow H(B, G, n) + 1 \geq H(A, G, n) \Rightarrow H(A, G, n) \leq 1 + H(B, G, n)$$

Thus, the heuristic is consistent.

Q10 Consider the following RJM:

	1	2	3	4
A	2	3	2	3
B	1	2	1	1
C	3	2	2	3
D	1	1	3	G

On this maze, using the heuristic from Q9, complete the A* search process shown below, starting with Step 2. Terminate your search when the Goal node (cell D4) is popped from the priority queue. Double check your calculations! (8)

Initialization:

- Priority Queue, PQ = {}
- cost = 0
- v = `NULL`
- path = []

Priority for B2 = cost + Heuristic(B2) = 0 + 4/3

Push start state (`v`=B2, `priority`=4/3, `cost`=0, `path`=[]) to PQ

PQ = {(B2, 4/3, 0, [])}

Step 1

Since PQ $\neq \emptyset$, pop by priority (lower is better in this setting, since priority is based on total heuristic cost).

- Priority Queue, PQ = {}
- cost = 0
- v = B2
- path = [B2]

Since B2 is not the goal state, continue.
Neighbors of B2 = {D2, B4}

Priority for D2 = cost + wt(B2, D2) + H(D2) = 0 + 1 + 2/3 = 5/3
Priority for B4 = cost + wt(B2, B4) + H(B4) = 0 + 1 + 2/3 = 5/3

New cost for D2 = cost at B2 + wt(B2, D2) = 0 + 1 = 1
New cost for B4 = cost at B2 + wt(B2, B4) = 0 + 1 = 1

Push (`v`=D2, `priority`=5/3, `cost`=1, `path`=[B2])
Push (`v`=B4, `priority`=5/3, `cost`=1, `path`=[B2])

PQ = {(D2, 5/3, 1, [B2]), (B4, 5/3, 1, [B2])}

Step 2, . . . :

Step 2

Pop from PQ: `v` = D2, `priority` = $\frac{5}{3}$, `cost` = 1, `path` = [B2]
New path: [B2, D2]
D2 is not the goal.
Neighbors of D2 (value = 1): D1, D3

- D1: priority = $1 + 1 + \frac{2}{3} = \frac{8}{3}$, cost = 2
- D3: priority = $1 + 1 + \frac{1}{3} = \frac{7}{3}$, cost = 2

Push to PQ:

- (D3, $\frac{7}{3}$, 2, [B2, D2])
- (D1, $\frac{8}{3}$, 2, [B2, D2])

PQ = {(B4, $\frac{5}{3}$, 1, [B2]), (D3, $\frac{7}{3}$, 2, [B2, D2]), (D1, $\frac{8}{3}$, 2, [B2, D2])}

Step 3

Pop from PQ: `v` = B4, `priority` = $\frac{5}{3}$, `cost` = 1, `path` = [B2]
New path: [B2, B4]
B4 is not the goal.
Neighbors of B4 (value = 1): B3

- B3: priority = $1 + 1 + \frac{2}{3} = \frac{8}{3}$, cost = 2

Push to PQ:

- (B3, $\frac{8}{3}$, 2, [B2, B4])

PQ = {(D3, $\frac{7}{3}$, 2, [B2, D2]), (D1, $\frac{8}{3}$, 2, [B2, D2]), (B3, $\frac{8}{3}$, 2, [B2, B4])}

Step 4

Pop from PQ: `v` = D3, `priority` = $\frac{7}{3}$, `cost` = 2, `path` = [B2, D2]
New path: [B2, D2, D3]
D3 is not the goal.
Neighbors of D3 (value = 3): D1, D4

- D4: priority = $2 + 1 + 0 = 3$, cost = 3

Push to PQ:

- (D4, 3, 3, [B2, D2, D3])

D1 already in PQ with same or better priority; do not update.
PQ = {(D4, 3, 3, [B2, D2, D3]), (D1, $\frac{8}{3}$, 2, [B2, D2]), (B3, $\frac{8}{3}$, 2, [B2, B4])}

Step 5

Pop from PQ: `v` = D4, `priority` = 3, `cost` = 3, `path` = [B2, D2, D3]
Goal node reached.
Final path: [B2, D2, D3, D4]
Total cost: 3
Search terminated.

Local Search

Your professor is a coffee snob, and since he knows a thing or two about AI, he tries to build an espresso machine that will learn a user's preferences over time to pull the perfect shot. He lets the machine control the following values:

- Weight of coffee beans (15-21 grams)
- Grind size (250-400 microns)
- Water Pressure (7-11 bars)
- Brew time (27-33 seconds)

`Sidenote: There was a kickstarter project in 2016 that proposed something similar and failed spectacularly. Link to story.`

Q11 Based on our discussions in class, how you would implement such a program using local search? What would be a reasonable objective function? (This question is somewhat open-ended, so don't worry too much about the 'right' answer. I am really looking for whether your thought process reflects a solid understanding of local search.) (2)

I would represent each state as a combination of the four parameters: coffee weight, grind size, water pressure, and brew time. The local search would start with a random configuration and iteratively adjust parameters to find better ones. A reasonable objective function would be the average user rating of the espresso (e.g., on a 1–10 scale). The goal is to maximize this score. The algorithm would explore neighbors by tweaking parameters slightly and keeping changes that lead to better ratings.

Q12 Explain how the three refinements to local search discussed in class (random restarts, varying step size, and simulated annealing) may be applicable to this problem. (2)

- **Random restarts:** If the machine gets stuck producing only mediocre shots (local maxima), restarting with a random configuration may help discover better tasting regions in the parameter space.
- **Varying step size:** Initially using larger steps enables broader exploration of the configuration space; smaller steps later allow fine-tuning once near a peak of user preference.
- **Simulated annealing:** This allows the system to occasionally accept worse configurations early on to escape local optima, with the likelihood of accepting such configurations decreasing over time.

Q13 An important design choice in the implementation of a simulated-annealing-based coffee machine would be the initial temperature T and the decay constant d. Given the objective function you proposed in Q11, what values of T and d should you choose such that the probability of accepting **any** worse neighboring configuration starts at above 0.9, and decays to less than 0.1 over 30 iterations. Explain how you arrived at your values for T and d. (2)

[HINT: if your objective function in Q11 is designed to be maximized, then you should convert it to an energy function that needs to be minimized in accordance with simulated annealing conventions.]

Let the energy difference $\Delta E = 1$. We want the initial acceptance probability to be above 0.9:

$$e^{-1/T} > 0.9 \Rightarrow T > \frac{1}{-\ln(0.9)} \approx 9.49$$

To ensure the probability drops below 0.1 after 30 iterations:

$$T \cdot d^{30} < \frac{1}{-\ln(0.1)} \approx 0.434$$

Using $T = 10$:

$$10 \cdot d^{30} < 0.434 \Rightarrow d < (0.0434)^{1/30} \approx 0.88$$

So, a good choice is:

$$T = 10, \quad d = 0.88$$

This starts with a high chance of accepting worse moves and cools quickly over time.