SockFarm Project

Sayak Chakrabarty
Dartmouth College, USA, Sayak.Chakrabarty.GR@Dartmouth.edu

Andrea Pugliese
University of Calabria, Italy, andrea.pugliese@unical.it

V.S. Subrahmanian
Dartmouth College, USA, vs@dartmouth.edu

Youzhi Zhang
Dartmouth College, USA, youzhi.zhang@dartmouth.edu

November 9, 2020

1 Introduction

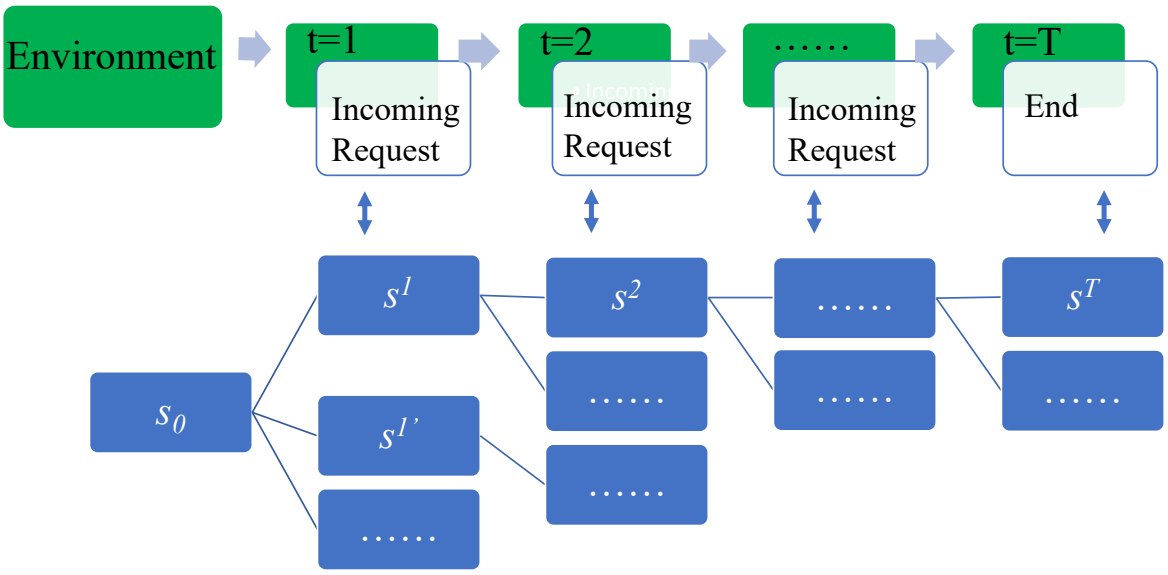

Figure 1: The operator's MDP

A sockfarm operator helps customers to post fake reviews to promote their products. The operator tries to finish each request from customers for posting reviews while minimizing the cost (e.g., the cost to post reviews and the cost for being detected by the shopping platform). At each time step, the operator receives incoming requests and assigned some of his controlled accounts to post reviews for some requests based on his observation (current state). We use the Markov Decision Process (MDP) to model this sequential decision process of the operator shown in Figure 1. Each state in the MDP includes the information about his controlled accounts (the possible detected

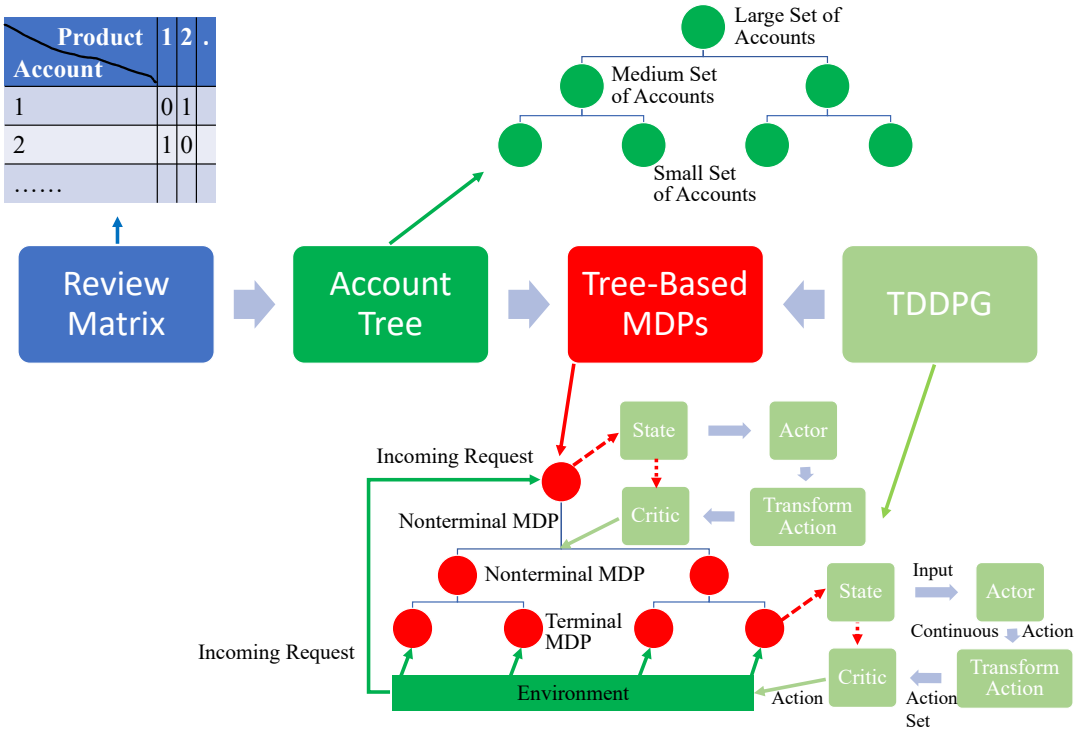

Figure 2: Our solution framework

probability of each account) and the unfinished requests. At each state, the the operator will receive the incoming request from the environment and then take an action (assigning some accounts to post reviews for some request). The operator will receive the reward (the payoff from finished requests and cost to take actions) from the environment, and the environment will determine the transition from the current state to the next state based on the probability of the incoming requests and the probability of that the detection probability of each account changes after taking actions. The operator's optimal policy in this MDP will maximize the operator's expected reward, i.e., finishing requests while minimizing cost, where each state's executed action is best after taking the future reward into account.

However, it is almost impossible to compute the operator's optimal policy because the operator's MDP has an extremely large action space. To conquer this challenge, we propose our novel solution approach to transform the large MDP into small MDPs shown in Figure 2. Motivated by that some accounts are independent of each other (do not post reviews for the same product), starting from the review matrix including the information on whether an account has posted a review for a product or not, we create an account tree to separate the large set of accounts into small sets of accounts. Based on that, we build an MDP on each set of accounts to form the tree-based MDPs on the tree. Each MDP on the tree is similar to the operator MDP in Figure 1. Here, each state in the MDPs of nonterminal nodes in the tree has two kinds of actions (actually, they are corresponding to the branches of this node): assigning how many reviews of the incoming request to the right or left group, and the operator in the MDPs of the terminal nodes in the tree take actions to complete each assigned request at each state. The actions of the terminal MDPs interact with the environment, and the incoming request of the environment goes to the MDP on the root of the tree. In addition, the tree-based MDPs influence each other according to the hierarchical structure in the tree. Here,

the policies of low-level MDPs influence the transition function and reward function of high-level MDPs, while the policies of high-level MDPs influence the transition function of the low-level MDPs. The policies of these MDPs form an tree-based policy. Then we propose our algorithm (TDDPG) to solve the tree-based MDPs. TDDPG is able to handle the tree structure by using the tree-based traversal for updating parameters and able to handle the large action space in our problem by using action transformation.

