COMP3702 Assignment 3: MDP-based Investment Strategy

by Achintya Chavan #: 43213889

1. MDP Problem

The goal here is to develop a system that uses historical data from a customer/user's businesses (or ventures) to decide how the user will allocate fundings in order to maximise the profit in the long run. Here we do not know nor can accurately predict how many orders each of the ventures will get in a time period or fortnight so the funding model will need to incorporate the stochastic nature of the number of orders. Hence this problem is modelled using a Markov Decision Process in order to maximise the expected rewards. The MDP problem can be divided into four main parts -

- State Space (S)
- Action Space (A)
- Transition Function (T)
- Reward Function (R)

State Space

For this investment problem the state space is simply the manufacturing funding (f) given to the i^{th} venture (v_i) at the start of the fortnight (before orders are given). For a problem which contains N ventures this can be represented as

$$S = (f_{v_1}, f_{v_2}, \dots, f_{v_N})$$

Here f_{v_i} is a positive integer and all states are constrained by the maximum manufacturing fund, M that a user (venture owner) can allocate to all the ventures given as

$$f_{v_1} + f_{v_2} + \cdots + f_{v_N} = \sum_{i=1}^{N} f_{v_i} \leq M$$

Action Space

Similar to the state space, the action space is a combination of possible states of additional funding, a_i that a user can allocate to each of his ventures at the end of the fortnight (after orders) in preparation for the upcoming fortnight. The domain of the action space is the same as that of the state space.

$$A = (a_{v_1}, a_{v_2}, \dots, a_{v_N})$$

It is constrained by the maximum additional fund, E that a user can allocated to all the ventures and is given as

$$a_{v_1} + a_{v_2} + \cdots + a_{v_N} = \sum_{i=1}^{N} a_{v_i} \leq E$$

Furthermore allocating funds to a given state is only legal if it does not exceed M so an additional constraint is introduced here by modifying the state space constraint

$$f_{v_1} + a_{v_1} + \cdots + a_{v_N} + f_{v_N} = \sum_{i=1}^{N} (f_{v_i} + a_{v_i}) \leq M$$

Transition Function

Given the probabilistic nature of the outcomes of this problem, we need a function which defines the transition (in terms of probabilities) to another state (S') given our current state (S) and an action state (A). It is known that the orders for each venture are independent of each other so the transition states are independent thus being defined by their own function. The overall transition function is given as

$$T(S, A, S') = \prod_{i=1}^{N} T_i(s_i, a_i, s_i')$$

The probabilities can be obtained by indexing the probabilities matrices for a given (s_i, a_i, s_i'). If $s_i' > s_i + a_i$ then the s_i' is illegal because the additional funds have appeared out of nowhere and the action does not create a legal state. Hence $T = 0$ for this condition.

If $0 < s_i' \leq s_i + a_i \rightarrow T(s_i, a_i, s_i') = P_i[s_i + a_i, s_i + a_i - s_i']$. This means what is the conditional probability given the funding $(s_i + a_i)$ and the number of orders given as $(s_i + a_i - s_i')$

Finally if $s_i' = 0$ we know the order size was at least $s_i + a_i$ (orders matched the funding available). However it also implies that there were more orders but these could not be satisfied as the funds weren't available or were capped. So for this situation we need to know the cumulative probabilities of these excess orders

$$T(s_i, a_i, s_i') = \sum_{k=s_i+a_i}^{M} P_i[s_i + a_i, k] \quad (s_i' = 0)$$

The combined transition function for venture is given as

$$T_i(s_i, a_i, s_i') = \begin{cases} 0, & s_i' > s_i + a_i \\ P_i[s_i + a_i, s_i + a_i - s_i'], & 0 < s_i \leq s_i + a_i \\ \sum_{k=s_i+a_i}^{M} P_i[s_i + a_i, k], & s_i' = 0 \end{cases}$$

Reward Function

The reward function must seek to minimise the orders that cannot be fulfilled since these incur a penalty. Since we do not know anything about how many orders there could be for a particular state and action, we must therefore use the expected values of the orders in maximising the total utility. Hence the reward function is

$$R(s,a) \propto \sum Order \times P(Order|(S,A) - \sum \overline{Orders} \times P(\overline{Orders}|(S,A)$$

Here the first term is the sales/revenue term and the second term is the loss or penalty term. $\overline{Orders}$ is denoted as the number of unfulfilled orders. If it is zero then there is no penalty. We sum up the expected values for each order number because we are interested to know how many orders can be fulfilled given a particular (S, A).

