Applied Computer Science 2026, vol. 22, no. 1, pp. 47–62
https://doi.org/10.35784/acs_8031

Submitted: 2025-07-02 | Revised: 2025-09-30 | Accepted: 2025-10-17

Keywords: Q-learning, machine learning, optimisation, assignment problem

Wojciech MISZTAL [1], *Sybilla NAZAREWICZ* [2*]
[1] University of Life Sciences in Lublin, Poland, wojciech.misztal@up.lublin.pl, sybilla.nazarewicz@up.lublin.pl
[*] Corresponding author: sybilla.nazarewicz@up.lublin.pl

Reinforcement learning for solving optimization problems: Opportunities and limitations on the example of the assignment problem

Abstract

The application of reinforcement learning techniques to optimization problems has gained increasing attention due to their adaptability, generalization potential, and capacity to handle complex decision-making processes. This study explores the opportunities and limitations of Q-learning, in the context of the classical Assignment Problem, which plays an important role in transportation logistics and resource allocation scenarios. Four variants of the algorithm were developed and evaluated: a basic version, a version incorporating min-max normalization of cost values, a long-term profitability strategy, and a backward optimization approach. For each of the algorithms, the hyperparameters were optimized using the Optuna library and tests were performed on randomly generated cost matrices of varying dimensions (5, 10, 50, 100, and 200). The quality of the solutions was evaluated based on degradation relative to the optimal objective function value. The time to generate solutions was also measured. The results indicate significant differences in the capabilities of different algorithm variants. The basic Q-learning version is characterized by limited effectiveness and high variability, particularly for larger problem instances. Normalization improved computational efficiency and reduced variance, but did not lead to substantial improvements in solution quality for more complex cases. In contrast, the long-term profitability variant demonstrated notable improvements in both solution quality and stability, especially for smaller and medium-sized problems. The backward optimization variant yielded the highest overall solution quality.

1. INTRODUCTION

The specific characteristics of transportation operations, in particular their spatial and dynamic nature, mean that logistical efficiency, incurred costs, and service quality depend not only on the resources used, but also on the strategies and approaches used during implementation (Chládek et al., 2018; Pečený et al., 2020; Taran et al., 2024). These operational characteristics make transport logistics a complex system in which various interrelated factors, such as route planning, resource allocation, time constraints, and vehicle capacity, need to be optimized simultaneously. To date, a wide range of problems have been identified, each differing in their application, specificity, importance, and inherent complexity. Among these, a notable subset of problems matches the characteristics of many real-world scenarios that exhibit a high degree of complexity. Representative examples include problems belonging to well-known problem classes such as the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and the Assignment Problem, which are frequently encountered in transportation and logistics systems (Asef-Vaziri et al, 2022; Baller et al., 2020; Boccia et al., 2021; Chen et al., 2024; Rabbani et al., 2019; Shopov & Markova, 2021; Shramenko et al., 2022; Vadseth et al., 2023; Zhu & Hu, 2019). These problems are highly relevant because they involve optimizing the allocation of resources (e.g., vehicles, routes, and personnel) while minimizing cost, time, or distance, all of which are critical factors in improving operational efficiency. Optimization methods, techniques, and tools are essential for identifying feasible and effective solutions to these types of problems. Advanced optimization algorithms can provide decision makers with valuable insights that allow them to make more informed decisions. However, the complexity of certain problems, especially those involving a large number of variables, makes it extremely challenging to develop solutions that are both effective and computationally feasible. This challenge is particularly pronounced when it comes to NP-hard problems, where finding an optimal solution in a reasonable amount of time is computationally impractical for large instances (Boccia et al., 2021; Cárdenas-Montes, 2018; Zhang et al., 2023). Consequently, this highlights the need for innovative approaches that can provide near-optimal solutions in a more efficient manner, even when an exact solution is unattainable.

