A Study on Reinforcement Learning Algorithms for Machine Learning

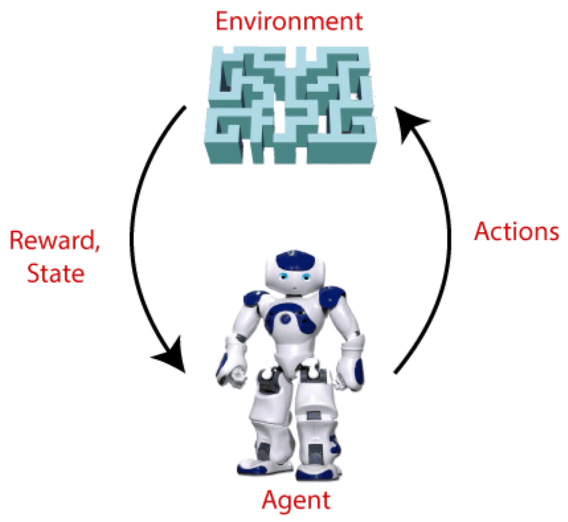
Dr.R.Basheer Mohamed,
Associate Professor, Department of CSE, St.Peter's Engineering College, Hyderabad, Telangana

Abstract: Machine Learning is an indispensable part of Artificial Intelligence. It is the investigation of projects that makes computer to express like humans. Machine learning has come into existence as an important innovation with its adequate number of uses. Reinforcement Learning is one of the major applications of Machine Learning that enables machines and software agents to work explicitly and also resolve the conduct within a definite situation to maximize its performance. Due to the aspect of self-improving, web-based learning and less programming effort Reinforcement Learning becomes an intelligent agent in core technologies. With the advancement of more robust and efficient algorithms, there is still a requirement for more work to be done. Thus, the main aim of this study is to provide the review of Reinforcement Learning and its applications by utilizing various algorithms from Machine learning perspective.

Keywords: Agent, Environment, Action, State, Reward, Policy, Value, Q-value

REINFORCEMENT LEARNING

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty. In Reinforcement Learning, the agent learns automatically using feedbacks without any labelled data, unlike supervised learning. Since there is no labelled data, so the agent is bound to learn by its experience only. RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc. The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that *"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."* How a Robotic dog learns the movement of his arms is an example of Reinforcement learning. It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback. The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment. The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



KEY FEATURES OF REINFORCEMENT LEARNING

In RL, the agent is not instructed about the environment and what actions need to be taken. It is based on the hit and trial process. The agent takes the next action and changes states according to the feedback of the previous action. The agent may get a delayed reward. The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

APPROACHES TO IMPLEMENT REINFORCEMENT LEARNING

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. Value-based:
   The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π.
2. Policy-based:
   Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the

future reward. The policy-based approach has mainly two types of policy:

- Deterministic: The same action is produced by the policy (π) at any state.
- Stochastic: In this policy, probability determines the produced action.

3. Model-based: In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

ELEMENTS OF REINFORCEMENT LEARNING

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment

1) Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:
For deterministic policy:$a=\pi(s)$
For stochastic policy: $\pi(a \mid s) = P[A_t = a \mid S_t = s]$

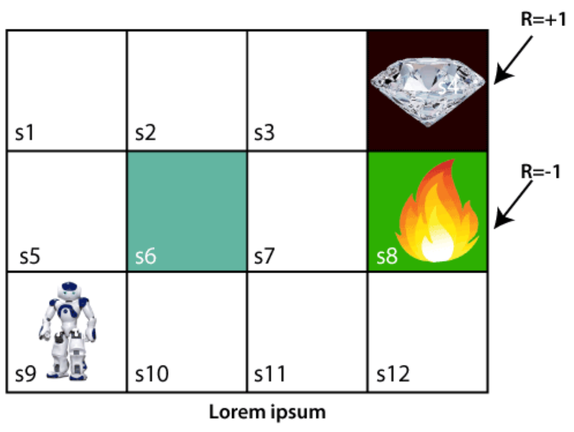
2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the immediate signal for each good and bad action, whereas a value function specifies the good state and action for the future. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