Now to quantitatively define the function we know that orders can only be fulfilled up to $(s_i + a_i)$. If orders are greater than the maximum will be $(s_i + a_i)$ because we can't supply more. If it is lower than naturally the sales will be only made on that many orders. So the sales/revenue term can be expressed as

$$R_{isale}(s_i, a_i) = \sum_{k=1}^{M} \min(k, s_i + a_i)\, P_i[s_i + a_i, k]$$

The loss term is similar except it calculates the total expectation for orders that cannot be met. For a given number of orders, k and a particular (s_i, a_i) if $k > (s_i + a_i)$ then the penalty incurred would be for $k - (s_i + a_i)$ orders. If $k < (s_i + a_i)$ then naturally no penalties are incurred. It is expressed as

$$R_{iloss}(s_i, a_i) = \sum_{k=s_i+a_i+1}^{M} (k - s_i - a_i) P_i[s_i + a_i, k]$$

Now the venture owner can earn 60% from a sale and a 25% penalty for failing to satisfy the order. So these parameters can be used as coefficients in front of their respective functions thus giving us the final expression for the reward function of a venture i

$$R_i(s_i, a_i) = 0.6 \sum_{k=1}^{M} \min(k, s_i + a_i)\, P_i[s_i + a_i, k] - 0.25 \sum_{k=s_i+a_i+1}^{M} (k - s_i - a_i) P_i[s_i + a_i, k]$$

The overall reward function is the summation over all ventures which is

$$R(S, A) = \sum_{i=1}^{N} R_i(s_i, a_i)$$

2. Solution

The baseline solution used here is value iteration. Value iteration is an exhaustive iterative method that updates the optimal value functions, $V(S)$ of all states in the state space via the Bellman update equation. The pseudocode for value iteration is given below

Value Iteration Pseudocode

$\boldsymbol{V(s) \rightarrow V^0(s) = 0}$

Loop while $\boldsymbol{\left|V^{t+1}(s) - V^t(s)\right| > \varepsilon}$

  Loop for $\boldsymbol{s \in S}$

    Loop for $\boldsymbol{a \in A}$

      $\boldsymbol{Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s')}$

      $\boldsymbol{V(s) = \max_a Q(s,a)}$

  End loop

Output: V

Value Iteration for solving MDP problems will always produce optimal results since it is exhaustive and go through all the states. Hence once it is computed for the given state space the solution can be used from a lookup table during simulations. This is characterised as an offline solution because it generates the optimal policies independently of the simulation environment. However this also makes it very computationally expensive having a complexity of $O(s^2|a|)$ and so there is no guarantee of reaching the optimal policy (action) in a finite time although we are guaranteed to reach very close to the optimal solution after a while. This makes value iteration feasible for smaller problems.

Similar to value iteration, another offline method implemented here is policy iteration. In this method you start with a random policy (action), compute its value function and then find an improved policy based on the previous value function. This way each policy is a strict improvement over the previous one (unless of course the policy is already optimal). Policy iteration is very similar in working and implementation to value iteration except the difference is that convergence in policy iteration is based on the policy rather than the value function. In our case this makes policy iteration faster because the policies are discrete and therefore converge quickly to other discrete value compared to the value function which is continuous.

The issue with offline solution is that it tends to update all states which are not always feasible especially in a large state space. For these kind of problems it is more useful to Bellman update the relevant states. This can be achieved by the relaxing the optimality requirement of the solution to become approximately optimal. Hence for large problems the other solution being used is the Real Time Dynamic Programming (RTDP). RTDP algorithm performs a greedy search from the initial state till it reaches the goal state. During these iterations it only updates the states generated from the search. For this particular problem there is no goal state so instead the loop is constrained by the number of fortnights specified in the problem. The pseudocode for this method is given below.