2 Motivation and Problem Description

In this section, we first describe the scenario that the operator posts fake reviews. We then formally define the problem that the operator tries to finish each request for posting reviews while minimizing the cost. After that, we use the Markov Decision Process (MDP) to model the operator's sequential decision process to maximize the profit.

2.1 Motivation

Nowadays, when we are shopping online, the reviews can give more information on the product and then help us with which products are good or bad. Most of these reviews are verified purchases, which means that the reviews are posted by users who have purchased the product. However, many of these reviews are fake reviews.[1]

A higher rate means more sales for a product. Therefore, when shopping platforms, e.g., Amazon, try to forbid fake reviews, sellers are trying to "game the system" to increase their products' ratings. One of these methods to game the system is buying service from thirty-party companies who promise to provide 'real' reviews. Indeed, there many such third parties online[2].

After sellers buy fake reviews, the operator of these services will post reviews from the various accounts he maintains. Evidences show that the operator will 'game the system' to avoid being detected. For example, the operator considers how to post reviews to avoid being detected as fake reviews: "We can add all reviews at once but this can make your reviews look fake if you are not getting reviews at the same period of time in which we have provided. Also Lazada make remove all your reviews as well."[3] Correspondingly, the operator's strategy is gradually posting reviews: "Depending on the quantity you have purchased we can provide 1 review per day if your product is not sell hot. Otherwise we can provide 2-3 reviews per day."[4] In this work, we play the role of the operator to maximize his profit while avoiding being detected.

2.2 Problem Description

The sellers send requests to the operator of the fake reviews platform for posting fake reviews to the corresponding product. Assume, during T time intervals, the operator receives a set of requests **REQ** for the corresponding set of products $\mathcal{P}$.

Definition 2.1 (A Request)**.** *A* Request *is a tuple* $\mathbf{r} = (p, n, d, cp) \in \mathbf{REQ}$*, where* $p \in \mathcal{P}$*,* n *is the number of reviews to be written,* $d \leq T$ *is the deadline for the job to be finished, and* cp *is the price the operator will change the customer as soon as the job is finished.*

[1] https://reviewmeta.com/

[2] E.g., https://woorke.com/, https://www.amzonreview.com/, https://alpharavenhouse.com/

[3] https://woorke.com/buy/lazada

[4] https://woorke.com/buy/amazon-reviews

The number of reviews n in a request changes dynamically. For example, if a review is posted for a request $\mathbf{r} = (p, n, d, cp)$, then $\mathbf{r}$ becomes $(p, n-1, d, cp)$. During this period, the number of reviews n also represents the remaining task for request $\mathbf{r}$, and request $\mathbf{r}$ is finished if $n = 0$. If the time has not reached the deadline d with $n > 0$, then $\mathbf{r}$ is called unfinished request. Let $\mathbf{REQ}^t(\subseteq \mathbf{REQ})$ is a set of new requests that probably appear at time t, and $Pr(\mathbf{REQ}'^t)$ is the probability that a subset of new requests $\mathbf{REQ}'^t(\subseteq \mathbf{REQ}^t)$ appear.

The operator can deploy a set of accounts $\mathcal{A}$ to post reviews. Let $\mathcal{REV}$ be the set of reviews, where each review in $\mathcal{REV}$ is a pair (a, p) with $a \in \mathcal{A}$ and $p \in \mathcal{P}$. Let $\mathcal{REV}(\mathcal{A}', \mathbf{REQ}')$ be a set of reviews that the set of accounts $\mathcal{A}'(\subseteq \mathcal{A})$ can post for the set of request $\mathbf{REQ}'(\subseteq \mathbf{REQ})$. A *review fraud detection algorithm* of the shopping platform computes, for each account $a \in \mathcal{A}$, the probability of the account a being fraudulent. That is, $dp(a)$ is the detection probability of account a, which will change after a posts more and more reviews. If an account is detected, the reviews posted by this account will be removed by the shopping platform. If account a posted a review for an unfinished request $\mathbf{r} = (p, n, d, cp)$ and is detected, then $\mathbf{r}$ changes to $= (p, n+1, d, cp)$.

Let c_{setup} be the setup cost of each account, which will be charged after an account posts a review. Let c_{rev} be the cost of writing each review. Let c_a be the cost of account a is detected. The operator' profit is the payoff from finished requests minus the cost. To maximize the profit, the operator needs to decide to use which accounts to post reviews for which requests to finish the job while minimizing the probability of being detected at each time step.

2.3 Operator's MDP

We use the Markov Decision Process (MDP) to model the operator's sequential decision process. The operator's MDP is a tuple $(S, A, \mathcal{TR}, R, \pi)$ in which:

- S is a set of states. Each state $s = (t_s, \{dp_s(a) \mid a \in \mathcal{A}\}, \mathbf{REQ}(s), \mathcal{REV}(s))$, where t_s with $0 \le t_s \le T$ represents the time step, $dp_s(a)$ is the detection probability of account a at state s, $dp(a) = 1$ if account a is detected, $\mathbf{REQ}(s)(\subseteq \mathbf{REQ})$ is a set of unfinished requests including residual unfinished requests from previous states and the new coming request, and $\mathcal{REV}(s)$ is a set of reviews posted for requests in $\mathbf{REQ}(s)$. Let $D(s)$ be the set of accounts detected at state s, $\mathbf{REQ}_{t_s}(s)$ be the set of new requests added to state s, $\mathbf{REQ}^*_{t_s}(s)$ be the set of new finished requests at state s, and $\mathcal{REV}^*(s)$ is the set of new posted reviews to reach state s. The initial state $s_0 = (0, \{dp_{s_0}(a) \mid a \in \mathcal{A}\}, \emptyset, \emptyset)$. Let S^t be a set of terminal states where for each terminal state s^t, $t_{s^t} = T$ or $dp_{s^t}(a) = 1(\forall a \in \mathcal{A})$.

- A is a set of actions for choosing which accounts to post reviews for which requests, and $A(s) = pow(\mathcal{REV}'(s))$ is the set of available actions for state s, where $\mathcal{REV}'(s) = \mathcal{REV}(\{a \mid dp_s(a) < 1, a \in \mathcal{A}\}, \mathbf{REQ}(s)) \setminus \mathcal{REV}(s)$, and $pow(\mathcal{REV}'(s))$ is the power set of $\mathcal{REV}'(s)$. That is, each action α at state s is a subset of possible reviews in $\mathcal{REV}'(s)$. $A(s^t) = \emptyset(\forall s^t \in S^t)$.

