CS747 Assignment 2

Geet Sethi (23B2258)

October 12, 2025

1 Task 1: MDP Solver (`planner.py`)

1.1 Problem Representation

We parse standard MDP files with:

- **States** indexed as $\{0, \ldots, S-1\}$,
- **Actions** indexed as $\{0, \ldots, A-1\}$,
- **Terminal set** given by an `end` line (can be empty),
- **Transitions** lines: `transition s a s' r t`,
- **Type** `mdptype` which can be either `continuous` or `episodic`
- **Discount** γ which is a float between 0 and 1

Rewards and transitions are stored as dense $S \times A \times S$ arrays for simplicity.

1.2 Action-Value Computation

For any value function V, we compute

$$Q(s,a) \;=\; \sum_{s'} P(s' \mid s,a)\,\Big(R(s,a,s') + \gamma V(s')\Big).$$

This is used in both policy improvement and in extracting a greedy policy from the LP solution.

1.3 Policy Evaluation

Given a deterministic policy π, we solve the linear system

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')\big(R(s,\pi(s),s') + \gamma V^{\pi}(s')\big) \forall s$$

with absorbing terminal states clamped to $V(s) = 0$.

$$\begin{aligned} A_s &\leftarrow I - \gamma T(s,\pi(s),:) \\ b_s &\leftarrow \sum_{s'} T(s,\pi(s),s') \times R(s,\pi(s),s') \\ V_{\pi} &\leftarrow A^{-1} b \end{aligned}$$

where A_s is the sth row of A and b_s is the sth element of b.

1.4 Howard's Policy Iteration (HPI)

We implement standard policy iteration:

1. Initialize π arbitrarily (all zeros).
2. **Policy evaluation**: solve V_π exactly as above.
3. **Policy improvement**: for each non-terminal s, $\pi(s) \leftarrow \arg\max_a Q(s,a)$.
4. Repeat until no improvement exceeds a small ε (used for numerical tie-breaking).

Note that we set the policy to 0 (fixed) for terminal states.

1.5 Linear Programming (LP) Formulation

We minimize $\sum_s V(s)$ subject to Bellman optimality inequalities:

$$V(s) \;\geq\; \sum_{s'} T(s,a,s')\Big(R(s,a,s') + \gamma V(s')\Big) \quad \forall s, a,$$

and enforce $V(s) = 0$ for terminal s. The optimal $\{V(s)\}$ are then used to extract a greedy policy via $\arg\max_a Q(s,a)$.

We use CBC via PuLP and check for optimal termination.

1.6 Observations

- **HPI vs LP**: HPI is typically faster for small to medium problems while LP is robust and convenient for correctness checks especially since HPI will not be able to tell us if the MDP is solvable or not whereas LP will.
- **Policy evaluation**: The policy evaluation can be significantly sped up by using efficient matrix operations in numpy instead of loops.

2 Task 2: Game Encoding (`encoder.py`)

2.1 Game and MDP Design

Game: We are given a deck of 13 hearts and 13 diamonds (i.e. 2 copies of each rank 1-13) and a *threshold* Θ which defines the bust condition that if the hand sum reaches or exceeds Θ, the episode ends with zero reward.

Actions:

1. **Add** (action 0): draw a random card uniformly from the remaining deck.
2. **Swap** (actions 1-13): give one card of a chosen rank from hand, then draw uniformly from the deck.
3. **Stop** (action 14): terminate; receive current hand sum plus a possible bonus.

Note that we merge the duplicate swap actions for the same rank into a single action since our MDP is symmetric and we can output the appropriate actions based on the hand in the `decoder.py`.

States: Tuples $x \in \{0,1,2\}^{13}$, where x_i is the count of rank i in hand and a single absorbing terminal state together make up the state space. We enumerate all reachable states from the empty hand up to the deck constraint and Θ.

Rewards:

- **Stop**: $R = \sum_i i \cdot x_i + \text{bonus}(x)$.
- **Bust (Add/Swap)**: transition to terminal state with $R = 0$.
- **Non-terminal Add/Swap**: transition to next state with $R = 0$.

Bonus: If the hand contains at least one card from each of a configured rank triple (a, b, c), we add a fixed bonus B on *Stop*.

Termination and discount: We output `mdptype episodic` and $\gamma = 1.0$ always since episodes terminate on `Stop`, bust, or deck exhaustion.

2.2 Transition Model

Deck accounting: Let $N(x) = 26 - \sum_i x_i$ be remaining cards.

- If no cards remain, **Stop** is the only action and we go to terminal with $R = \sum_i i \cdot x_i + \text{bonus}(x)$.
- For **Add**, the probability of drawing rank j is $\frac{2-x_j}{N(x)}$ if $2 - x_j > 0$. The next state is $x + e_j$ if it does not bust (otherwise we jump to terminal with the corresponding probability).
- For **Swap(k)**, the probability of drawing rank j is $\frac{2-x_j}{N(x)}$ if $2 - x_j > 0$ and $x_k > 0$. The next state is $x - e_k + e_j$ if it does not bust (otherwise we jump to terminal with the corresponding probability). Also if $2 - x_j = 0$, we go to terminal with probability 1 and reward 0.

2.3 Strategies and Insights from the Generated Actions:

- **Always Add when Safe**: The optimal policy seems to add while there is zero bust probability. This shortens effective horizons and helps reach the threshold quicker.
- **Conservatism near threshold**: The model sends any bust probability mass to terminal with $R = 0$, so near-threshold states often prefer `Stop` unless the expected gain from Add/Swap outweighs the bust risk and/or helps secure the bonus.