Mini Project 4: Planning Under Uncertainty

Instructions

- The assignment is due by April 16, 2026 at midnight.
- There is a Google Colab with helper code located at `https://colab.research.google.com/drive/1MdNKQij9xMr9SsB2uIr1QV6lfYyOGVDN`. Open the notebook, go to "File" in the top left, and **save a copy**.
- Please submit to Gradescope (accessible via Canvas). There will be **three** assignments on Gradescope which you will need to submit to: one for your written answers, one for code upload, and one for video submissions.
- Your written answers should be in a single PDF file, ideally produced with LaTeX. Your code should also be in a single PDF. To save the Colab to PDF, go to "File" in the top left, click "Print" and then save as PDF.
- Some questions ask you to submit animations as **.mp4** videos (not .mov). To generate a video from the Colab, use:

  ```
  from matplotlib.animation import FFMpegWriter
  anim.save("filename.mp4", writer=FFMpegWriter(fps=5))
  ```

  Then download the file from Colab (click the folder icon in the left panel, find the file, right-click, and select "Download"). Upload the `.mp4` files to the **video submission** assignment on Gradescope. Please use the exact filenames specified in each question.
- The goal here is learning. You can work with other students, but be sure that, in the end, the solutions you submit were your work and that you understand them completely.
- You may use LLMs to help with this assignment; note, though, that we've tried these questions in several LLMs and they often produce incorrect answers, so the correctness is up to you!

Setup

We are going to continue the theme from earlier mini-projects, but now we consider a *stochastic* version of the pickup and rescue problem in the form of an MDP. You will develop approximate online-planning solutions using heuristic search and Monte-Carlo tree search.

The pickup and rescue problem

We consider a gridworld domain consisting of a robot, a patient, and a target hospital.

The robot's goal is to pick up the patient and deliver them to the hospital. The blue circle represents the robot, the orange circle represents the patient, and the red cross represents the hospital. The environment has **one-way roadblocks**, indicated by a flat triangle between two cells (the robot can only cross in one direction), and **two-way roadblocks**, indicated by thick black boundaries between two cells (the robot cannot cross in either direction).

The evolution of fire

Our grid now has fire. The fire starts at some initial locations and evolves through time. The fire grid at time t is independent of the robot and patient, and depends only on the fire grid at the previous time step. Fire completely ignores walls, roadblocks, and the hospital.

Given the fire grid at time t, the probabilities of fire at any two different cells at time $t+1$ are independent:

$$P(\mathbf{F}^{t+1} \mid \mathbf{F}^t) = \prod_{(i,j)\in\texttt{grid}} P\Big(\mathbf{F}^{t+1}_{(i,j)} \mid F^t_{(i',j')\in\texttt{neighbors}((i,j))}\Big)\,,$$

where $\texttt{neighbors}((i,j)) = \{(i',j') \mid |i-i'| \le 1 \wedge |j-j'| \le 1\}$ is the 3×3 patch of cells centered at (i,j), including (i,j) itself.

The probability of fire in cell (i,j) at time $t+1$ is the weighted probability of its neighboring cells being on fire at time t, for a given fixed weight matrix $W \in \mathbb{R}^{3\times 3}$:

$$P\Big(\mathbf{F}^{t+1}_{(i,j)} \mid F^t_{(i',j')\in\texttt{neighbors}((i,j))}\Big) = \sum_{i',j'} W[i'-i+1,\ j'-j+1]\cdot F[i',j'],$$

where the weight matrix W is normalized so that its entries sum to one.

You might recognize that given the fire grid at time t (a matrix of 0–1 values), the probability of fire at time $t+1$ is the 2D convolution of the fire grid with the weights W.

Here is another way to understand the fire process:

- We start with some initial fire grid.
- At each time step t, for each cell, we randomly select a neighbor (including the current cell) with probability proportional to the weight matrix; the current cell at time $t+1$ gets the selected neighbor's fire value at time t. Neighbors outside the grid do not have fire.

Our approach

We adopt an **online-planning** approach, where at every step our agent:

1. plans according to the current state,
2. executes an action, and
3. observes a new state of the fire grid and replans (restarts from step 1).

Please take a moment to look at the code at the top of the Colab notebook. The `PickupProblem` class represents the deterministic pickup-and-rescue problem (without fire). The `FireProcess` class represents the stochastic fire process. The `FireMDP` class combines these into the full MDP.

