Toronto Metropolian University

Data Science and Analytics program

DS8010 Interactive Learning in Decision Process

Course project

Empirical Analysis of Built-In POMDPs.jl Algorithms: A Comparative Study

Create by

Alice Yang

501149113

April 2023

Contents

1 Introduction

Markov Decision Processe (MDP) is a well-known method for modelling decision-making processes in a perfect environment under certainty where the outcome depends on the current state of the system and the action taken by the decision maker. All states are given to the agent in MDP problem. Unlike MDP, partially Observable Markov Decision Processes (POMDPs) is a mathematical framework for modelling an agent decision process under an uncertain environment, where the underlying states are not directly observed. POMDPs problems simulate a real-world environment where directly observing the current state is not possible. For instance, the demand distribution of merchandise is unknown to the seller of a store in real life [4]. POMDPs is a popular tool for modelling decision-making problems in a wide range of applications such as aircraft [18], finance [3], and health care [17].

Solving POMDPs problems can be computationally challenging and time consuming due to the number of state and action spaces involved in the problem set. We choose Julia programming language for our study. Julia is a high-level, general-purpose dynamic programming language which is designed for parallelism. Because of this design, Julia is renowned for its fast execution time, comparable to languages like C or C++ while offering a high-level approach comparable to Python [11].

Several algorithms have been developed for solving POMDPs, each with its own strengths and weaknesses. In this study, we aim to perform an empirical analysis of multiple algorithms using the package POMDPs.jl [10] written in Julia and examine the performance of each algorithm. POMDPs.jl provides functionality for defining models, simulating problems, and solving them with built-in algorithms.

1.1 Objetivos

The objectives of this study are to implement several algorithms using the POMDPs.jl package, compare the performance of these algorithms on several POMDPs problems, and generate a performance table comparing the algorithms on different metrics, such as computation time and collected rewards. The results will be analyzed to identify each algorithm's strengths and weaknesses and determine which algorithm performs best on which type of problem. We also expect to identify areas for future research and improvement.

The contributions of this study can be summarized as follows:

- Eight POMDPs instances are involved in this study. The complexity of each instance differs from each other and provides diverse test sets for our empirical analysis.
- Seven algorithms are introduced to solve the POMDPs problems. We provide a comparative analysis of their performance on a set of benchmark problems described above.
- A simulation mechanism is designed to examine the performance of each algorithm using build-in functions in POMDPs.jl.

2 Literature Reivew

This section provides an overview of POMDPs and reviews of recent studies on this topic.

In a standard POMDPs problem, a system can be represented using six components: state space, observation space, action space, transition function, observation function, and reward function. The first three components are the sets of all possible states the system can be in, all possible observations can be made, and actions can be taken by the agent. The transition function is a probability distribution that describes how the system evolves from one state to another in response to an action. Similarly, the observation function is a probability distribution that describes the likelihood of observing a certain observation given the current state of the system. Lastly, the reward function maps states, actions, and observations to a scalar reward value that indicates how desirable a certain state-action-observation combination is.

2.1 Recent works

Although the agent does not have full access to the underlying states in POMDPs setting, we may calculate the most likely states instead and take appropriate action accordingly. A short-term reward value will be received immediately after making an action. The ultimate goal of the POMDPs problem is to learn a policy that maximizes the cumulative long-term reward over time. To achieve this goal, the agent must balance the trade-off between immediate rewards and long-term gains. In some cases, maximizing the long-term cumulative reward may require the agent to disregard the short-term effects. Some of the most popular algorithms used to solve POMDPs problems are summarized in Table 1. We categorize these algorithms as exact solution methods and approximate solution methods.