RTDP is an online algorithm and so it requires simulations of the MDP environment to randomly sample and generate new states s'. For this problem a random generator is used to generate orders, s_{ord} for a given venture by sampling the rows of that venture's probability matrix (within the transition function). For example if $s = (2,1)$ then we take row-2 from P_1 (probability matrix of venture-1) and select a random column index from that row by comparing its corresponding element to a randomly generated value between 0 and 1. This is done for all other ventures and then combined to produce the next state s' which is the additional manufacturing funds.

RTDP Algorithm Pseudocode

Input: initial state S_o, *Fortnights left*

$\boldsymbol{V^0(s) = 0}$

t = 0

$\boldsymbol{s = S_o}$

Loop while *t* < *Fortnights left*

$\boldsymbol{a_{greedy} = \arg\max_a (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s'))}$

$\boldsymbol{V(s) = R(s, a_{greedy}) + \gamma \sum_{s' \in S} T(s, a_{greedy}, s')V(s')}$

$\boldsymbol{s' = \text{Sample } (T(s, a_{greedy}))}$

$\boldsymbol{s = s'}$

End loop

Output: $\boldsymbol{a_{greedy}[S_o]}$

3. Performance Evaluation

The first set of evaluations is done on the bronze test case using offline methods only since they converge quickly for the small state space of this problem.

Table-1: Bronze Test Case. (Statespace size is 10. Convergence threshold, $\varepsilon = 5 \times 10^{-5}$)

	Value Iteration		Policy Iteration	
s	$V(s)$	Best Action	$V(s)$	Best Action
(0,0)	2.80	(2,0)	2.80	(2,0)
(0,1)	2.92	(2,0)	2.92	(2,0)
(0,2)	2.47	(1,0)	2.47	(1,0)
(0,3)	1.90	(0,0)	1.90	(0,0)
(1,0)	3.35	(2,0)	3.35	(2,0)
(1,1)	2.92	(1,0)	2.92	(1,0)
(1,2)	2.47	(0,0)	2.47	(0,0)
(2,0)	3.35	(1,0)	3.35	(1,0)
(2,1)	2.92	(0,0)	2.92	(0,0)
(3,0)	3.35	(0,0)	3.35	(0,0)

The results from both methods are identical for this problem so any of one of these algorithms will suffice. As for timing, the value iterations takes approximately 15.86 secs on average whereas the policy iteration is 14 secs. It is difficult to gauge which method is faster but both are well within the maximum time of 3 mins.

For the other criterion, maximum profit at the end of the fortnights, the solutions yield the desired profit ($\geq 22.5 = 0.5 * 3 * 15$ for bronze test case) for 7 out of 100 runs (7%). It could be a problem with the MDP modelling or the test case. So it would be useful to test this for the next testcase which is the platinum problem.

For the platinum case, all three algorithms are used with the time taken given below

Table-2: Platinum Test Case. (Statespace size is 11. Convergence threshold, $\varepsilon = 0.5$)

Method	Value Iteration	Policy Iteration	RTDP
Time Taken(s)	172.86	116.15	12.33

Both offline methods for this problem are still within the time limit of 3 mins. It is also evident that the policy iteration is faster than value iteration as expected due to the discrete nature of convergence in this method. The RTDP computes all the fortnights within 13 secs so it also falls well under the required computation time of 30 secs per fortnight (for online methods) so it is definitely the better option for complex problems like gold and platinum.

The minimum profit needed for the platinum testcase is 25 $(0.5 * 5 * 10)$ and so the experimental success rate is about 89/100 runs ($\sim$ 90%).

Given that the offline methods still work for the platinum problem, it is well worth to use them instead of RTDP because they are exhaustive and update all states instead of randomly updating some. As a result I have decided to use policy iteration for solving all the given venture types for it is faster than value iteration and will generally yield the same results.