2017

COMP7702
Artificial Intelligence

ASSIGNMENT 3: MDP-BASED INVESTMENT STRATEGY

TEAM: SIRIUS

2017 SEMESTER 2 | University of Queensland, Brisbane

Table of Contents

1.0 Define MDP Problem

1.1 State Space

In this problem, a state is a tuple consisting the amount of funding each venture keeps after the orders of each fortnight. As per the problem description, the total amount of funding of a state should be no greater than the total reserved finding for manufacturing products. Thus, the state space is a collection of all valid states which are defined as follow.

1.1.1 Bronze User

Thus, the states of the problem can be represented as tuples, i.e.,

$$S_{Bronze}\ \{(S_{v1},\ S_{v2}) :\ S_{v1}, S_{v2} \in \{0, 1, 2, 3\}\ and\ S_{v1} +\ S_{v2}\ \le 3\}$$

where S_{v1} is the amount of funding business venture 1 holds and S_{v2} is the amount of funding business venture 2 holds every fortnight. This means there are actually 10 possible states.

1.1.2 Silver User

Thus, the states of the problem can be represented as tuples, i.e.,

$$S_{Silver}\ \{(S_{v1},\ S_{v2}),\ S_{v1}, S_{v2} \in \{0, 1, 2, 3, 4, 5\}\ and\ S_{v1} +\ S_{v2}\ \le 5\}$$

where S_{v1} is the total amount of funding business venture 1 holds and S_{v2} is the total amount of funding business venture 2 holds every fortnight. This means there are actually 21 possible states.

1.1.3 Gold User

Thus, the states of the problem can be represented as tuples, i.e.,

$$S_{Gold}\ \{(S_{v1},\ S_{v2},\ S_{v3}),\ S_{v1}, S_{v2}, S_{v3} \in \{0, 1, 2, 3, 4, 5, 6\}\ and\ S_{v1} +\ S_{v2} +\ S_{v3}\ \le 6\}$$

where S_{v1} is the total amount of funding business venture 1 holds, S_{v2} is the total amount of funding business venture 2 holds and S_{v3} is the total amount of funding business venture 3 holds every fortnight. This means there are actually 28 possible states.

1.1.4 Platinum User

Thus, the states of the problem can be represented as tuples, i.e.,

$$S_{Platium}\ \{(S_{v1},\ S_{v2},\ S_{v3}),\ S_{v1}, S_{v2}, S_{v3} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}\ and\ S_{v1} +\ S_{v2} +\ S_{v3}\ \le 8\}$$

where S_{v1} is the total amount of funding business venture 1 holds, S_{v2} is the total amount of funding business venture 2 holds and S_{v3} is the total amount of funding business venture 3 holds every fortnight. This means there are actually 45 possible states.

1.2 Action Space

The only action the business owner takes is to allocate funds at the beginning of the fortnight based on the state of last fortnight. It is a tuple of additional funding for each venture and the total amount of funding should be no greater than the total amount of funding. Selling products does not need to be modelled as an action, since it is done by the customer and the only decision for owner is how much funding to be added fortnightly. Thus, the action space is a collection of all valid actions and defined as follow.

1.2.1 Bronze User

Thus, the action space is:

$$A_{Bronze}\ \{(a_{v1},\ a_{v2}):\ a_{v1}, a_{v2} \in \{0,1,2,3\}\ and\ a_{v1} +\ a_{v2}\ \leq 3\}$$

where, on every fortnight, a_{v1} is the amount of funding added to the business venture 1 and a_{v2} is the amount of funding added to the business venture 2.

1.2.2 Silver User

Thus, the action space is:

$$A_{Silver}\ \{(a_{v1},\ a_{v2}),\ a_{v1}, a_{v2} \in \{0,1,2,3,4\}\ and\ a_{v1} +\ a_{v2}\ \leq 4\}$$

where, on every fortnight, a_{v1} is the amount of funding added to the business venture 1 and a_{v2} is the amount of funding added to the business venture 2.

1.2.3 Gold User

Thus, the action space is:

$$A_{Gold}\ \{(a_{v1},\ a_{v2},\ a_{v3}),\ a_{v1}, a_{v2}, a_{v3} \in \{0,1,2,3,4\}\ and\ a_{v1} +\ a_{v2} +\ a_{v3}\ \leq 4\}$$

where, on every fortnight, a_{v1} is the amount of funding added to the business venture 1, a_{v2} is the amount of funding added to the business venture 2 and a_{v3} is the amount of funding added to the business venture 3.