Artificial Intelligence (AI), in particular Machine Learning (ML) and Deep Learning (DL), has emerged as a promising avenue for solving complex optimization problems in a wide range of domains. Due to their ability to learn from large datasets and adapt to new information, AI methods have shown great potential for improving decision-making processes in transportation logistics and other domains (Małek et al., 2023; Nichols et al., 2019; A. Sharma et al., 2020; N. Sharma et al., 2021; Shinde & Shah, 2018; Surblys et al., 2024; Tarapata et al., 2022). However, despite their successes, machine learning and deep learning techniques are not without their limitations. In certain cases, these approaches may struggle to produce effective results, particularly when the problem at hand involves limited data or highly complex, non-linear relationships that are difficult for these algorithms to model. In addition, the implementation of such approaches can be computationally expensive and time-consuming, and in some applications they may prove difficult to implement effectively (Jarrett et al., 2019). These challenges require careful consideration of which AI techniques are most appropriate for a given problem, and how they can be fine-tuned to achieve the best possible results.

This study attempts to apply the Q-learning machine learning algorithm to solve an optimization problem in the form of the assignment problem. By applying Q-learning, the study aims to explore the capabilities of the algorithm in this specific context, examining how well it can handle the complexity of the problem and whether it can identify near-optimal solutions in a reasonable amount of time. In addition, the study will assess the limitations of Q-learning and the sensitivity of the results to the choice of parameters and implementation techniques. The analysis will also include an exploration of the techniques and parameters that influence the quality of the solutions and the time required to obtain them, providing a comprehensive evaluation of the potential of Q-learning as a tool for solving complex optimization problems in transportation and logistics.

2. METHOD

The effort to identify the capabilities and limitations, as well as the necessary steps to enable the application of reinforcement learning to optimization problems, was framed as an attempt to apply the Q-learning algorithm to the assignment problem. The considered problem is characterized by a high degree of complexity and has practical relevance in numerous real-world applications, such as the assignment of transportation resources to tasks or the scheduling of machine maintenance (Rabbani et al., 2019; Shopov & Markova 2021). The choice of this particular optimization problem was motivated by the intention to assess the quality of the solutions returned by the machine learning algorithm by comparing them with exact solutions. In the case of the assignment problem, it is possible to generate exact solutions even for large problem instances using methods such as the Hungarian algorithm (Kuhn-Munkres method) (Shopov & Markova, 2021).

In general, the assignment problem can be formulated as follows: Given a set of heterogeneous tasks and a set of heterogeneous resources capable of executing them, the objective is to allocate resources to tasks in such a way that the value of the objective function is minimized (or maximized). In the basic version of the problem, the number of tasks is equal to the number of resources, and the goal is to assign each resource to exactly one task while minimizing or maximizing a cost function, which typically represents the cost, time, or distance associated with each possible assignment (Shopov & Markova, 2021).

The mathematical model of the assignment problem takes the following form (Taha, 2017):

$$\min Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

with restrictions:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, 2, \dots, n \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j = 1, 2, \dots, n \tag{3}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{4}$$

where: $c_{ij} \in R_{\geq 0}$ – the cost of assigning resource i to task j, where $i = 1, 2, \dots, n$, $j = 1, 2, \dots, n$ and $n \in N$,
x_{ij} – equals 1 if resource i is assigned to task j, and 0 otherwise.

The application of Q-learning to the assignment problem stems from its inherent ability to model environments where decisions are sequential and their effects unfold over time. Reinforcement learning algorithms, including Q-learning, are designed to learn optimal decision policies through interactions with an environment (Hu, 2023; Miller, 2024).

Q-learning is a reinforcement learning algorithm that attempts to identify the optimal policy for action selection in a finite decision process. This algorithm is model-free, meaning that it does not rely on a model to predict future states and rewards. Instead, it learns directly by interacting with the environment, taking actions to change the state, and updating Q-values based on the rewards or penalties resulting from those actions. The Q-values represent the quality of specific actions depending on the current state and are used to determine the best action to take (Clifton & Laber, 2020). This algorithm has applications in gaming and robotics, as well as in determining optimal treatment strategies (setting a treatment regimen) or automatically adjusting machine and device parameters (Clifton & Laber, 2020). It also shows potential for use in solving optimization problems.

The implementation of the Q-learning algorithm requires the following steps:

1. Define the environment that simulates the task problem (state space, action space, reward function).
   States - representations of the situation (e.g., available resources, task status).
   Actions - possible actions to take (e.g. assign resources to tasks).
   Rewards - assessments of the quality of actions taken (e.g. profit from assigning an employee to a task, or penalty for assigning an employee to more tasks).
2. Implementation of Q-learning algorithm.