- $\mathcal{TR}$ is the transition function, which is defined as $\mathcal{TR} : S \times A \times S \to [0, 1]$, i.e. the probability of ending up in state s' after doing action $\alpha \in A(s)$ in state s is denoted $\mathcal{TR}(s, \alpha, s')$. Here, $\mathcal{TR}(s, \alpha, s') = Pr(\mathbf{REQ}_{t_{s'}}(s')) \times Pr(\{dp_{s'}(a), a \in \mathcal{A}\} \mid \{dp_s(a), a \in \mathcal{A}\}, \alpha)$, which is a combination of the probability of new appearing requests and the probability of that the detection probability of each account changes after taking action α.

- R is the reward function, i.e., $R : S \times A \to \mathbb{R}$, which considers the payoff for finished requests and the cost for the detected accounts and posting reviews. That is, $R(s, \alpha) = \sum_{\mathbf{r} \in \mathbf{REQ}^*_{s,\alpha}} cp_{\mathbf{r}} - \sum_{a \in D(s,\alpha)} c_a - C_\alpha$, where $cp_{\mathbf{r}}$ is the payoff for finished request $\mathbf{r}$, C_α is the

cost of posting the set of reviews by taking α , $\mathbf{REQ}^*_{s,\alpha}$ is the set of finished request after taking α at s, and $D(s,\alpha)$ is the set of detected accounts after taking α at s.

- $\pi : S \to A$ is a deterministic policy that assigns an action for each state. The operator's expected reward is $V(\pi) = \mathbb{E}_{t=1}^{T} \gamma^t R(s_{t-1}, \alpha_{t-1})$, where $\gamma \in (0, 1]$ is the discounted factor. Denote the policy space as Π, then solving the operator's MDP is to compute the optimal policy $\pi^\star$ such that $V(\pi^\star) = \max_{\pi \in \Pi} V(\pi)$.

	1	2	3	4	5	6	7	8	9
a	1		1						
b	1	1			1				
c		1	1						
d	1			1					
e		1		1					
f			1		1				
g						1		1	
h						1	1		
i							1		1
j						1		1	1

Table 1: Review matrix decomposition: Only a small part of users post reviews for a request (product), and a user only posts reviews for a small part of requests (products). In this example, there are users (accounts) a–j and products 1–9. the submatrix with users a–f and products 1–5 is independent of the submatrix with users g–j and products 6–9. Intuitively, the detection probability of a user on one matrix is independent of the detection probability of a user on the other matrix.

3 Solution Approach: Hierarchical Solution of the Large MDP

The operator's MDP has an extremely large action space. For example, some operators can control more than one hundred thousand accounts to post reviews [5]. Then, even there are at most 100 reviews to be posted, the number of actions at a state will be more than 2^{100}. The extremely large action space will lead to extremely large state space, and then make general MDP solving approaches, e.g., dynamic programming, reinforcement learning, fail to solve the operator's MDP. In this section, we propose a hierarchical approach to conquer this challenge. Firstly, we show the motivation for using the hierarchical approach. Secondly, we show how to efficiently create a hierarchical structure in our problem. Thirdly, we formulate our tree-based MDPs. Finally, we propose our approach to solve the tree-based MDPs.

3.1 Motivation: Review Matrix and Account Graph

A review matrix M_R includes the information on whether an account has posted a review for a product or not, i.e., $M_R(a, j) = 1$ represents that account a has posted a review for a product j. In real-world, the review matrix is sparse: only a small part of users post reviews for a product, and a user only posts reviews for a small part of products. In the example of Table 1, the submatrix with users a–f and products 1–5 is independent of the submatrix with users g–j and products 6–9.

[5] https://amztigers.com/, Accessed: 2020.09.03

Intuitively, the detection probability of a user on one submatrix is independent of the detection rate of a user on the other submatrix in this example. Our goal is to finish each request with the least cost. If we can assign a small group of accounts to a request, then we can separate the original MDP into small MDPs. Each small MDP includes a small group of accounts and a request (or several requests) and is solved independently. Note that, each small MDP will have a much smaller action space than the action space in the original MDP.

Our decomposition for the account set is based on the property of the review matrix. Given account a, let $\mathcal{P}(a)$ be the set of products where a has posted reviews. The similarity between two accounts is determined by the products they have posted reviews, which is to measure the correlation of the detection probabilities of two accounts.

Definition 3.1. *The similarity between accounts a_i and a_j is measured by Jaccard similarity coefficient:*

$$J(a_i, a_j) = \frac{|\mathcal{P}(a_i) \cap \mathcal{P}(a_j)|}{|\mathcal{P}(a_i) \cup \mathcal{P}(a_j)|} \tag{1}$$

where $J(a_i, a_j) = 0$ if $\mathcal{P}(a_i)$ or $\mathcal{P}(a_j)$ is $\emptyset$.

Note: We might use any method for determining similarity, Jaccard coefficient is just an example.

Definition 3.2. *The τ-account graph is $G = (V, E)$, where $V = \mathcal{A}$ and $E = \{(a_i, a_j) \mid J(a_i, a_j) > \tau, a_i, a_j \in \mathcal{A}\}$.*

Assumption 1. *The detection probabilities $dp(a_i)$ and $dp(a_j)$ of accounts a_i and a_j are independent if $J(a_i, a_j) = 0$.*

Based on this assumption, we can separate the whole graph G into independently subgraphs that are corresponding to independent small MDPs before solving the operator's MDP. We say account a_i is independent of a set of accounts $\mathcal{A}'$ if $J(a_i, a_j) = 0$ for each account $a_j \in \mathcal{A}'$, and two account sets A_i and A_j are independent if $J(a_i, a_j) = 0$ for all accounts $a_i \in A_i$ and $a_j \in A_j$. We can separate the accounts into independent subsets, and then assign an independent subset of accounts to finish a request. For the high-level MDP, the operator needs to assign each incoming request to an independent group of accounts. For the low-level MDP including a small group of accounts and request, the operator needs to dynamically assign accounts to each request in this group.

3.2 Create an Account Tree Based on the Review Matrix

Our goal is to create the hierarchical structure of the accounts. However, it is hard to create an exact account graph because of the large number of accounts. Therefore, we cannot create a tree with hierarchical structure of the accounts based on the account graph. In this section, we heuristically create the a tree for accounts based on the clustering approach on the review matrix.

We start with the review matrix, where the rows correspond to the accounts and the columns correspond to the products available. Suppose we have m accounts and n products giving rise to $m \times n$ matrix where each account has an associated review vector of length n showing which products it has reviewed. It is hard to directly apply the clustering algorithm on this review matrix because of the large number of products. To conquer this challenge, we fist use singular value decomposition on this matrix to reduce the number of columns, e.g., the existing algorithm [3].