Table 1: Summary of the relevant literature on POMDPs

Research	**Exact**	**Approximation**	**Algorithms**	**Problem Instance**
Cassandra et al.(1997) [1]	Yes	No	IP	Simple grid world, paint
Litterman et al. (1995) [9]	No	Yes	QMDP	Mini hallway, tiger, grid world, painting
Hauskrecht (2000) [6]	Yes	No	FIB	-
Pineau et al. (2003) [12]	No	Yes	PBVI	-
Kurniawati et al. (2008)[7]	NO	Yes	SARSOP	Rock sample
Silver and Veness (2010) [14]	No	Yes	POMCP	Rock sample
Sunberg and Kochenderfer (2010) [16]	No	Yes	POMCPOW	-

Oftentimes, the exact solution method performs well when there is a small set of states and observations. Finding an exact solution to a POMDPs problem is generally intractable, especially for large and complex problems. The reason is that most exact algorithms use a form of dynamic programming approach, wherein a value function represented by a piecewise linear and convex function is transferred into another. The size of the belief state space grows exponentially with the number of state variables and the horizon of the problem, making it difficult to compute exact solutions. The exact solution method is used in algorithms such as the value iteration algorithm [13], policy iteration algorithm [15], witness algorithm [8], and the IP algorithm [1] that we will examine in our study.

In the contrast, the approximate technique uses randomized simulations to approximate the value function and policy. The approximation methods can handle larger POMDPs problems and scale better to high-dimensional state and observation spaces, but they may not guarantee convergence to the optimal solution. Popular algorithms that used approximation methods are listed in Table 1. The choice of approximation technique depends on the problem at hand and the available computational resources.

2.2 Solve POMDPs problems in other languages

POMDPs have become an increasingly popular research area in recent years, leading to the development of various solvers using different programming languages other than Julia the one we choose. First, the pomdp_py package [19], written in Python and Cython, is an effective frame for solving POMDPs problems. In addition, the R package pomdp [5] offers algorithms such as incremental pruning and enumeration for solving POMDPs problems. Moreover, the C-written Pomdp-solve [2] package provides a broad range of algorithms, including enumeration, two-pass, witness, incremental pruning, and PBVI, to effectively solve POMDPs problems. These different language packages allow researchers and practitioners to apply a variety of tools to solve complex problems. In our study, we will focus on POMDPs.jl as Julia language because Julia enables fast execution times that are comparable to languages such as C or C++ and provides a more intuitive and user-friendly experience like python.

3 Methodology

The purpose of this study is to compare the performance of POMDPs.jl algorithms on different POMDPs models. This research aims to answer the research question: which algorithm performs best in terms of computation time and collected rewards. The study will implement seven built-in algorithms: IP, SARSOP, FIB, PBVI, POMCP, POMCPOW, and QMDP. The algorithms were chosen for their popularity and usefulness. The implementation will be in Julia programming language.

Our evaluation of each algorithm will depend on two metrics: computation time and collected rewards. Average computation time measures the time required by an algorithm to compute the optimal policy, while average collected rewards measure the expected reward that an algorithm achieves during the simulation of each problem. We will also report the standard deviation to provide a measure of variability in the results.

To assess the effectiveness of each algorithm, we will utilize a set of seven POMDP models. These models encompass the tiger problem, crying baby problem, paint problem, query problem, mini hallway problem, rock problem, simple grid world problem, and T-maze problem. The selection of these models was based on their differing levels of intricacy. A brief description of the problems follows:

- Tiger problem: A decision-making problem in which an agent must choose between opening one of two doors, one of which leads to a reward and the other to a penalty for opening the door with a tiger behind.
- Crying baby problem: The agent needs to determine whether to feed a crying baby, which could be a true signal of hungry or a false alarm that wastes the agent's time.
- Paint problem: A problem in which the factory aim to decide whether to ship each item or reject it based on the item's condition.
- Query problem: A problem involves two servers with different speeds and failure probability. The agent is trying to minimize the time takes to process the queries.
- Mini Hallway problem: A navigation problem in which an agent must move through a narrow hallway to a target position with minimal steps.