The general formula of Q-learning (so-called action values) (Zhu et al., 2024):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right) \tag{5}$$

where: $Q(s_t, a_t)$ – the value of the function Q for state s_t and action a_t,
α – the learning coefficient, controlling the size of the impact of new information on the Q value (0 – the agent does not learn anything, it only uses previously acquired knowledge, 1 – the agent ignores the knowledge it has and constantly explores),
r_t – the reward related to the action at carried out in the state s_t,
γ – the discount factor, which determines the importance of future rewards (0 – the agent is focused only on current short-term rewards, 1 – the agent is focused only on long-term rewards), plays a critical role in shaping the agent's decision-making strategy,
$\max_{a'} Q(s_{t+1}, a')$ - the maximum Q-value in the next state s_{t+1} (assuming the agent selects the best possible action), guides the agent towards achieving the optimal action strategy (policy), this value represents the best expected future reward achievable from the current state, factoring in the agent's future decisions.

An important aspect of Q-learning is maintaining a proper balance between exploration and exploitation. Exploration is essential for identifying activities that provide the opportunity to earn the greatest rewards. Exploitation, on the other hand, allows knowledge to be used for maximum benefit. This balance is typically managed using the ε-greedy policy, where a random action is selected with probability ε and the best known action (i.e., the one with the highest Q-value) is selected with probability 1- ε. This balance is crucial for the agent to both learn from its environment and avoid getting stuck in suboptimal actions. The learning process involves continuous refinement of the agent's policy, with exploration being more frequent initially and exploitation increasing as the agent becomes more confident in its knowledge. Q-learning demonstrates the ability to achieve optimal Q-values with appropriate exploration and decreasing learning rates. The general scheme of the Q-learning algorithm requires the implementation of the following sets of activities (Miller, 2024):

- Initialization - the Q-table is created for all state-action pairs, with all entries initially assigned a value of 0. This represents a neutral starting point, assuming no prior knowledge of the environment, and allows the agent to learn value estimates solely through interaction and experience.
- Action Selection - starting from a random initial position, an action is selected from the action space according to the ε-greedy strategy.
- Q-table Update - the Q-table is updated based on the observed state and the reward resulting from the executed action.
- Iterate and Terminate - the process is repeated for a predefined number of iterations.

In this study, four variants of the Q-learning algorithm are considered, selected on the basis of previous analyses. These include: the basic form of the algorithm (adapted to the specifics of the considered problem), a modification incorporating cost normalization, as well as versions implementing the long-term profitability strategy and backward optimization.

The comparison of the algorithms was performed using a set of test cases consisting of randomly generated cost matrices of dimensions 5, 10, 50, 100, and 200. Five different test cases were generated for each matrix size. For each of these cases, all considered algorithms were tested. The objective function values of the optimal solutions (which serve as a reference for evaluating the quality of the results produced by the considered approaches) were obtained using the linear_sum_assignment function from the scipy.optimize library, which implements an exact algorithm for solving the assignment problem.

Before running the tests, hyperparameter optimization was performed for each variant of the Q-learning algorithm. The parameters subject to optimization included α (learning rate), γ (discount factor), ε (exploration rate), and the number of episodes. The optimization process was performed using the Optuna library (Akiba et al., 2019). This library is an advanced tool designed for hyperparameter optimization in machine learning algorithms. It is effective in automatically identifying the most favorable hyperparameter combinations, which is particularly important in the context of reinforcement learning algorithms, where such parameters critically affect the ability to achieve high-quality solutions (Akiba et al., 2019; Ilosvay & Iaccarino, 2023). Optuna provides the ability to dynamically define the search space, as well as the conditional relationships between hyperparameters. This allows the optimization process to be tailored to the specifics of the problem under consideration. The efficiency of hyperparameter space exploration is achieved by region selection using a sampling algorithm. The computational cost is reduced by an early stopping mechanism that terminates unpromising trials at an early stage (Akiba et al., 2019; Eimer et al., 2023; Shekhar et al., 2021).

An approach was adopted in which hyperparameter optimization was performed separately for each size of the considered test cases. The optimization process was performed within the following parameter bounds $\alpha \in [0.01, 1]$, $\gamma \in [0.5, 0.99]$, $\varepsilon \in [0.01, 1]$, and the number of episodes ranged from 100 to 5000, using variable cost matrices corresponding to each dimension. The goal was to identify a configuration that minimizes the percentage degradation of the objective function value relative to the optimal solution (i.e., the lowest relative error). The calibration procedure involved 30 test iterations (trials) per configuration.