Hierarchical Clustering of the Divisive Method. We use the K-Mean Clustering method [9] to divide the set of accounts into clusters. We build a tree of accounts based on hierarchical clustering of the divisive method [9]. For each node in the tree, it includes a set of accounts, and

Algorithm 1: Balanced 2-means($\mathcal{A}_N$)

```
1  Randomly select 2 points as cluster centers (c_1, c_2) in clusters (C_1, C_2);
2  repeat
3      for a_i ∈ A_N do
4          if |dist(a_i, c_1) − dist(a_j, c_2)| ≤ ε then
5              if |C_1| > |C_2| then
6                  C_2 ← C_2 ∪ {a_i};
7              else
8                  C_1 ← C_1 ∪ {a_i};
9      for each c_i do
10         c_i ← (Σ_{a_i ∈ C_i} a_i) / |C_i|, C_i ← ∅;
11 until All clusters do no change;
12 return (C_1, C_2)
```

Algorithm 2: Build($\mathcal{A}_N, i$)

```
1  Create tree node N with the set of accounts A_N;
2  if |A_N| <= τ_0 then
3      return Node N;
4  (A'_1, A'_2) ← 2-means(A_N);
5  L ← Build(A'_1, i + 1) recursively;
6  R ← Build(A_2, i + 1) recursively;
7  left child of N ← L;
8  right child of N ← R;
9  return Node N;
```

we try to make sure that the number of these accounts will not be more than τ_0. As shown in Algorithm 2, if the number of accounts in this node is not larger than τ_0, then we return this node immediately (Line 3), i.e., it is a leaf/terminal node in the tree. Otherwise, it will use 2-means algorithm as shown in Algorithm 1 to separate the accounts into two subsets based on two clusters of these accounts. Then each subset of accounts corresponds to a child of this node (Lines 5–8). Algorithm 1 is similar to the traditional k-means algorithm except that we try to create even clusters without interrupting the property of clusters (Lines 4–8). That is, if the distance to two centers are very close, then the account will be assigned to the cluster with less accounts. Based on these, we build a tree G_{tree} of the accounts through Algorithm 3. In Line 1, we reduce the dimension of the review matrix. In Line 2, we build the account tree based on the reduced review matrix.

3.3 Tree-Based MDPs Based on the Account Tree

Now we have a tree for the accounts. For each node in the account tree, there is a corresponding MDP. Each state in the MDPs of nonterminal nodes in the tree has two kinds of actions (actually, they are corresponding to the branches of this node): assigning the incoming request with the corresponding reviews to the right or left group. Then the MDP of the terminal nodes in the tree take actions to complete each assigned request. The goal of all MDPs is to finish each request with the least cost.

Algorithm 3: Tree Construction via Hierarchical Clustering of the Divisive Method

1 Reduce dimension of $\mathcal{A}$;
2 $G_{tree} \leftarrow$Build$(\mathcal{A}, 0)$;
3 **return** G_{tree};

3.3.1 Nonterminal MDPs

Each nonterminal MDP $\mathcal{M}_i$ on nonterminal node i is similar to the operator MDP (see Section 2.3) except that:

- Each state $s_i = (t_{s_i}, \{dp_{s_i}(a) \mid a \in \mathcal{A}_i\}, \mathbf{REQ}(s_i), \mathcal{REV}(s_i))$ only includes the information for the subset of accounts $\mathcal{A}_i$ in this node i of the account tree.
- Each action α_i in A_i is assigning each child in $\mathcal{M}_i$ to how many reviews in each request in $\mathbf{REQ}_{t_{s_i}}(s_i)$. According to the account tree, $\mathcal{M}_i$ has two children $\mathcal{M}_{i,1}$ and $\mathcal{M}_{i,2}$ corresponding to two children of node i. That is, each action α_i can be represented by a binary $2 \times |\mathbf{REQ}_{t_{s_i}}(s_i)|$ matrix M_i, where $M_i(j, \mathbf{r}) = l$ represents that l reviews in request $\mathbf{r}$ is assigned to child j, and $M_i(1, k) + M_i(2, k) = n$ for each $\mathbf{r} = (p, n, d, cp) \in \mathbf{REQ}_{t_{s_i}}(s_i)$.
- The transition function is defined as: $\mathcal{TR}(s_i, \alpha_i, s_i') = Pr(\mathbf{REQ}_{t_{s_i'}}(s_i')) \times Pr(\{dp_{s_i'}(a), a \in \mathcal{A}_i\} \mid \{dp_{s_i}(a), a \in \mathcal{A}_i\}, \alpha_i)$, where $Pr(\{dp_{s_i'}(a), a \in \mathcal{A}_i\} \mid \{dp_{s_i}(a), a \in \mathcal{A}_i\}, \alpha_i)$ is determined by the terminal MDPs (defined later) and the succeeding nonterminal MDPs reached by α_i, and $Pr(\mathbf{REQ}_{t_{s_i'}}(s_i'))$ is determined by the environment (for the MDP on the root) or the parent of MDPs that are not on the root. Node that, if M_i is on the root of the account tree, then $\mathbf{REQ}_{t_{s_i}}(s_i)$, new appearing requests in s_i, is determined by the environment; and if M_i is not on the root of the account tree, then $\mathbf{REQ}_{t_{s_i}}(s_i)$ is determined by M_i's parent MDP.

That is, each nonterminal MDP includes the global information about its child MDPs, and then makes the decision on assigning which child to which new request.

3.3.2 Terminal MDPs

Each terminal MDP $\mathcal{M}_i$ on terminal node i is similar to the operator MDP (see Section 2.3) except that:

- Each state $s_i = (t_{s_i}, \{dp_{s_i}(a) \mid a \in \mathcal{A}_i\}, \mathbf{REQ}(s_i), \mathcal{REV}(s_i))$ only includes the information for the subset of accounts $\mathcal{A}_i$ in this node i of the account tree.
- Each action α_i, for choosing which accounts to post reviews for which requests, is represented by a binary $|\mathcal{A}_i| \times |\mathbf{REQ}_{t_{s_i}}(s_i)|$ matrix M_i, where $M_i(a, \mathbf{r}) = 1$ represents that request $\mathbf{r}$ is assigned to account a, $\sum_{a \in \mathcal{A}_i} M_i(a, \mathbf{r}) \leq n$ for each $\mathbf{r} \in \mathbf{REQ}_{t_{s_i}}(s_i)$ with n reviews in $\mathbf{r}$, $M_i(a, \mathbf{r}) = 0$ for each $\mathbf{r} \in \mathbf{REQ}_{t_{s_i}}(s_i)$ with $(a, p) \in \mathcal{REV}(s_i)$ and p in $\mathbf{r}$, and $M_i(a, \mathbf{r}) = 0$ for all $a \in \mathcal{A}_i$ with $dp_{s_i}(a) = 1$.
- The transition function, $\mathcal{TR}(s_i, \alpha_i, s_i') = Pr(\mathbf{REQ}_{t_{s_i'}}(s_i')) \times Pr(\{dp_{s_i'}(a), a \in \mathcal{A}_i\} \mid \{dp_{s_i}(a), a \in \mathcal{A}_i\}, \alpha_i)$, has $Pr(\{dp_{s_i'}(a), a \in \mathcal{A}_i\} \mid \{dp_{s_i}(a), a \in \mathcal{A}_i\}, \alpha_i)$ determined by environment after taking acion α_i and $Pr(\mathbf{REQ}_{t_{s_i'}}(s_i'))$ determined by the environment (for the MDP on the root) or the parent of MDPs that are not on the root.

That is, each terminal MDP makes the decision on assigning which accounts to post reviews for which requests.