1 Determinized planning

In our first approach, at each planning step we turn the MDP into a **deterministic min-cost path problem**:

- The state space no longer includes the fire grid; it only contains the state of the pickup-and-rescue problem (robot location, patient location, and whether the robot is carrying the patient).
- For a transition (s_t, a, s_{t+1}) at time t, we charge a cost of

  $$c - \log(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1})))\,,$$

  where c is a small constant cost for each step, and $P(\texttt{on\_fire}_t(s))$ is the marginal probability of the robot's destination cell being on fire at time t.
- We find the least-cost path to reach the patient and deliver them to the hospital—this path becomes our open-loop plan.

Once we have a min-cost path problem, we use A* search with a heuristic that ignores fire: the sum of the Manhattan distance from the robot to the patient plus the Manhattan distance from the patient to the hospital, scaled by the step cost c. When the robot is carrying the patient, the robot-to-patient distance is zero.

Complete the implementation of `DeterminizedFireMDP` in the Colab. In particular:

- Complete `fire_dist_at_time` to compute the marginal probability of each cell being on fire at time t, given the true fire state at time 0.
- Using `fire_dist_at_time`, complete `step_cost`.
- Complete the heuristic function `h` based on the description above.

Then complete `FireMDPDeterminizedAStarAgent` by implementing the `determinized_problem` method. You may build on your A* search implementation from HW1.

Hints:

- Before you code, try to derive the marginal probabilities of each grid cell being on fire at time t as an expression of the marginal probabilities at time $t-1$. More concretely, start small: consider a grid consisting of only two cells, X and Y, and assume W is uniform. Write $P(X_t = 1 \mid X_0, Y_0)$ as an expression of $P(X_{t-1} = 1 \mid X_0, Y_0)$ and $P(Y_{t-1} = 1 \mid X_0, Y_0)$.
- We formulate the A* state as (`robot_loc`, `patient_loc`, `time`) and use time as an index into a precomputed sequence of marginal fire probability grids. See `DeterminizedFireMDPState` and `DeterminizedFireMDP` in the Colab for details.

2 Questions

2.1 Formulation and experiments

1. i. Why aren't we approaching this problem with dynamic programming? What is the size of the state space?

> **Solution:** Exact dynamic programming is impractical because the full MDP state must include both the pickup-and-rescue state and the complete fire configuration. If the grid has N cells, then the fire grid alone has 2^N possible configurations.
>
> The pickup-and-rescue component includes the robot location, the patient location, and whether the robot is carrying the patient. This is on the order of $N^2 \cdot 2$ states, ignoring impossible states. Thus the full state space is on the order of
>
> $$O(N^2 2^N),$$
>
> or more explicitly roughly
>
> $$|\mathcal{S}| \approx N \cdot N \cdot 2 \cdot 2^N.$$
>
> This exponential dependence on the number of grid cells makes exact dynamic programming infeasible for even moderately sized grids.

ii. Derive an expression for the marginal probability of a cell being on fire at time t as a function of the marginals at time $t-1$. What mathematical operation does this correspond to? What happens to the marginal fire probabilities as $t \to \infty$ for any initial fire configuration, and why?

> **Solution:** Because fire at each cell at time $t+1$ depends independently on its 3×3 neighborhood at time t, the marginal probability of cell (i, j) being on fire at time t is:
>
> $$P(F^t_{(i,j)} = 1) = \sum_{i',j'} W[i' - i + 1,\ j' - j + 1] \cdot P(F^{t-1}_{(i',j')} = 1).$$

This is exactly a **2D convolution** of the marginal fire grid at time $t-1$ with the weight matrix W (with zero-padding at the boundaries).

As $t \to \infty$, the marginal fire probabilities converge to 0 for any initial configuration. Intuitively, the convolution with zero-padded boundaries "leaks" probability mass out of the grid at each step: boundary cells mix with zero-valued out-of-grid neighbors, reducing the total fire probability. Formally, the operation is a linear map whose spectral radius is strictly less than 1 under zero-padding, so repeated application drives the marginals to zero.

iii. The step cost for a transition at time t is $c - \log(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1})))$. Show that minimizing the total path cost is related to maximizing the probability that the robot survives (does not step on fire) along the entire path. Why do we include the constant c?

Solution: The total path cost for a path of length n is:

$$\sum_{t=0}^{n-1}\left[c - \log(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1})))\right] = nc - \sum_{t=0}^{n-1}\log(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1}))) = nc - \log\prod_{t=0}^{n-1}(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1})))$$

The product $\prod_{t=0}^{n-1}(1 - P(\texttt{on\_fire}_{t+1}(s_{t+1})))$ is exactly the probability that the robot survives (does not step on fire) at *every* step along the path, under the independence approximation. Minimizing the total cost is therefore equivalent to maximizing this survival probability (since $-\log$ is monotone decreasing).