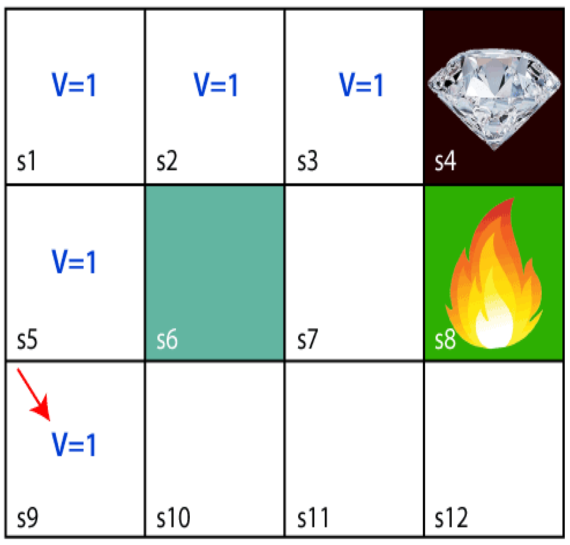
4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.
The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems with the help of the model are termed as the model-based approach. Comparatively, an approach without using a model is called a model-free approach.

WORKING OF REINFORCEMENT LEARNING

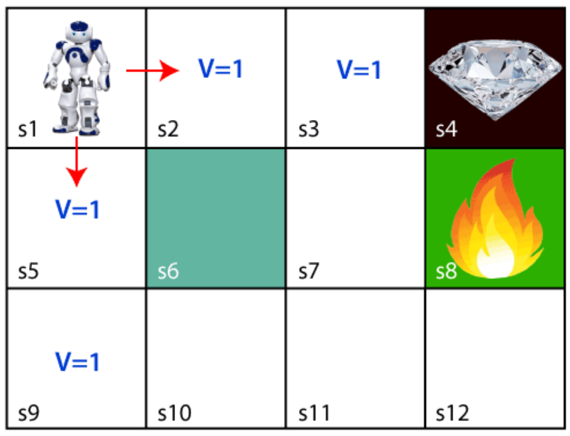
To understand the working process of the RL, we need to consider two main things:
Environment: It can be anything such as a room, maze, football ground, etc.
Agent: An intelligent agent such as AI robot.
Let's take an example of a maze environment that the agent needs to explore. Consider the below image:



In the above image, the agent is at the very first block of the maze. The maze is consisting of an S_6 block, which is a wall, S_8 a fire pit, and S_4 a diamond block. The agent cannot cross the S_6 block, as it is a solid wall. If the agent reaches the S_4 block, then get the +1 reward; if it reaches the fire pit, then gets -1 reward point. It can take four actions: move up, move down, move left, and move right. The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path S9-S5-S1-S2-S3, so he will get the +1-reward point. The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step.
Consider the below step:



Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides?
Consider the below diagram:



It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the Bellman equation, which is the main concept behind reinforcement learning.

The Bellman Equation

The Bellman equation was introduced by the Mathematician Richard Ernest Bellman in the year 1953, and hence it is called as a Bellman equation. It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma "γ."

The Bellman equation can be written as:

$$V(s) = \max [R(s,a) + \gamma V(s`)]$$

Where,

V(s)= value calculated at a particular point.

R(s,a) = Reward at a particular state s by performing an action.

γ = Discount factor

V(s`) = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For 1st block:

V(s3) = max [R(s,a) + γV(s`)], here V(s')= 0 because there is no further state to move.

V(s3)= max[R(s,a)]=> V(s3)= max[1]=> V(s3)= 1.

For 2nd block:

V(s2) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 1, and R(s, a)= 0, because there is no reward at this state.

V(s2)= max[0.9(1)]=> V(s)= max[0.9]=> V(s2) =0.9

For 3rd block:

V(s1) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.9, and R(s, a)= 0, because there is no reward at this state also.

V(s1)= max[0.9(0.9)]=> V(s3)= max[0.81]=> V(s1) =0.81

For 4th block:

V(s5) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.81, and R(s, a)= 0, because there is no reward at this state also.