Let $\mathcal{M}$ be the set of MDPs on the tree. Given an MDP $\mathcal{M}_i$ on the tree, denote $H(i)$ as the the set of MDPs from the root to the parent of $\mathcal{M}_i$, $CH(i)$ as the set of MDPs that are reachable from $\mathcal{M}_i$, $Child(i)$ be the set of children of $\mathcal{M}_i$, $\wr(i) = H(i) \cup CH(i) \cup \{i\}$ be the set MDPs that are directly related to $\mathcal{M}_i$ in the tree, and $\mathcal{M}_{-\wr(i)} = \mathcal{M} \setminus \wr(i))$ be the set MDPs that are are directly related to $\mathcal{M}_i$ in the tree. Let π_i be the policy of MDP $\mathcal{M}_i$ in the space Π_i, and V_i be the expected reward function of MDP $\mathcal{M}_i$. Note that, the policies in the MDPs that are not in $\wr(i)$ also will influence $\mathcal{M}_i$, which are a part of the environment corresponding to $\mathcal{M}_i$, and then influence the expected reward of $\mathcal{M}_i$. We denote $\times_i \pi_i$ as the tree-based policy, which includes a policy π for each MDP $\mathcal{M}_i$ in the tree. Then, we have $V_i(\times_i \pi_i) = \sum_{i' \in Child(_i)} V_{i'}(\times_i \pi_i)$, and $V_0(\times_i \pi_i)$ is the operator's expected reward. We define the best response policy of the succeeding MDPs of i against $(\times_{h \in H(i)} \pi_h, \pi_i)$ as:

$$br(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h) \in \arg \max_{\times_{h \in CH(i)} \pi_h} V_i(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in CH(i)} \pi_h, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h)$$

If $br(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h) = \times_{h \in CH(i)} \pi_h^*$, then

- for each nonterminal MDP $i' \in CH(i)$,

$$br(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in H(i') \setminus (H(i) \cup \{i\})} \pi_h^*, u_{i'}^*, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h, \times_{h \in (\wr(i') \setminus \wr(i))} \pi_h^*) = \times_{h \in CH(i')} \pi_h^*$$

  .

- for each terminal MDP $i' \in CH(i)$,

$$\begin{aligned} &br(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in H(i') \setminus (H(i) \cup \{i\})} \pi_h^*, \pi_{i'}^*, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h, \times_{h \in (\wr(i') \setminus \wr(i))} \pi_h^*) \\ \in &\arg \max_{\pi_{i'} \in \Pi_{i'}} V_{i'}(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in H(i') \setminus (H(i) \cup \{i\})} \pi_h^*, \pi_{i'}, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h, \times_{h \in (\wr(i') \setminus \wr(i))} \pi_h^*). \end{aligned}$$

  That is,

$$\begin{aligned} &V_{i'}(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h, \times_{h \in CH(i)} \pi_h^*) \\ = &\max_{\pi_{i'} \in \Pi_{i'}} V_{i'}(\times_{h \in H(i)} \pi_h, \pi_i, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h, \times_{h \in (CH(i) \setminus \{i'\})} \pi_h^*, \pi_{i'}). \end{aligned}$$

br is recursively defined to make sure that each child's policy is a best policy given other MDPs' policies.

$\times_i \pi_i^*$ is an optimal tree-based policy if, for each $\mathcal{M}_i$,

$$\begin{aligned} &V_i(\times_{h \in H(i)} \pi_h^*, \pi_i^*, \times_{h \in CH(i)} \pi_h^*, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h^*) \\ = &\max_{\pi_i \in \Pi_i} V_i(\times_{h \in H(i)} \pi_h^*, \pi_i, br(\times_{h \in H(i)} \pi_h^*, \pi_i, \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h^*), \times_{h \in \mathcal{M}_{-\wr(i)}} \pi_h^*). \end{aligned}$$

3.4 Soving Tree-Based MDPs

We can see that, the policies of the child MDPs influence the transition function and reward function of their parent MDPs, while the policies of parent MDPs influence the transition function of their child MDPs. We first formulate the tree-based tabular Q-learning algorithm, and then we propose our Tree based Deep Deterministic Policy Gradient (TDDPG) as shown in Algorithm 4 based on DDPG [5] and action transformation [1].

Algorithm 4: Tree-Based Deep Deterministic Policy Gradient

Initialize nonterminal MDP's actor network $\pi^u(s^u \mid \theta^u)$ and critic network $Q^u(s^u, \alpha^u \mid \phi^u)$ and terminal MDP's actor network $\pi^l(s^l \mid \theta^l)$ and critic network $Q^l(s^l, \alpha^l \mid \phi^l)$ with weights θ^u, θ^l, ϕ^u, and ϕ^l randomly;
Initialize target networks with weights $\theta^{u'} \leftarrow \theta^u$, $\phi^{u'} \leftarrow \phi^u$, $\theta^{l'} \leftarrow \theta^l$, and $\phi^{l'} \leftarrow \phi^l$;
Initialize g's dictionary of actions in A;
Initialize replay buffer B;
while *not converge* **do**
    Initialize a random process $\mathcal{N}$ for exploring actions;
    $t \leftarrow 0$;
    for $t < T$ **do**
        $\mathbf{REQ}^t$ is the request set in s_0^t;
        DFS$(t, Node_{root}, \mathbf{REQ}^t, \theta^l, \theta^u, \mathcal{N}^t)$;
        Observe the reward for each MDP $\mathcal{M}_i$, and the next state s_0^{t+1} for the root MDP;
        Store transition set $\{(s_i^t, \overrightarrow{\alpha}^t, r_i^t) \mid \mathcal{M}_i \in \mathcal{M}, i \neq 0\} \cup \{(s_0^t, \overrightarrow{\alpha}^t, r_0^t, s_0^{t+1})\}$;
        Randomly sample a minibatch of N transitions $\{(s_i^j, \overrightarrow{\alpha}^j, r_i^j) \mid \mathcal{M}_i \in \mathcal{M}, i \neq 0\} \cup \{(s_0^j, \overrightarrow{\alpha}^j, r_0^j, s_0^{j+1})\}$;
        $\mathbf{REQ}_{j+1}$ is the request set in s_0^{j+1};
        DFS$(j+1, Node_{root}, \mathbf{REQ}_{j+1}, \theta^{l'}, \theta^{u'}), -)$ without exploration noise;
        Obtain $\overrightarrow{\alpha}^{j+1}$;
        for *Each MDP* $\mathcal{M}_i$ **do**
            Update$(\mathcal{M}_i)$;
        end
        $t \leftarrow t + 1$;
    end
end

3.4.1 Preliminaries

Q-Learning and Deep Q-Network (DQN) [6, 2]. Q-Learning learns an action-value function for a policy: $Q(s, \alpha) = \mathbb{E}[R \mid s^t = a, \alpha^t = a] = \mathbb{E}[R(s, \alpha) + \gamma \mathbb{E}[Q(s', a')]]$. For each transition $(s, \alpha, r = R(s, \alpha), s')$, Q-learning updates according to:

$$Q(s, \alpha) = Q(s, \alpha) + \beta(r + \gamma \max_{\alpha'} Q(s', \alpha') - Q(s, \alpha)),$$

where $\beta \in [0, 1)$ is the learning rate. DQN learns the optimal action-value Q^* corresponding to an optimal policy by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,\alpha,r,s'}[r + \max_{\alpha'} \overline{Q}^*(s', a') - Q^*(s, \alpha \mid \theta)],$$

where θ is the parameter of DQN, and $\overline{Q}$ is a target of Q function where the parameter is periodically updated according to the most recent θ. To stabilize DQN, an experience replay buffer including tuples (s, α, r, s') are used.