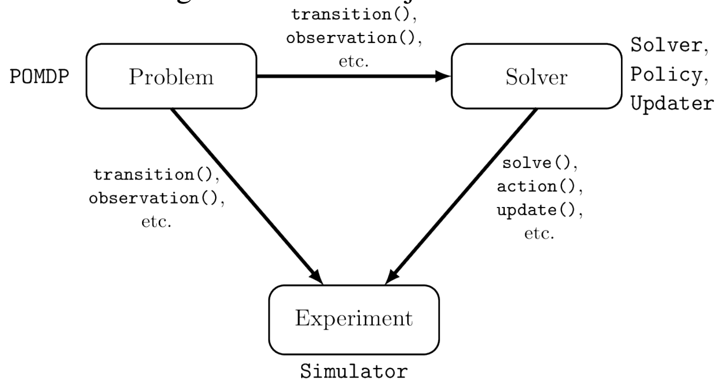
- Rock sample problem: The problem of the rock sample involves making decisions where an agent is required to travel through a grid to retrieve valuable mineral samples, all while avoiding mines that could cause harm to the agent.
- Simple grid problem: A simple problem in which an agent must navigate a grid and collect rewards while avoiding obstacles and penalties.
- T-maze problem: the agent must navigate a T-shaped maze and choose the path that leads to a reward while avoiding the path that leads to a penalty.

3.1 POMDPs.jl architecture

POMDPs.jl developed three main components outlined in Figure 1: the problem, solver, and experiment [20]. The framework of this package defined a set of abstract types and functions that each component must adhere to, for example, transition(). These standardized behaviours allow for seamless integration between the different components of the framework, making it easy to develop, test, and analyze POMDPs models.

We can choose to either use the built-in problem or define our own problem and simulation steps using POMDPs.jl components. Firstly, the problem component is responsible for defining the POMDPs model, including all six components of a POMDPs problem: state space, observation space, action space, transition function, observation function, and reward function. Second, the solver component is responsible for implementing the algorithms that solve the POMDP model. Lastly, the experiment component is responsible for running simulations of the POMDP model and analyzing the results.

Figure 1: POMDPs.jl architecture


transition(), observation(), etc.
POMDP
Problem
Solver
Solver, Policy, Updater
transition(), observation(), etc.
solve(), action(), update(), etc.
Experiment
Simulator

3.2 Incremental Pruning (IP)

The Incremental Pruning algorithm [1] is a dynamic-programming update method for solving POMDPs by iteratively refining an initial belief-dependent value function estimate using a Bellman-like backup operation. In this method, a set of vectors is gradually pruned down to only those that have non-empty witness regions, which helps to reduce the computational complexity of solving the POMDPs. This method has been found to be more efficient than other exact solution method algorithms for solving POMDPs, including the witness algorithm [8].

3.3 QMDP

The QMDP algorithm [9] works by first computing an optimal policy for a fully observable MDP that is equivalent to the POMDP. In this way, it is approximating the POMDPs problem as an MDP problem and derives the upper bound of the q-function. Then, the QMDP algorithm uses the optimal policy to select the best action to take based on the current belief state. This algorithm is an efficient method as it avoids the need to compute and store a belief state representation of the problem.

3.4 Fast Informed Bound (FIB)

The FIB algorithm [6] was introduced in 2000 and involves iterating between two steps: find all values through the value iteration method [13] and derive the upper bound by using the previous. This is similar to the QMDP method but instead of using the current state to find optimal action, optimal action depends on the previous state.

As an exact solution method, FIB algorithm computationally expensive and is closely related to the standard value iteration [13] and policy iteration algorithms[15]. Other POMDP algorithms, such as PBVI described below use different heuristics to improve the scalability and efficiency.

3.5 Point-Based Value Iteration (PBVI)

PBVI [12] is a point-based value iteration algorithm that approximates the value function of POMDPs by maintaining a finite set of representative belief points. At each iteration, the algorithm selects a subset of the belief points and computes the value function for those points exactly, while approximating the value for the remaining belief points using the previously computed values. The algorithm terminates when the difference between consecutive iterations falls below a certain small threshold.

PBVI is efficient and scalable, making it suitable for solving large-scale POMDPs problems. The algorithm has been extended to handle different types of POMDPs, such as continuous state and action spaces, non-linear dynamics, and non-linear observation models.