The constant c serves two purposes: (1) it ensures all edge costs are strictly positive even when fire probability is zero, which is required for A* correctness; (2) it penalizes longer paths, so among equally safe paths the agent prefers shorter ones.

iv. Run your determinized agent in the MDP given by `get_problem("just_wait")` several times. Generate an animation of a **successful run** (the robot successfully rescues the patient) in which the fire does **not completely die out** when the robot moves down from the top row. Submit this animation as `just_wait.mp4`. If you cannot produce this animation after roughly 10 attempts, your implementation may be buggy. Describe and explain the agent's "waiting" behavior, connecting it to how the marginal fire probabilities evolve over time.

Solution: The agent "waits" at the top of the grid for several time steps before moving down toward the patient. This behavior arises because the marginal fire probabilities in the cells along the path decrease over time: the 2D convolution with zero-padded boundaries causes the fire to diffuse and leak out of the grid. By waiting, the agent reduces the expected cost (fire risk) of traversing those cells.

At each replanning step, the agent re-observes the actual fire grid and recomputes the marginals from the current state. The determinized cost function sees that future cells along the path will have lower marginal fire probabilities if the agent waits, so the optimal A* plan says "stay put for now, move later." Once the marginals have decayed sufficiently, the agent commits to moving down and completing the rescue.

2.2 Lessons

2. Run the determinized agent in the MDP given by `get_problem("the_choice")`. In this environment, the agent faces a choice from its initial location:

   - **Go down**: take a shortcut through a one-way passage, but risk getting close to fire with no room to maneuver.
   - **Go right**: enter a larger room with more fire, but with space to move around and avoid it.

Generate an animation of a successful run and submit it as `the_choice_determinized.mp4`.

i. What choice does your determinized agent make? The determinized approximation replaces a stochastic fire process with marginal fire probabilities. What information about the fire process is lost in this approximation, and how does that explain the agent's choice?

> **Solution:** The determinized agent may go **either direction**—down the narrow left corridor or right into the larger room and down the right corridor. The choice depends on the cumulative marginal fire costs along each path. Going down the left corridor is shorter but column 0 is close to the fire cluster in columns 2–3 (and fire ignores walls). Going right, column 5 is farther from that cluster but the path is longer. Both choices are plausible depending on how the marginals balance path length against fire proximity.
>
> Regardless of which path the agent takes, the key information lost is the **correlation structure** between cells. The determinized approximation replaces the *joint* fire distribution with independent *marginal* probabilities at each cell. In the stochastic fire process, nearby fire cells are highly correlated—fire tends to be "clumped" in a contiguous region, so there are often safe gaps to navigate through. The marginal approximation cannot see these gaps; it only sees that each cell has some nonzero fire probability. The agent's path choice is driven entirely by which corridor has lower cumulative marginal cost, without accounting for the fact that correlated fire might make a seemingly-riskier path actually safer.

ii. The determinized planner produces an open-loop plan but replans after each step. Give a concrete scenario in which this approach can be arbitrarily bad compared to a closed-loop policy. Characterize the types of environments where you would expect the determinized approach to perform well versus poorly.

> **Solution: Concrete scenario**: Consider a long narrow corridor of length n with fire that blocks one end with probability 0.5 and the other end with probability 0.5 (but never both simultaneously). The marginals show 0.5 fire probability at both ends, making both directions look equally costly. The determinized planner might commit to one direction; if fire appears there after a few steps, the agent is trapped in the narrow corridor with no room to turn around. A closed-loop policy would observe which end is actually blocked and go the other way immediately. By making n large, the cost of the wrong commitment grows without bound.
>
> **Works well**: Environments where fire spreads slowly or predictably, open grids with many alternative paths, and situations where marginals are a reasonable proxy for the true distribution (e.g., nearly independent fire cells).
>
> **Works poorly**: Environments with narrow passages, highly correlated fire dynamics, or situations where small stochastic deviations can "trap" the agent (high-stakes binary outcomes).

iii. If planning time were very expensive compared to execution, would you still replan at every step? Describe a replanning strategy that reduces the number of replanning calls while still maintaining good performance. What are the trade-offs?