Deep Deterministic Policy Gradient (DDPG) [5]. DDPG is an actor-critic approach for extending DQN to handle the continuous action space, which uses a deterministic policy $\pi : S \rightarrow A$ in the continuous action space with the following gradient of the objective $\mathcal{J}(\theta) = \mathbb{E}_{s^t \sim p^\pi}[R]$:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{s^t \sim p^\pi}[\nabla_\alpha Q(s, \alpha \mid \phi) \mid_{s=s^t, \alpha=\pi(s)} \nabla_\theta \pi(s \mid \theta) \mid_{s=s^t}],$$

Algorithm 5: DFS($t, Node_N, \mathbf{REQ}_N, \theta^l, \theta^u, \mathcal{N}^t$))

if *$Node_N$ is a terminal node* **then**
  s^t is a state corresponding to the set of accounts $\mathcal{A}_N$ and request $\mathbf{REQ}_N$;
  Choose action $\overline{\alpha}^t = \pi^l(s^t \mid \theta^l) + \mathcal{N}^t$ according to the exploration noise and the current policy;
  Execute action $\alpha^t \leftarrow g(\overline{\alpha}_i^t)$;
else
  s^t is a state corresponding to the set of accounts $\mathcal{A}_N$ and request $\mathbf{REQ}_N$;
  Choose action $\overline{\alpha}^t = \pi^u(s^t \mid \theta^u) + \mathcal{N}^t$ according to the exploration noise and the current policy ;
  Execute action $\alpha^t \leftarrow g(\overline{\alpha}^t)$;
  for *Each child $Node_i$ in $Node_N$* **do**
    $\mathbf{REQ}_i$ is the set of requests according to α^t;
    DFS($t, Node_i, \mathbf{REQ}_i, \theta^l, \theta^u, \mathcal{N}^t$));

Algorithm 6: Update($\mathcal{M}_i$)

if *$\mathcal{M}_i$ is a terminal MDP* **then**
  $y_i \leftarrow r_i^j + \gamma Q^l(s_i^{j+1}, \overrightarrow{\alpha}^{j+1} \mid \phi^{l'})$);
  Update the terminal MDP's critic by minimizing the loss:
  $\mathcal{L}(\phi^l) : \frac{1}{N}\sum_i (y_i - Q^l(s_i^j, \overrightarrow{\alpha}^j \mid \phi^l))$;
  Update the terminal MDP's actor by using the sampled policy gradient:
  $\mathcal{J}(\theta^l) : \frac{1}{N}\sum_i \nabla_{\alpha_i^j} Q^l(s_i^j, \overrightarrow{\alpha}^j \mid \phi^l) \nabla_{\theta^l} \pi^l(s_i^j \mid \theta^l)$;
  Update the terminal MDP's target network: $\theta^{l'} \leftarrow \tau\theta^l + (1-\tau)\theta^{l'}$,
  $\phi^{l'} \leftarrow \tau\phi^l + (1-\tau)\phi^{l'}$;
else
  Operate similarly to Lines 2–5 for nonterminal MDP's actor π^u with θ^u, critic Q^u with ϕ^u, target network with $\phi^{u'}$;

where $\pi(s \mid \theta)$ is a parameterized actor function specifying the current policy that deterministically maps sates to a specific action, and $Q(s, \alpha \mid \phi)$ is the critic learned by DQN.

Action Transformation [1]. In DDPG, π runs on a continuous space, i.e., $\pi : S \rightarrow R^n$ with $\pi(s) = \overline{\alpha}$. In our problem, the action space A is discrete. Therefore, we need a transformation method [1]: $g : \mathbb{R}^n \rightarrow A$ with $g(\overline{\alpha}) = \arg\min_{\alpha \in A} |\alpha - \overline{\alpha}|_2$. That is, g is a k-nearest-neighbor mapping to a discrete set from a continuous space. If $k = 1$, then g is a nearest neighbor mapping. Then we can have our final policy π' with $\pi'(s) \in \arg\max_{\alpha \in g(\pi(s))} Q(s, \alpha \mid \phi)$.

3.4.2 Tree-Based Tabular Q-learning

Let $Q_i : S_i \times_{\mathcal{M}_j \in \mathcal{M}} A_j$ be the Q function of MDP $\mathcal{M}_i$ with learning rate β_i, where $(s_i, \alpha_i, r_i, s_i')$ is a transition, S_i is the set of states, and A_i is the set of actions. Q_i is a function related to all actions of all MDPs. To update Q_i, we needs to estimate the best action in s_i', and also needs to estimate the best action in the child MDP of $\mathcal{M}_i$, unless $\mathcal{M}_i$ is a terminal MDP. To make the MDPs in the tree update consistently, we develop our tree-based tabular Q-learning. That is, in the root MDP

$\mathcal{M}_0$, with the transition $(s_0, \alpha_0, r_0, s_0')$, we need to estimate all best action in the tree-based MDPs. Let $\overrightarrow{\alpha}' = \times_i \alpha_i'$ be the profile of actions for each MDP $\mathcal{M}_i$ in the state s_i' reached by $\overrightarrow{a}'$ starting from the root with state s_0'. That is, a_0' is taken at state s_0'. After taken action α_0' at the root, the root's one child MDP reaches a new state s_1', then s_1' will take the action α_1'. This procedure will continue until the terminal MDP is reached. Similar to the definition of the best response policy, we define

$$br_\alpha(\times_{h\in H(i)}\alpha_h, \alpha_i, \times_{h\in\mathcal{M}_{-\lambda(i)}}\alpha_h) = \times_{h\in CH(i)}\alpha_h^*,$$

where, for each $i' \in Ch(i)$, $\overrightarrow{\alpha_{i'}} = (\times_{h\in H(i)}\alpha_h, \alpha_i, \times_{h\in H(i')\setminus(H(i)\cup\{i\})}\alpha_h, \alpha_{i'}, \times_{h\in\mathcal{M}_{-\lambda(i)}}\alpha_h, \times_{h\in(\lambda(i')\setminus\lambda(i))}\alpha_h^*)$,

$$Q_{i'}((s_i', \times_{h\in H(i)}\alpha_h, \alpha_i, \times_{h\in\mathcal{M}_{-\lambda(i)}}\alpha_h, \times_{h\in CH(i)}\alpha_h^*) = \max_{\alpha_{i'}} Q_{i'}(s_i', \overrightarrow{\alpha_{i'}}, br_\alpha(\overrightarrow{\alpha_{i'}})),$$

and $br_\alpha(\overrightarrow{\alpha_{i'}}^*) = \times_{h\in CH(i')}\alpha_h^*$. $\overrightarrow{\alpha}'^*$ is a best action profile if

$$Q_0((s_0', \overrightarrow{\alpha}'^*) = \max_{\alpha_0'} Q_0(s_0', \alpha_0', br_\alpha(\alpha_0')).$$

Then, for each MDP, we can update the Q function according to:

$$Q_i(s_i, \overrightarrow{\alpha}) = Q_i(s_i, \overrightarrow{\alpha}) + \beta(r + \gamma Q_i(s_i', \overrightarrow{\alpha}'^*) - Q_i(s_i, \overrightarrow{\alpha}))$$

.