3.6 Successive Approximations of the Reachable Space under Optimal Policies(SARSOP)

SARSOP works by approximating the POMDPs problem using a procedure similar to other point-based algorithms [7], which samples a set of points from the belief states, reduces the size of the belief space and then optimizes the policy. It generates an approximate representation of the space through symbolic approximation which involves clustering the state and observation space of the POMDPs problem to a smaller size. Then, the sampled points and alpha-vectors generated through the approximation are pruned to further improve efficiency.

It is important to note that SARSOP is not guaranteed to find the exact optimal solution, and its performance can depend on the specific problem instance and the choice of parameters.

3.7 Partially Observable Monte Carlo Planning (POMCP)

POMCP is a Monte Carlo Tree Search algorithm that solves POMDPs problems by constructing a tree of possible actions and observations using random simulations [14]. It estimates the value of each action given the current state of the environment and the agent's current belief.

3.8 POMCP with observation widening (POMCPOW)

Unlike POMCP algorithm described above that combines random simulation with tree search, POMCP with observation widening [16] combines Monte Carlo simulations and tree search to approximate the belief state and find an optimal policy. POMCPOW algorithm improves the efficiency of Monte Carlo simulations by using the online method. This generates simulation trajectories that are biased towards regions of the state space that are likely to lead to high-quality solutions.

POMCPOW algorithm also uses a modified form of Upper Confidence Bounds (UCB) to balance exploration and exploitation during the tree search. This takes into account the history of the search, including the actions and observations that have been made, to guide the exploration more effectively.

References

[1] Anthony Cassandra, Michael L Littman, and Nevin L Zhang. Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. 1997.

[2] Anthony R. Cassandra. *pomdp-solve: POMDP Solver Software*, 2015.

[3] Xi-Ren Cao De-Xin Wang. Event-Based Optimization for POMDPs and Its Application in Portfolio Management. *ScienceDirect*, 44:3228–3233, January 2011.

[4] Tianhu Deng, Zuo-Jun Max Shen, and J. George Shanthikumar. Statistical Learning of Service-Dependent Demand in a Multiperiod Newsvendor Setting. *Operations Research*, 62(5):1064–1076, October 2014.

[5] Michael Hahsler and Anthony R. Cassandra. *pomdpSolve: Interface to 'pomdp-solve' for Partially Observable Markov Decision Processes*, 2023. R package version 1.0.3.

[6] M. Hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94, August 2000. arXiv:1106.0234 [cs].

[7] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. June 2008.

[8] Michael Littman. The witness algorithm: Solving partially observable markov decision processes. 02 1995.

[9] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. pages 362–370, 1995.

[10] Edward Balaban Tim A. Wheeler Jayesh K. Gupta Maxim Egorov, Zachary N. Sunberg and Mykel J. Kochenderfer. POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty. *Journal of Machine Learning Research*, 18, April 2017.

[11] Tomáš Omasta. Efficient MDP Algorithms in POMDPs.jl. 2021.

[12] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. 2003.

[13] Katsushige Sawaki and Akira Ichikawa. Optimal control for partially observable markov decision processes over an infinite horizon. *Journal of The Operations Research Society of Japan*, 21:1–16, 1978.

[14] David Silver and Joel Veness. Monte-Carlo Planning in Large POMDPs. 2010.

[15] Edward Sondik. The optimal control of partially observable markov process over the infinite horizon: Discounted costs. *Operations Research*, 26:282–304, 04 1978.

[16] Zachary Sunberg and Mykel the s. Online algorithms for POMDPs with continuous state, action, and observation spaces. September 2018.

[17] Tarek Taha, Jaime Valls Miro, and Gamini Dissanayake. Wheelchair Driver Assistance and Intention Prediction Using POMDPs. December 2007.

[18] Travis B. Wolf and Mykel J. Kochenderfer. Aircraft Collision Avoidance Using Monte Carlo Real-Time Belief Space Search. *Jornal of Intelligent Robotic Systems*, January 2011.

[19] Kaiyu Zheng and Stefanie Tellex. pomdp_py: A framework to build and solve pomdp problems. In *ICAPS 2020 Workshop on Planning and Robotics (PlanRob)*, 2020. Arxiv link: "`https://arxiv.org/pdf/2004.10099.pdf`".

[20] zsunberg. Pomdps.jl: Concepts and architecture.