The impact of hyperparameter values on the quality of the returned solutions was assessed using interpretability techniques, specifically permutation importance ranking and SHAP summary plots.

Permutation Importance is a method for evaluating the importance of hyperparameters by randomly permuting their values in the input data set and measuring the resulting impact on model performance. In the context of Q-learning, this approach makes it possible to assess how individual hyperparameters affect the effectiveness of the learning process. Since the algorithm relies on a balance between exploitation and exploration, hyperparameter values can significantly affect the quality of the solutions obtained. This method provides insight into which hyperparameters play the most critical role in ensuring the desired solution quality, which is essential both for optimizing their values and for adapting the structure of the applied machine learning algorithms (Bladen & Cutler, 2024).

SHAP (Shapley Additive Explanations) is a model interpretability method based on Shapley values, which quantify the contribution of each feature to the model output by considering all possible combinations and interactions among features. SHAP Summary Plots provide a visual and quantitative overview of these relationships, allowing one to determine how changes in individual hyperparameter values (such as learning rate, discount factor, exploration rate, and number of episodes) affect the performance of the Q-learning algorithm, providing a deeper understanding of the capabilities and limitations of the approach under study (Loecher, 2024; Salih et al., 2025; Scikit-learn, 2024; Wang et al., 2024).

Both the analysis of hyperparameter importance (permutation importance) and the calculation of SHAP values, on the basis of which SHAP summary plots are generated, use the Random Forest machine learning algorithm. This method is based on the use of decision trees, each of which is trained on a different subset of the training data using bootstrap aggregation. The results of each tree are then combined to produce results with greater stability and accuracy. The Random Forest method reduces the correlation between the trees by randomizing the features used in the construction of each tree. The strategy applied makes this approach resistant to overfitting, capable of reflecting nonlinear relationships between features (hyperparameters) and the outcome, and also shows high efficiency in solving problems involving large datasets (Fumagalli et al., 2023).

The use of the Random Forest model in this context involves training it on a dataset containing different hyperparameters (α,γ,ε and the number of episodes) along with their respective influence on the quality of the solutions produced by the Q-learning algorithm. This approach allows the development of a nonlinear regression model capable of capturing the complex interactions between these hyperparameters and the resulting outputs. Specifically, in this study, the Random Forest Regressor is trained on a dataset containing information about the values of the aforementioned hyperparameters, as well as the percentage degradation of the objective function values of the solutions obtained, compared to the optimal solutions. The importance of each hyperparameter is then evaluated, and the process is repeated 30 times to ensure the stability and robustness of the results. Based on this, a ranking of the hyperparameters is generated, where the features are sorted according to the average importance calculated over the multiple iterations. The trained random forest model was then used to determine the SHAP values. This was done using the TreeExplainer tool from the SHAP package, which is designed for tree-based models.

The study used the SHAP package along with the RandomForestRegressor and permutation_importance functions from the scikit-learn library, a widely recognized and used set of tools in the Python programming language for data analysis and machine learning model development.

The history of the hyperparameter optimization process was used to evaluate the impact of different hyperparameter configurations on the quality of the generated solutions. This analysis allowed the identification of the most effective hyperparameter settings, which were then used in a comparative evaluation of the different algorithm variants. The comparison was performed over the entire set of test cases, with each test scenario repeated ten times to ensure statistical reliability and to minimize the potential impact of random fluctuations on the results. The evaluation of the algorithm variants was based on the objective function values of the solutions produced by each approach. In addition to evaluating the quality of the solutions, the time required to generate each solution was also recorded and analyzed, providing a comprehensive understanding of both the performance and computational efficiency of the algorithms under consideration.

The tests were performed on a laptop equipped with an Intel Core Ultra 5 125U 1.30 GHz processor and 32 GB of RAM.