3.4.3 Tree-Based DDPG

It is hard to directly run the tree-based tabular Q-learning due to the large action space. In this section, we use neural network to approximate the Q function, and use continuous action space to represent the large action space. Then we develop our TDDPG as shown in Algorithm 4, where MDPs on nonterminal nodes use the same actor and critic networks, and MDPs on terminal nodes uses the same actor and critic networks because we can identify each MDP by its state s_i. That is, there are two actor-critic threads. In addition, we have to handle the following challenges.

Tree Structure. The tree structure of our problem leads to the following two challenges: 1) tree-based MDPs: the MDPs in different levels of the tree influence each other, and 2) delay results: in each state, after taking an action, we cannot obtain the reward and transit to the next state immediately because the property of tree-based MDPs and the tree as a whole interacts with the environment. To overcome the challenge of tree-based MDPs, we use a Depth-First-Search (DFS) manner to generate each MDP's action as shown in Algorithm 5. For the terminal MDP (the MDP on the terminal node of the tree), it receives the request from its parent, and then reaches its new state s^t at Line 2. Then the action are executed. For the nonterminal MDP (the MDP on the nonterminal node of the tree), it receives the request from its parent, and then reaches its new state s^t at Line 6. Then the action are executed, and the MDP will traverse to its children at Line 11 while passing its action information to its children. To handle the challenge of the delay results, we observe the reward for each MDP and the next state for the root and store the transition after traversing each MDP's action at Lines 11–12 in Algorithm 4. Note that we store the transition set including a transition for each MDP at Line 12 to make the update stable. After that, we sample a minibatch to update parameters at Lines 13–18, which traverses each MDP to estimate the action first. Note that in tree-based tabular Q-learning, we need to enumerate all action combinations to find the best estimated action profile, which is hard to do that. Therefore, we only use the target actor $\pi(s \mid \theta')$ to find the best estimated action for each MDP at Lines 3 and 7 of Algorithm 5 without exploration.

Action Representation. In our problem, the number of requests dynamically changes in different states, then the number of actions will change as well. If each action is corresponding to an output of the network, then we need a network for each state, which is impossible because we cannot enumerate all states. To conquer this challenge, our solution is that, in the terminal MDP, the output of the network is a $|\mathcal{A}_i| \times |\mathbf{REQ}_{max}|$ vector, where $\mathbf{REQ}_{max}$ is the maximum request set received from the environment at a time step. Each action is binary vector, and the value of the output is in $[0, 1]$ in the network. And then, the in the nonterminal MDP, the output of the network is a $|\mathbf{REQ}_{max}|$ vector, where each element in [0,1] represents the portion of reviews in each request is assigned the left group while the less is assigned to the right group. Then, we first receive a real-valued action at Line 3 and then we execute a real action in the environment at Line 4 in Algorithm 5. Here, g is a nearest neighbor mapping.

Compare to directly apply DDPG to the original operator MDP defined in Section 2.3, our TDDPG increases the dimension for the input while reducing the dimension for the output, which will increase the speed to train the network.

3.4.4 Theoretical Properties

In this section, we will show that the optimal tree-based policy is equivalent to the optimal policy in the original operator's MDP defined in Section 2.3, and there is always an optimal tree-based policy. Finally, we will show that our algorithm can converge to an optimal tree-based policy.

Note that $\times_i \pi_i$ is a tree-based policy for each MDP with that $V_0(\times_i \pi_i)$ is the operator's expected reward, and π is an operator's policy in the original MDP defined in Section 2.3.

Lemma 3.0.1. *For each tree-based policy $\times_i \pi_i$, there is an operator's policy π, such that $V_0(\times_i \pi_i) = V(\pi)$, and vice versa.*

Proof. Firstly, the state space for the root MDP in the tree is the same as the original state space. Second, for each state s in the root MDP, under $\times_i \pi_i$, the terminal MDP will finally assign accounts to post reviews to finish requests. Then, if the policy π, under the same state s, assign the same accounts to post the reviews for the same requests at state s, then we will have $V_0(\times_i \pi_i) = V(\pi)$. Similarly, for each state s in the original MDP, π will assign accounts to post reviews for some requests. Then, the tree-based policy $\times_i \pi_i$, for each same state s in the rule, makes sure that the same account posts reviews for the same requests under π. To do that, the nonterminal MDPs need to assign the corresponding number of reviews of each request to the corresponding group of accounts, and then the terminal MDPs assign the corresponding accounts to the corresponding requests as the same assignment under π. Therefore, $V_0(\times_i \pi_i) = V(\pi)$. □

Theorem 3.1. *If $\times_i \pi_i^*$ is the optimal tree-based policy, and π^* is the operator's optimal policy in the original MDP, $V_0(\times_i \pi_i^*) = V(\pi^*)$.*

Proof. This is an direct result from Lemma 3.0.1. □

Theorem 3.2. *There is at least one optimal tree-based policy.*

Proof. The operator's MDP is a finite MDP, then there is always an optimal policy π^* [7]. By Lemma 3.0.1 and Theorem 3.1, there is at least one optimal tree-based policy $\times_i \pi_i^*$. □

Convergence of Tree-Based Q-Learning. In each MDP $\mathcal{M}_i$, s_i is the result of the sequence of actions in MDPs of $H(i)$. Then, $Q_i(s_i, \overrightarrow{\alpha})$ can be represented by $Q_i(s_0, \overrightarrow{\alpha})$. For simplification,

we use $Q_i(s, \overrightarrow{\alpha})$, where $s \in S$ is the state of the root MDP and the global sate of the tree-based MDPs, with the following update rule:

$$Q_i(s, \overrightarrow{\alpha}) = Q_i(s, \overrightarrow{\alpha}) + \beta(r + \gamma Q_i(s', \overrightarrow{\alpha}'^*) - Q_i(s, \overrightarrow{\alpha}))$$

.

Here, we follow the result for convergence of Nash Q-learning [4] to show the convergence of our algorithm.

Assumption 2. *Every state $s \in S$, and action A_i for each MDP $\mathcal{M}_i$, are visited infinitely often.*

Assumption 3. *For all $s, t, \overrightarrow{\alpha}$, the learning rate β^t satisfies:*

1. $0 \leq \beta^t(s, \overrightarrow{\alpha}) < 1$.
2. $\sum_{t=0}^{\infty} \beta^t(s, \overrightarrow{\alpha}) = \infty, \sum_{t=0}^{\infty} [\beta^t(s, \overrightarrow{\alpha})]^2 < \infty$, *which uniformly hold with probability 1.*
3. $\beta^t(s, \overrightarrow{\alpha}) = 0$ *if* $(s, \overrightarrow{\alpha}) \neq (s^t, \overrightarrow{\alpha}^t)$, *i.e.,only updating the Q-function element for the current state s^t and action profile $\overrightarrow{\alpha}^t$.*

Firstly, we depend the following Lemma for the convergence of a Q-learning algorithm updated by a pseudo-contraction operator. Let the space of all Q function be $\mathbb{Q}$.