1.2.4 Platinum User

Thus, the action space is:

$$A_{Platium}\ \{(a_{v1},\ a_{v2},\ a_{v3}),\ a_{v1}, a_{v2}, a_{v3} \in \{0,1,2,3,4,5\}\ and\ a_{v1} +\ a_{v2} +\ a_{v3}\ \leq 5\}$$

where, on every fortnight, a_{v1} is the amount of funding added to the business venture 1, a_{v2} is the amount of funding added to the business venture 2 and a_{v3} is the amount of funding added to the business venture 3.

1.3 Transition Function

The transition function defines the probabilistic transitions to next states, given current states and actions, notated as $T\ (s, a, s')$. The transition function has three components where s represents a state at the beginning of a fortnight and s' is the next state after performing an action a in the state s. As mentioned in the problem specification, we know that the transition for each business venture is independent of one another, so we can express the transition function in terms of separate functions T_{v1}, T_{v2}, T_{v3} with the overall transition probability being defined as:

For *Bronze* and *Silver* Users:

$$T\big((v1, v2), (a_{v1}, a_{v2}), (v1', v2')\big) = T_{v1}\big(S_{v1},\ a_{v1,}\ S_{v1}{}'\big) \cdot T_{v2}\big(S_{v2},\ a_{v2,}\ S_{v2}{}'\big)$$

For *Gold* and *Platinum* Users:

$$T\big((v1, v2, v3), (a_{v1}, a_{v2}, a_{v3}), (v1', v2', v3')\big) = T_{v1}\big(S_{v1},\ a_{v1,}\ S_{v1}{}'\big) \cdot T_{v2}\big(S_{v2},\ a_{v2,}\ S_{v2}{}'\big) \cdot T_{v3}\big(S_{v3},\ a_{v3,}\ S_{v3}{}'\big)$$

1.3.1. Bronze User

We can define T as follows:

$$T_{v1}(S_{v1} + a_{v1}, S_{v1}{}') \begin{cases} 0, & if\ S'_{v1} > S_{v1} + a_{v1} \\ P_{v1}[S_{v1} + a_{v1}, S_{v1} + a_{v1} - S'_{v1}], & if\ 0 < S'_{v1} \le S_{v1} + a_{v1} \\ \sum_{i=S_{v1}+a_{v1}}^{3} P_{v1}[S_{v1} + a_{v1}, i], & if\ S_{v1}{}' = 0 \end{cases}$$

as similarly for T_{v2}.

Finally, the complete transition function (as stated before) is

$$T\big((v1, v2), (a_{v1}, a_{v2}), (v1', v2')\big) = T_{v1}\big(S_{v1},\ a_{v1,}\ S_{v1}{}'\big) \cdot T_{v2}\big(S_{v2},\ a_{v2,}\ S_{v2}{}'\big)$$

1.3.2. Silver User

We can define T as follows:

$$T_{v1}(S_{v1} + a_{v1}, S_{v1}{}') \begin{cases} 0, & if\ S'_{v1} > S_{v1} + a_{v1} \\ P_{v1}[S_{v1} + a_{v1}, S_{v1} + a_{v1} - S'_{v1}], & if\ 0 < S'_{v1} \le S_{v1} + a_{v1} \\ \sum_{i=S_{v1}+a_{v1}}^{5} P_{v1}[S_{v1} + a_{v1}, i], & if\ S_{v1}{}' = 0 \end{cases}$$

as similarly for T_{v2}.

Finally, the complete transition function (as stated before) is

$$T\big((v1, v2), (a_{v1}, a_{v2}), (v1', v2')\big) = T_{v1}\big(S_{v1},\ a_{v1,}\ S_{v1}{}'\big) \cdot T_{v2}\big(S_{v2},\ a_{v2,}\ S_{v2}{}'\big)$$

1.3.3 Gold User

We can define T as follows:

$$T_{v1}(S_{v1}+a_{v1}, S_{v1}') \begin{cases} 0, & if\ S_{v1}' > S_{v1}+a_{v1} \\ P_{v1}[S_{v1}+a_{v1}, S_{v1}+a_{v1}-S_{v1}'], & if\ 0 < S_{v1}' \leq S_{v1}+a_{v1} \\ \sum_{i=S_{v1}+a_{v1}}^{6} P_{v1}[S_{v1}+a_{v1}, i], & if\ S_{v1}' = 0 \end{cases}$$

as similarly for T_{v2} and T_{v3}.