2.1. The basic version of the Q-learning algorithm

The basic version of the Q-learning algorithm, adapted to the specifics of the assignment problem, assumes the inclusion of a cost matrix. The activity is aimed at selecting the best solution that satisfies the feasibility conditions for the problem under consideration. In this case, the Q-table is designed in such a way that the states represent tasks, the actions represent resources (allocations), and the Q-values correspond to the evaluation of the allocation of a resource to a task. After training the algorithm and obtaining the Q-table, it is necessary to use an additional procedure to determine the final solution (the algorithm does not take into account the specificity of assignments). The final solution is generated based on the trained Q-table by selecting the highest Q-value for each row, iterating from the first to the last. The pseudocode for this version of the procedure is shown in Algorithm 1.

Algorithm 1. Pseudocode of the Q-learning Algorithm Adapted to the Specifics of the Assignment Problem

```
FUNCTION initialize_environment:
  Initialize AssignmentEnv(num_tasks, num_workers, cost_matrix, alpha, gamma, epsilon, max_steps,
epizodes)
  Initialize Q [0...num_tasks - 1, 0...num_workers - 1] to 0
Initialize empty list assignments

FUNCTION choose_action(state):
  available_workers = list of available workers in state
  IF available_workers is empty:
    RETURN -1 (no available workers)
  IF random number < epsilon:
    RETURN random worker from available_workers (exploration)
  ELSE:
    RETURN worker with the highest Q value for state from available_workers (exploitation)

FUNCTION step(action):
  IF not all tasks assigned:
    IF worker is available:
      reward = -cost_matrix[state][action]
      mark worker as unavailable
      state += 1
    ELSE:
      reward = -10
  ELSE:
    RETURN state, reward, done = TRUE
  RETURN state, reward, done = FALSE

FUNCTION train:
  FOR each episode from 1 to num_episodes:
    state = reset()
    done = False
    total_reward = 0
    FOR each step from 1 to max_steps:
      action = choose_action(state)
      IF action == -1:
        done = True
        BREAK
      next_state, reward, done = step(action)
      IF done:
        next_state = state
      Q[state, action] = Q[state, action] + alpha * (reward + gamma * max(Q[next_state]) - Q[state, action])
      total_reward += reward
      state = next_state
      IF done:
        BREAK

FUNCTION get_assignments:
  reset worker_availabilities
  FOR each task in 0...num_tasks - 1:
    available_workers = list of available workers
    IF available_workers is not empty:
      best_worker = worker with the highest Q value for task from available_workers
      assignments.append(task, best_worker)
      worker_availabilities[best_worker] = False
```

The function initialize_environment is responsible for creating the environment for the Q-learning algorithm. In the case of the algorithm designed to solve the assignment problem, the Q-table has dimensions corresponding to the number of tasks (num_tasks) and the number of workers (num_workers), as it stores values for each task-worker state. These values represent the expected future cost of performing a given action in a given state. In this work, we consider cases where these initial values are equal. This function also initializes the cost matrix (cost_matrix), the learning rate (alpha), the discount factor (gamma), the exploration rate (epsilon), the maximum number of steps (max_steps), the number of episodes (num_episodes), and an empty list of assignments (assignments). It is important to note that in this approach, the number of steps is equal to the number of required assignments (i.e., max_steps = num_tasks = num_workers).

The choose_action function serves as an action selection mechanism. For a given state representing the current task, the function chooses an action (assigns a worker). The function implements the assumed exploration-exploitation trade-off strategy, following the ε-greedy policy. If there are no workers left, the function returns -1, signaling the end of the current episode.

The step function handles a single step in the environment (assigning a worker to the current task). It gives a reward if an available worker is assigned (negative assignment cost), or a penalty if an unavailable worker is selected. It also updates the environment state (moves to the next task) and indicates whether all tasks have been assigned (returns done = True).

The train function is responsible for training the agent. The training process is performed over a predefined number of episodes, during which the agent interacts with the environment and learns how to allocate resources to tasks. At the beginning of each episode, the environment is reset (worker availability is restored and the first task is set as the current state). At each step of the episode, the agent chooses an action based on the current state (using the choose_action function) and transitions to the next state. Performing an action yields a reward, represented as the negative cost associated with allocating a resource to a task. The Q values for the current state-action pair are updated according to equation (5). The episode continues until all assignments are completed (until there are no unused resources left).

The get_assignments function is responsible for generating the final assignments after the training phase is complete. The algorithm iterates through the tasks sequentially (starting with the first one) and selects for each task the available resource (worker) with the highest Q-value. During this activity, it selects only available resources. Once an assignment is made, the employee's availability is updated.

