CS747 : Detailed Explanation of codes on MDP Solvers using Policy Iteration and Linear Programming

1 Introduction

This report summarizes the functionality of the various files contained in the CS747 assignment submission, which implements solutions to Markov Decision Problems (MDPs) using Howard's Policy Iteration and Linear Programming techniques.

2 Code Files Overview

2.1 `autograder.py`

This script automates the grading and validation process. It runs test cases on implemented algorithms and compares the generated output (policy and value function) against ground truth data for correctness.

2.2 `create..venv.sh`

A shell script designed to set up a virtual environment. It installs required Python dependencies listed in `requirements.txt`, making the codebase portable and easy to set up.

2.3 `decoder.py`

Responsible for translating the computed numeric policy or value function into a human-readable or gridworld-compatible format. It ensures output matches the expected specification for evaluation.

2.4 `encoder.py`

Encodes gridworld or MDP environment data into a structured MDP format. This includes defining state spaces, actions, rewards, and transition dynamics, often read from .txt files.

2.5 `generateMDP.py`

Utility script to generate synthetic MDP problems. It allows the user to create MDPs with configurable parameters like number of states, actions, and stochastic transitions, suitable for debugging or testing solvers.

2.6 `gridworld.py`

Implements the gridworld environment. Defines cells, obstacles, rewards, and terminal states. Also provides functions to translate this into MDP-compatible structures.

2.7 `imageGEN.py`

Handles visualization of MDP policies and value functions using `matplotlib`. Useful for graphically representing the policy (e.g., arrows) overlaid on the gridworld.

2.8 `planner.py`

Core module of the assignment. Implements two primary algorithms for solving MDPs:

- **Howard's Policy Iteration (HPI)**: Iteratively evaluates and improves a deterministic policy until it converges to the optimal.
- **Linear Programming (LP)**: Uses the PuLP library to formulate and solve the Bellman optimality conditions as an LP problem.

The script is designed to be invoked from the command line with arguments such as `--algorithm hpi` or `--algorithm lp` and an input MDP file path.

2.9 `requirements.txt`

Lists all Python packages required to run the code, such as `pulp`, `matplotlib`, etc.

2.10 `report.pdf`

Contain student-written summaries, experimental results, and observations related to the MDP solution performance.

.

3 Data Files

- `data/gridworld/`: Contains 10 gridworld configurations used as MDP input.
- `data/mdp/`: Includes both manually and randomly generated MDPs in episodic and continuing formats.
- `data/test/`: Contains input-output pairs for evaluating correctness of gridworld-based solutions.

4 Conclusion

The submission presents a modular and complete framework to solve MDPs using both Howard's Policy Iteration and Linear Programming. It includes visualization tools, test data, and environment setup scripts, making it comprehensive and easy to use for both development and evaluation.