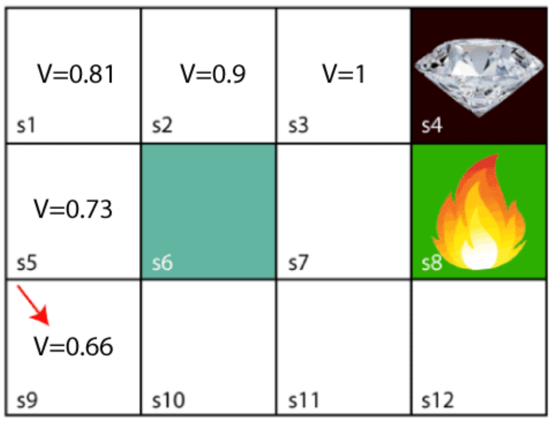
V(s5)= max[0.9(0.81)]=> V(s5)= max[0.81]=> V(s5) =0.73

For 5th block:

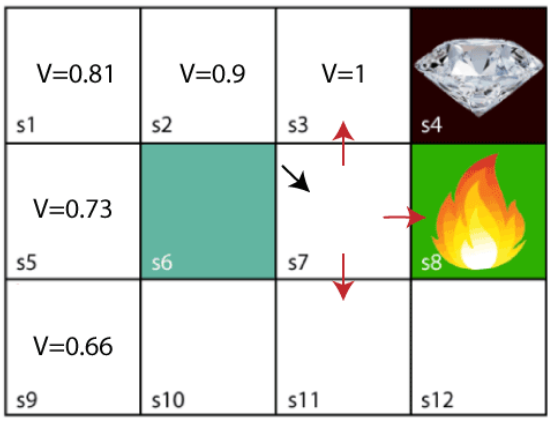
V(s9) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.73, and R(s, a)= 0, because there is no reward at this state also.

V(s9)= max[0.9(0.73)]=> V(s4)= max[0.81]=> V(s4) =0.66

Consider the below image:

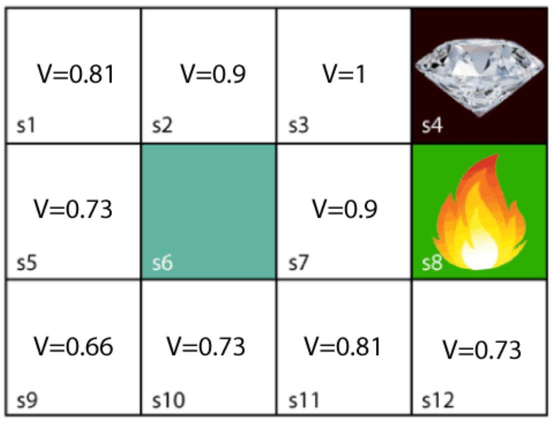


Now, we will move further to the 6th block, and here agent may change the route because it always tries to find the optimal path. So now, let's consider from the block next to the fire pit.



Now, the agent has three options to move; if he moves to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula.

Consider the below image:



TYPES OF REINFORCEMENT LEARNING

There are mainly two types of reinforcement learning, which are:

- Positive Reinforcement
- Negative Reinforcement

Positive Reinforcement:
The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.
This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

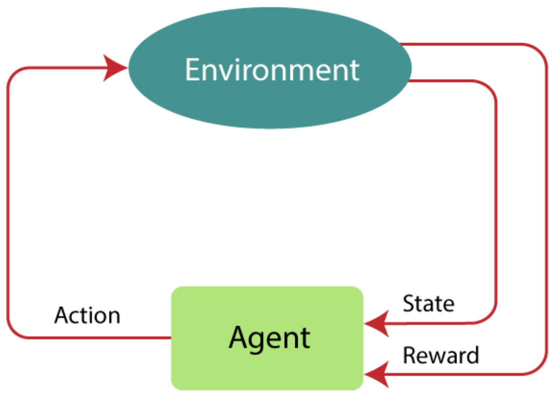
Negative Reinforcement:
The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.
It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior

Representing the agent state
We can represent the agent state using the Markov State that contains all the required information from the history. The State St is Markov state if it follows the given condition:

$P[S_t+1 \mid S_t] = P[S_t+1 \mid S_1,\ldots\ldots, S_t]$

The Markov state follows the Markov property, which says that the future is independent of the past and can only be defined with the present. The RL works on fully observable environments, where the agent can observe the environment and act for the new state. The complete process is known as Markov Decision process, which is explained below:

Markov Decision Process
Markov Decision Process or MDP, is used to formalize the reinforcement learning problems. If the environment is completely observable, then its dynamic can be modeled as a Markov Process. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.



MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.
MDP contains a tuple of four elements (S, A, P_a, R_a):

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S', due to action a.
- Probability P_a.

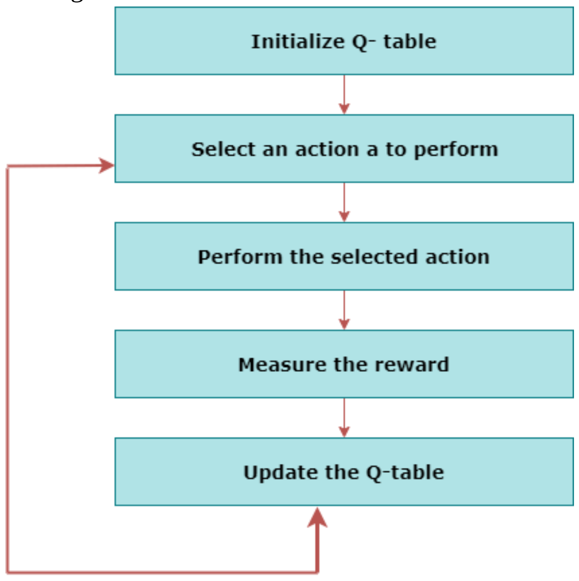
MDP uses Markov property, and to better understand the MDP, we need to learn about it.

Markov Property:
It says that *"If the agent is present in the current state S1, performs an action a1 and move to the state s2, then the state transition from s1 to s2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."*
Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a Chess game, the players only focus on the current state and do not need to remember past actions or states.

Finite MDP:
A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

Markov Process:
Markov Process is a memoryless process with a sequence of random states S_1, S_2, ..., S_t that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and transition function P. These two components (S and P) can define the dynamics of the system.
Reinforcement Learning Algorithms
Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

Q-Learning:
Q-learning is an Off policy RL algorithm, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions. It learns the value function Q (S, a), which means how good to take action "a" at a particular state "s."
The below flowchart explains the working of Q-learning:



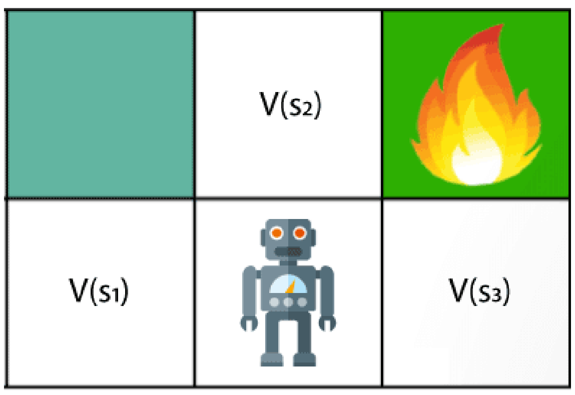
○ State Action Reward State action (SARSA):
SARSA stands for State Action Reward State action, which is an on-policy temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy. The goal of SARSA is to calculate the Q π (s, a) for the selected current policy π and all pairs of (s-a). The main difference between Q-learning and SARSA algorithms is that unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table. In SARSA, new action and reward are selected using the same policy, which has determined the original action. The SARSA is named because it uses the quintuple Q (s, a, r, s', a'). Where, s: originalstate, a:Original action, r:reward observed while following the states, s' and a': New state, action pair.

Deep Q Neural Network (DQN):
As the name suggests, DQN is a Q-learning using Neural networks. For a big state space environment, it will be a challenging and complex task to define and update a Q-table. To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.
Now, we will expand the Q-learning.