Lemma 3.2.1. (Szepesvári and Littman [8], Corollary 5) *If Assumption 3 holds and the mapping (pseudo-contractor operator) $\mathcal{C}^t : \mathbb{Q} \to \mathbb{Q}$ satisfies: there exists a number $0 < \gamma < 1$ and a sequence $\lambda^t \geq 0$ converging to zero with probability 1 such that $||\mathcal{C}^t Q - \mathcal{C}^t Q^*|| \leq ||Q - Q^*|| + \lambda^t$ for each $Q \in \mathbb{Q}$ and $Q^* = E[\mathcal{C}^t Q^*]$, then the iteration defined by*

$$Q^{t+1} = (1 - \beta^t)Q^t + \beta^t[\mathcal{C}^t Q^t] \tag{2}$$

converges to Q^* with probability 1.

For our tree-based MDPs, we define the operator $\mathcal{C}^t$ as follows.

Definition 3.3. *Let $Q = \times_i Q_i$, where $Q_i \in \mathbb{Q}^i$ for each MDP $\mathcal{M}_i \in \mathbb{M}$, and $\mathbb{Q} = \times_i \mathbb{Q}_i$. $\mathcal{C}^t : \mathbb{Q} \to \mathbb{Q}$ is a mapping on the complete metrics space $\mathbb{Q}$ into $\mathbb{Q}$, $\mathcal{C}^t Q = \times_i \mathcal{C}^t Q_i$, where, for each MDP $\mathcal{M}_i$,*

$$\mathcal{C}^t Q_i(s, \overrightarrow{\alpha}) = r_i^t(s, \overrightarrow{\alpha}) + \gamma Q_i(s', \overrightarrow{\alpha}'^*), \tag{3}$$

s' is a state at time $t+1$, and $\overrightarrow{\alpha}'^*$ is a best action profile at s'.

Now we prove that $E[\mathcal{C}^t Q^*] = Q^*$.

Lemma 3.2.2. *For the tree-based MDPs, $E[\mathcal{C}^t Q^*] = Q^*$, where $Q^* = \times_i Q_i^*$.*

Proof. For any $s \in S$, $\overrightarrow{\alpha} \in \overrightarrow{A}$, $\mathcal{M}_i \in \mathcal{M}$, a convergence point Q^* in the tree-based MDPs satisfies:

$$Q_i^*(s, \overrightarrow{\alpha}) = r_i(s, \overrightarrow{\alpha}) + \gamma \sum_{s' \in S} \mathcal{TR}(s, \overrightarrow{\alpha}, s') Q_i^*(s', \overrightarrow{\alpha}'^*),$$

$\overrightarrow{\alpha}'^*$ is a best action profile at state s'. Otherwise, the Q-function will change at the next step of update, and then Q^* is not a convergence point. Then, we have

$$Q_i^*(s, \overrightarrow{\alpha}) = \sum_{s' \in S} \mathcal{TR}(s, \overrightarrow{\alpha}, s')(r_i(s, \overrightarrow{\alpha}) + \gamma Q_i^*(s', \overrightarrow{\alpha}'^*)) = E[\mathcal{C}^t Q_i^*(s, \overrightarrow{\alpha})].$$

That is, $E[\mathcal{C}^t Q^*] = Q^*$ □

Now we show that $\mathcal{C}^t$ is a contraction operator.

Assumption 4. *The is a best action profile for each state s, and the corresponding payoff of each MDP is used to update the Q-functions.*

For $Q, \widehat{Q} \in \mathbb{Q}$, we have the following definition [4]:

$$||Q - \widehat{Q}|| = \max_j \max_s \max_{\overrightarrow{\alpha}} |Q_j(s, \overrightarrow{\alpha}) - \widehat{Q}_j(s, \overrightarrow{\alpha})|$$

Then, we can conclude that $\mathcal{C}^t$ is a contraction operator.

Lemma 3.2.3. $||\mathcal{C}^t Q - \mathcal{C}^t \widehat{Q}|| \leq \gamma ||Q - \widehat{Q}||$ *for all* $Q, \widehat{Q} \in \mathbb{Q}$.

Proof. Suppose $\overrightarrow{\alpha}^*$ and $\widehat{\overrightarrow{\alpha}^*}$ are the best action profile of Q and $\widehat{Q}$ at state s. If $Q(s, \overrightarrow{\alpha}^*) \geq \widehat{Q}(s, \widehat{\overrightarrow{\alpha}^*})$, then

$$Q(s, \overrightarrow{\alpha}^*) - \widehat{Q}(s, \widehat{\overrightarrow{\alpha}^*}) \leq Q(s, \overrightarrow{\alpha}^*) - \widehat{Q}(s, \overrightarrow{\alpha}^*) \leq \max_{\overrightarrow{\alpha}} |Q(s, \overrightarrow{\alpha}) - \widehat{Q}(s, \overrightarrow{\alpha})|.$$

Similarly, if $Q(s, \overrightarrow{\alpha}^*) \leq \widehat{Q}(s, \widehat{\overrightarrow{\alpha}^*})$, then

$$\widehat{Q}(s, \widehat{\overrightarrow{\alpha}^*}) - Q(s, \overrightarrow{\alpha}^*) \leq \widehat{Q}(s, \widehat{\overrightarrow{\alpha}^*}) - Q(s, \widehat{\overrightarrow{\alpha}^*}) \leq \max_{\overrightarrow{\alpha}} |Q(s, \overrightarrow{\alpha}) - \widehat{Q}(s, \overrightarrow{\alpha})|.$$

Therefore, we have:

$$||\mathcal{C}^t Q - \mathcal{C}^t \widehat{Q}|| = \max_j \max_s |\gamma Q_j(s, \overrightarrow{\alpha}^*) - \gamma \widehat{Q}_j((s, \widehat{\overrightarrow{\alpha}^*}))| \tag{4}$$

$$\leq \gamma \max_j \max_s \max_{\overrightarrow{\alpha}} |Q_j(s, \overrightarrow{\alpha}) - \widehat{Q}_j(s, \overrightarrow{\alpha})| \tag{5}$$

$$= \gamma ||Q - \widehat{Q}|| \tag{6}$$

□

Now we can present our main result.

Theorem 3.3. *Under Assumption 2–4, the sequence $Q^t = \times_i Q_i^t$, is updated by*

$$Q_i(s, \overrightarrow{\alpha}) = (1 - \beta^t) Q_i(s, \overrightarrow{\alpha}) + \beta^t (r + \gamma Q_i(s', \overrightarrow{\alpha}'^*))$$

converges to a fixed value $Q^* = \times_i Q_i^*$.

Proof. By Lemma 3.2.3, $\mathcal{C}^t$ is a contraction operator. By Lemma 3.2.2, $E[\mathcal{C}^t Q^*] = Q^*$. Then, by Lemma 3.2.1, the process $Q^{t+1} = (1 - \beta^t) Q^t + \beta^t [\mathcal{C}^t Q^t])$ converges to Q^*. □

4 Experiments

References

[1] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[2] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[3] Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, and Mark Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific computing*, 33(5):2580–2594, 2011.

[4] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[8] Csaba Szepesvári and Michael L Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural computation*, 11(8):2017–2060, 1999.

[9] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.