> **Solution:** No—if planning is expensive, replanning at every step is wasteful when the environment matches predictions.
>
> A practical strategy: **replan only when the observed state deviates significantly from the plan's assumptions.** Concretely:
>
> - Execute the current open-loop plan step by step.
> - After each step, compare the observed fire grid to the predicted marginal fire distribution. If the deviation exceeds a threshold (e.g., the plan's next cell is now on fire, or the total ℓ_1 difference between observed and predicted fire grids exceeds some ϵ), trigger a replan.
> - Also replan if the remaining plan is empty or has become infeasible.

Trade-offs: Less frequent replanning saves computation but risks following a stale plan when fire evolves unexpectedly. The threshold ϵ controls this trade-off: small ϵ replans often (safe but expensive), large ϵ replans rarely (cheap but risky). An intermediate approach is to replan every k steps for some fixed k.

iv. An alternative strategy would have been to use the most-likely-outcome determinization. Describe a concrete situation in which its behavior would be different from the method we've used here.

Solution: The method used here plans with marginal fire probabilities, so a cell with fire probability 0.4 still contributes a nonzero risk cost. In a most-likely-outcome determinization, that same cell would be treated as not on fire, because its most likely state is fire-free.

For example, suppose there is a short corridor where every cell has fire probability 0.4, and a longer detour where every cell has fire probability near zero. The marginal-cost method may prefer the longer detour because the short corridor accumulates substantial expected fire risk. The most-likely-outcome determinization would treat the short corridor as completely safe and would likely choose it.

Conversely, if a cell has fire probability 0.51, the most-likely determinization treats it as definitely on fire, while the marginal method treats it as risky but not impossible. Thus the most-likely method can behave more abruptly, ignoring moderate risks below 0.5 and overcommitting to obstacles above 0.5.

3 Monte-Carlo Tree Search

Now that we have seen a failure mode of our determinized planning agent, let's try to do better with closed-loop planning with MCTS!

With MCTS, we have more of a chance of hedging bets, so we might be inclined to go in directions where there are more options in case we get caught, even if the expected open-loop cost is higher.

We have provided you with an MCTS implementation, `run_mcts_search`, that works on both MDPs and POMDPs. Please take a look at the documentation of `run_mcts_search` to understand how to use it on MDPs, then implement an MCTS agent for MDPs.

3. Complete the implementation of `MCTSAgent`. Hint: You can pass in the `self.planning_horizon` to `run_mcts_search`, to handle both infinite-horizon problems (by receding-horizon planning) and finite-horizon problems.

   Similarly to in the previous problems, visualize the behavior of the MCTS planning agent in `the_choice` MDP and generate an animation of a successful run and submit it as `the_choice_mcts.mp4`.

   i. Compare the behavior of your `MCTSAgent` to the `FireMDPDeterminizedAStarAgent` on `the_choice`. Do the two agents behave differently in avoiding the narrow one-way passage? Briefly discuss why Monte Carlo rollouts might, in principle, capture the "risk" of being trapped differently from the "average fire probability" used in the determinized A* approach.

   Solution: We do not observe a consistent difference between the MCTS agent and the determinized A* agent. Even after running both algorithms multiple times, they behave similarly and do not differ in how often they avoid the narrow one-way passage. (We accept all answers to this questions becuase there is change to the question after release.)

   In principle, Monte Carlo rollouts can better capture the risk of being trapped because they simulate specific realizations of the environment. In some rollouts, the fire may completely block the passage, leading to a terminal failure with no escape, which explicitly reveals the high-risk nature of that path. In contrast, the determinized A* agent uses an average fire probability,

which smooths over these outcomes and represents the passage as moderately risky but still traversable. However, in this particular environment, this difference is not pronounced enough to produce a noticeable behavioral difference between the two agents.

ii. The performance of MCTS is highly dependent on the number of rollouts. If you significantly limit the computation time (e.g., only 10 rollouts per step), MCTS often performs worse than A*. Why does A* handle low-computation budgets more gracefully than MCTS in this specific gridworld?

Solution: A* uses a heuristic (h) to provide a global "sense of direction" even with no search. MCTS with very few rollouts has high variance and very little "vision." If it hasn't sampled a successful path to the hospital yet, its moves are essentially random. A* at least moves toward the goal because the Manhattan distance heuristic is always available.

iii. If the robot is in a "safe" area but the random rollout policy makes it walk into fire, how does this affect the value estimate of that state?

Solution: If a random rollout policy causes the robot to walk into fire from an otherwise safe state, it leads to a significant underestimation of that state's true value.