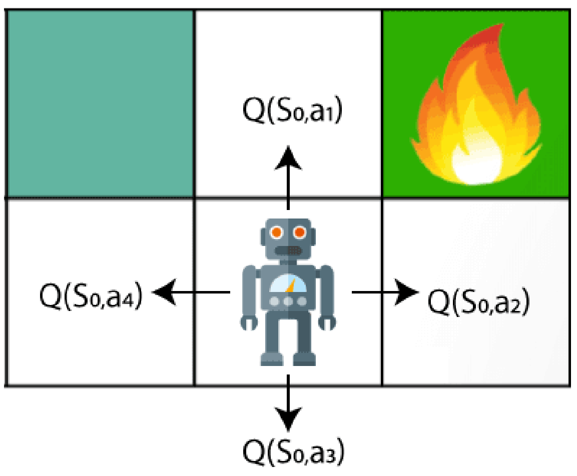
Q-Learning Explanation:
Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation. The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances. It is an off-policy RL that attempts to find the best action to take at a current state. The goal of the agent in Q-learning is to maximize the value of Q. The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

$$V(s) = \max [R(s,a) + \gamma \textstyle\sum_{s'} P(s,a,s')V(s')]$$

In the equation, we have various components, including reward, discount factor (γ), probability, and end states s'. But there is no any Q-value is given so first consider the below image:



In the above image, we can see there is an agent who has three values options, $V(s_1)$, $V(s_2)$, $V(s_3)$. As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up, Left, or Right), so he needs to decide where to go for the optimal path. Here agent will take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value. Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e., Q(s, a). Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.
To perform any action, the agent will get a reward R(s, a), and also he will end up on a certain state, so the Q -value equation will be:

$$Q(S, a) = R(s, a) + \gamma \textstyle\sum_{s'} P(s, a, s')V(s')$$

Hence, we can say that, *V(s) = max [Q(s, a)]*

$$Q(S, a) = R(s, a) + \gamma \textstyle\sum_{s'} (P(s, a, s') \max Q(s', a'))$$

The above formula is used to estimate the Q-values in Q-Learning.
'Q' in Q-learning
The Q stands for quality in Q-learning, which means it specifies the quality of an action taken by the agent.
Q-table:
A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., [s, a], and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table. The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

REFERENCES

[1] [Achterberg et al., 2005] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. Operations Research Letters, 33(1):42–54, Jan 2005.
[2] [Alvarez et al., 2014a] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-andbound. In IN ECML, 2014.
[3] [Alvarez et al., 2014b] Marcos Alvarez, Alejandro, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-andbound. 2014.
[4] [Applegate et al., 1995] D Applegate, R Bixby, V Chvatal, ´ and W Cook. Finding cuts in the tsp. DIMACS Technical Report 95-05, Mar 1995.
[5] [Balcan et al., 2018] MF Balcan, T Dick, T Sandholm, and E Vitercik. Learning to branch. In

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, July 2018. ACM.

[6] [Benichou et al., 1971] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. Mathematical Programming, 1:76–94, Dec 1971.

[7] [Etheve et al., 2020] Marc Etheve, Zacharie Ales, C ` ome Bis- ˆ suel, Olivier Juan, and Safia Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 176–185. Springer, 2020.

[8] [Gasse et al., 2019] Maxime Gasse, Didier Chetelat, Nicola ´ Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. arXiv preprint arXiv:1906.01629, 2019.

[9] [Gupta et al., 2020] Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. arXiv preprint arXiv:2006.15212, 2020.

[10] [Huang et al., 2021] Lingying Huang, Xiaomeng Chen, Wei Huo, Jiazheng Wang, Fan Zhang, Bo Bai, and Ling Shi. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. arXiv preprint arXiv:2111.06257, 2021.

[11] [Khalil et al., 2016] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016.

[12] [NeurIPS 2021 Competition, 2021] NeurIPS 2021 Competition. Machine learning for combinatorial optimization, https://www.ecole.ai/2021/ml4co-competition/,, 2021.

[13] [Patel and Chinneck, 2007] Jagat Patel and John W Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. Mathematical Programming, 110(3):445–474, 2007.

[14] [Prouvost et al., 2020] Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chetelat, and Andrea Lodi. Ecole: A gym-like library for ´ machine learning in combinatorial optimization solvers. arXiv preprint arXiv:2011.06069, 2020.

[15] [Sun et al., 2020] Haoran Sun, Wenbo Chen, Hui Li, and Le Song. Improving learning to branch via reinforcement learning. 2020.

[16] [Van Hasselt et al., 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016