Finally, the complete transition function (as stated before) is

$$T\big((v1, v2, v3), (a_{v1}, a_{v2}, a_{v3}), (v1', v2', v3')\big) = T_{v1}\big(S_{v1}, a_{v1,} S_{v1}'\big) \cdot T_{v2}(S_{v2}, a_{v2,} S_{v2}') \cdot T_{v3}(S_{v3}, a_{v3,} S_{v3}')$$

1.3.4 Platinum User

We can define T as follows:

$$T_{v1}(S_{v1}+a_{v1}, S_{v1}') \begin{cases} 0, & if\ S_{v1}' > S_{v1}+a_{v1} \\ P_{v1}[S_{v1}+a_{v1}, S_{v1}+a_{v1}-S_{v1}'], & if\ 0 < S_{v1}' \leq S_{v1}+a_{v1} \\ \sum_{i=S_{v1}+a_{v1}}^{8} P_{v1}[S_{v1}+a_{v1}, i], & if\ S_{v1}' = 0 \end{cases}$$

as similarly for T_{v2} and T_{v3}.

Finally, the complete transition function (as stated before) is

$$T\big((v1, v2, v3), (a_{v1}, a_{v2}, a_{v3}), (v1', v2', v3')\big) = T_{v1}\big(S_{v1}, a_{v1,} S_{v1}'\big) \cdot T_{v2}(S_{v2}, a_{v2,} S_{v2}') \cdot T_{v3}(S_{v3}, a_{v3,} S_{v3}')$$

1.4 Reward Function

As specified in the question, the owner gets 60% of the sale price as profit and 25% as the penalty. Thus, the reward function for a venture v_n can be defined as:

$$R_{v_n}(k) = SalePrice(v_n) * \sum_{i=0}^{N} P(k,i) * (0.6 * \min(k,i) - 0.25 * \max(0, i-k))$$

where N is the total reserved funding for manufacturing products, k is the row in probability matrix which represents $S_{v1}+a_{v1}$ and i is the column in probability matrix which represents the number of orders.

1.4.1 Bronze User

As with the transition function, reward function can be split into components, i.e.,

$$R\big((S_{v1}, S_{v2}), (a_{v1}, a_{v2})\big) = R_{v1}(S_{v1}+a_{v1}) + R_{v2}(S_{v2}+a_{v2})$$

where R_{v1} is the reward for the venture 1 and R_{v2} is the reward for the venture 2.

Additionally, we can derive R_{v1}, R_{v2} as

$$R_{v_n}(k) = SalePrice(v_n) * \sum_{i=0}^{3} P(k,i) * (0.6 * \min(k,i) - 0.25 * \max(0, i-k))$$

where n is the venture id, $k \in [0,3]$ and $i \in [0,3]$

1.4.2 Silver User

As with the transition function, reward function can be split into components, i.e.,

$$R\big((S_{v1}, S_{v2}), (a_{v1}, a_{v2})\big) = R_{v1}(S_{v1} + a_{v1}) + R_{v2}(S_{v2} + a_{v2})$$

where R_{v1} is the reward for the venture 1 and R_{v2} is reward for the venture 2.

Additionally, we can derive $R_{v1},\ R_{v2}$ as

$$R_{v_n}(k) = SalePrice(v_n) * \sum_{i=0}^{5} P(k,i) * (0.6 * \min(k,i) - 0.25 * \max(0, i-k))$$

where n is the venture id, $j \in [0,5]$ and $k \in [0,5]$

1.4.3 Gold User

As with the transition function, reward function can be split into components, i.e.,

$$R\big((S_{v1}, S_{v2}, S_{v3}), (a_{v1}, a_{v2}, a_{v3})\big) = R_{v1}(S_{v1} + a_{v1}) + R_{v2}(S_{v2} + a_{v2}) + R_{v3}(S_{v3} + a_{v3})$$

where R_{v1} is the reward for the venture 1, R_{v2} is the reward for the venture 2 and R_{v3} is the reward for the venture 3.

Additionally, we can derive $R_{v1},\ R_{v2},\ R_{v3}$ as

$$R_{v_n}(k) = SalePrice(v_n) * \sum_{i=0}^{6} P(k,i) * (0.6 * \min(k,i) - 0.25 * \max(0, i-k))$$

where n is the venture id, $j \in [0,6]$ and $k \in [0,6]$

1.4.4 Platinum User

As with the transition function, reward function can be split into components, i.e.,

$$R\big((S_{v1}, S_{v2}, S_{v3}), (a_{v1}, a_{v2}, a_{v3})\big) = R_{v1}(S_{v1} + a_{v1}) + R_{v2}(S_{v2} + a_{v2}) + R_{v3}(S_{v3} + a_{v3})$$

where R_{v1} is the reward for the venture 1, R_{v2} is the reward for the venture 2 and R_{v3} is the reward for the venture 3.

Additionally, we can derive $R_{v1},\ R_{v2}, R_{v3}$ as

$$R_{v_n}(k) = SalePrice(v_n) * \sum_{i=0}^{8} P(k,i) * (0.6 * \min(k,i) - 0.25 * \max(0, i-k))$$

where n is the venture id, $j \in [0,8]$ and $k \in [0,8]$

1.5 Discount Factor

In this application, the discount factor given in the input will be used to reduce the impact of later performance.

2.0 Problem Solution

In this application, we use Asynchronous Value Iteration to solve this problem. As an offline method, we need to calculate the optimal policy (π · (s)), which indicates the action to be taken in each state to achieve the maximum profit. The pseudo code is attached below.

```
$Initialize\ v^0(S) = R(s)\ for\ all\ s\ in\ S.$
$Loop\ until\ converge$
$For\ all\ s\ in\ S\ \{$
    $For\ all\ a\ in\ A\ \{$
        $V^{t+1}(s) = \overset{max}{a}(R(s,a) + \gamma \sum_{s'} T(s,a,s^{'})V^t(s'))$
    $\}$
    $V^t(s) = V^{t+1}(s)$
$\}$
$t = t + 1$
```

At first, 0 is assigned as the initial value for each state. The application then iterates until the convergence, where the 10^{-7} is set as the threshold. For each iteration t and state S, the Asynchronous Value Iteration algorithm performs all valid actions in this state to determine a maximum value $V^*(s)$ and corresponding best action A^*. After traversing all possible actions in state S, an optimal result A^* is recorded in policy and the value of the state S is updated by the maximum value $V^*(s)$. A new iteration $t+1$ occurs with the same process when values in all states are updated. The algorithm finishes if states s meet the condition of $|V^{t+1}(s) - V^t(s)| < 10^{-7}$.

The Bellman Update in this example is

$$V^{t+1}(s) = \overset{max}{a}(R(s,a) + \gamma \sum_{s\prime} T(s,a,s')V^t(s'))$$

The immediate reward is $R\ (s,a)$ rather than $R(s)$. As defined in question 1, $R\ (s,a) = R\ (s + a) = R\ (s')$. When all iterations finish, the program will have an optimal policy to suggest the best approach to achieve the maximized profits.

3.0Analysis of the Implemented Algorithm

3.1 Synchronous Value Iteration vs. Asynchronous Value Iteration

The main difference between the Basic Synchronous Value Iteration and the Asynchronous Value Iteration is at the time of updating the value function.

```
$Initialize\ v^0(S) = R(s)\ for\ all\ s\ in\ S.$
$Loop\ until\ converge$
$For\ all\ s\ in\ S\ \{$
    $For\ all\ a\ in\ A\ \{$
        $V^{t+1}(s) = \max_a (R(s) + \gamma \sum_{s'} T(s,a,s') V^t(s'))$
    $\}$
    $V^t(s) = V^{t+1}(s)$ // Asynchronous Value Iteration
$\}$
$For\ all\ s\ in\ S$
    $V^t(s) = V^{t+1}(s)$ // Synchronous Value Iteration

$t = t + 1$
```

For the Basic Asynchronous Value Iteration, the value function is updated when and only when new values in all states are calculated. That means $V^t(s)$ would not change during the Bellman Update in each iteration. By contrast, for each state s in the Asynchronous Value Iteration, a new $V(S)$ is updated directly before the loop of the next state.

3.2 Accuracy and Speed

Since both value iteration methods implement the Bellman Update, using the offline, they would produce the same optimal policy. Thus, the accuracy of these two methods are theoretically the same.

In terms of the time efficiency, the Asynchronous Value Iteration slightly reduce its time to converge. $V^{t+1}(s)$ is more likely to be closer to the convergence value because $V^{t+1}(s)$ is calculated based on $V^t(s')$ that has been already updated.

$$V^{t+1}(s) = \max_a (R(s) + \gamma \sum_{s'} T(s,a,s') V^t(s'))$$

However, the time complexity for both methods is $O(AS^2)$, where A is the size of the action space and S is the size of the state space. Since the required running time mostly depends on the action space and state space, the difference between two methods are not significant.

The test result below shows that the Asynchronous Value Iterations can saves 1/4 to 1/3 running time on average, compared to the Synchronous one.

Times \ Cases	bronze1 (ms)	Silver1 (ms)	Gold1 (ms)	Platinum (ms)
1	29	79	481	2098
2	32	86	421	2114
3	27	84	423	2080
4	29	72	423	2056
5	38	79	419	2073
6	25	72	430	2014
7	22	67	419	2044
8	40	94	424	2043
9	31	74	436	1988
10	21	79	452	2079

Table 1 Asynchronous Value iteration

Times \ Cases	bronze1 (ms)	Silver1 (ms)	Gold1 (ms)	Platinum (ms)
1	61	90	737	3255
2	50	82	657	3257
3	35	84	642	3285
4	37	87	651	3164
5	37	81	643	3221
6	67	81	673	3192
7	34	93	679	3195
8	59	80	646	3225
9	37	87	667	3220
10	39	83	668	3150

Table 2 Synchronous Value Iteration

3.3. Value Iteration vs. Monte Carlo Tree Search

Monte Carlo Tree Search algorithm is also implemented to solve problems with huge state space and action space. Since it is not always practical to enumerate all possible cases to solve some problems, a sampling-based could be a good choice to obtain a much better solution rather than the best one. MCTS is such kind of method which performs as many simulations as possible to get a better result in a given time interval.

The MCTS algorithm consists of 4 main steps, which are Selection, Expansion, Simulation and Backpropagation namely.

3.3.1 Selection and Expansion

1. If the root node has not been expanded, expand it.
2. If the root node has child which has not been simulated, return that child.
3. If the root is at the last levels of the tree, return best node by UCT.
4. Else get best node by UCT and recall this function recursively.

3.3.2 Simulation

Simulate the funding allocate process for 10 fortnights.

3.3.3 Backpropagation

Back to the root and update values of the nodes in the path.

```
Initialize root node.
Loop until time run out {
    node = SelectPromisingNode(root)
    performSimulation(node);
    backPropogation(node);
}
action = argmax(node.getOptimalChild())
Return action

selectPromisingNode(root){
    if root not expand
        expand root;
    for child in root.getChilden(){
        If child has not been simulated
            return child;
    }
    if(node.level == numberOfFortnights){
        node = findBestNodeByUCT(root);
        return node;
    }else{
        selectPromisingNode(findBestNodeByUCT(root));
    }
}
```

```
performSimulation(node, n){
    state = node.getState();
    loop  n times
        randomAction();
        randomOrder();
        updateProfit();
    return totalProfit;
}
backPropogation(node){
    while(node!= null){
        updateValue();
        node = node.parent;
    }
}
```

To select the promising node, Upper Confidence Bound(UCB) is used.

$$\pi_{UCT}(s) = arg\max_{a\in A} Q(s,a) + c\sqrt{\frac{\ln(n(s))}{n(s,a)}}$$

$n(s)$: #times node s has been visited.

$n(s,a)$: #times the out-edge of s with label a has been visited

c: Use 1.414 to balance exploration and exploitation.

The value estimates are based on stochastic simulations, so nodes must be visited several times to make the missing-part estimates more credible; the MCTS estimates will be less reliable at the beginning of the search and will eventually converge after a given enough time to a more reliable estimate, an optimal estimate can be reached in infinite time.

3.4 Accuracy and Speed

According to Jin & Benjamin (2015), the time complexity of the MCTS is the $O(mkI/C)$, where m is the number of random children to consider per search, k is the number of simulations of a child, I is the number of iterations and C is the number of cores available. Compared to value iteration, MCTS allows to obtain a decent result when dealing with large state space as it is no necessary to consider all states but just m states which can be reached from the current state. However, larger values of k and I enables to get better result but the time-consuming increases as well. Thus, MCTS is like a trade-off between accuracy and speed. Apart from that, the time needed for simulation should also be considered especially when solving a MDP problem with a small state space. If once simulation needs more time than traverse all state, MCTS is less capable of the problem. This is also the reason why we choose to use Asynchronous Value Iteration rather than MCTS.

Reference

Fragkiadaki, K. (2017). Planning in Markov Decision Processes. Retrieved form https://www.cs.cmu.edu/~katef/DeepRLControlCourse/lectures/lecture3_mdp_planning.pdf

Jin, Y., & Benjamin, S. (2015). CME 323, Report. Retrieved from http://stanford.edu/~rezab/classes/cme323/S15/projects/montecarlo_search_tree_report.pdf

Kurniawati, H. (2017). Markov Decision Processes. Retrieved from http://robotics.itee.uq.edu.au/~ai/lib/exe/fetch.php/wiki/planunderunc3-mdp-aftlec.pdf