2.2. Normalization of values in the cost matrix

The presence of extremely large (or small) values within the cost matrix can cause these values to dominate the action selection process, which is undesirable from the perspective of achieving long-term goals. Normalizing the considered cost values shows the ability to mitigate this problem. It improves the stability of the learning process (as the Q-table values are brought to a more uniform scale), increases the exploration efficiency (especially in the early stages of learning), and accelerates convergence (by promoting more consistent updates of Q-values in response to rewards and penalties). In the considered problem, min-max normalization is applied, which transforms the cost values such that the smallest value in the cost matrix is mapped to 0 and the largest to 1. To achieve this, each element of the matrix is transformed using equation (6).

$$c_{ij}^{norm} = \frac{c_{ij} - c_j}{c^j - c_j} \quad for \quad 1 \le i,j \le n \tag{6}$$

where: $c_j = \min\{c_{ij}: 1 \le i \le n\}$ and $c^j = \max\{c_{ij}: 1 \le i \le n\}$

The modification of the procedure involves calling a function responsible for normalizing the cost matrix (Algorithm 2) at the moment the environment is initialized.

Algorithm 2. Function responsible for the normalization of values in the cost matrix

1	**FUNCTION** normalize_cost_matrix (cost_matrix):
2	min_vals = Calculate minimum values for each column in cost_matrix
3	max_vals = Calculate maximum values for each column in cost_matrix
4	normalized_matrix = (cost_matrix - min_vals) / (max_vals - min_vals)
5	**RETRUN** normalized_matrix

2.3. Long-term profitability and backward optimization

The version of the algorithm using the strategy of long-term profitability is based on the consideration of the entire policy and long-term profitability of orders. The concept of this approach is to optimize the tasks based on the history of actions, with a focus on improving the overall result (long-term profit). This history stores all actions during the whole training process. The actions themselves are selected in the same way as in the basic version of Q-learning (using the exploration and exploitation mechanism). The algorithm is still based on the use of the Q-table (during exploitation the most advantageous assignment is chosen based on it). The main difference is in the way the final solution is generated. The function get_assignmentsis replaced by optimize_assignments whose pseudocode is shown in Algorithm 3.

Algorithm 3. The function responsible for striving to improve the quality of returned solutions (operating in accordance with the long-term profitability strategy)

```
1   FUNCTION optimize_assignments:
2     Initialize best_assignment = None
3     Initialize min_cost = infinity
4     FOR each episode_actions in assignments_history:
5       Initialize task_assignment as empty list
6       Initialize total_cost = 0
7       Initialize available_workers as list of True with size num_workers
8       FOR each (state, action, reward) in episode_actions:
9         IF available_workers[action] is True:
10          Add -reward to total_cost
11          Append (state, action) to task_assignment
12          Set available_workers[action] to False
13          IF total_cost < min_cost:
14            Set min_cost to total_cost
15            Set best_assignment to task_assignment
16    RETURN best_assignment
```

The pseudocode in Algorithm 3 shows the process of optimizing tasks based on their history derived from the agent training phase. The code iterates over the elements of the history record. Each time, the costs for each order are determined and the most advantageous variant is selected.

The backward optimization variant is a modification of the long-term profitability approach. The difference lies in the approach to searching the history of assignments. In the case of long-term profitability, the order in which the history is searched is the same as the order in which it was created. The backward optimization variant, on the other hand, assumes that the history is searched in reverse order. The FOR loop from the 8th line of pseudocode in Algorithm 3 thus iterates over the reverse list *episode_actions*.

3. RESULTS AND DISCUSSION

The Ranking Permutation Importance for the hyperparameters of each algorithm version is shown in Table 1. SHAP summary plots illustrating the impact of each hyperparameter on the quality of the solutions returned are shown in Figures 1-4.

Tab. 1. Hyperparameter impact ranking (permutation importance)

Hyperparameter	Importance			
	Q-learning	Q-learning NMM	Q-learning LTP	Q-learning BO
α	0.593078	0.331367	0.150942	0.249195
γ	0.293569	0.536748	0.199687	0.305251
ε	0.276070	0.381063	1.001838	1.093285
Num. Ep.	0.447528	0.411164	0.298991	0.264460

The most advantageous hyperparameter configurations identified for individual algorithms using the Optuna library are presented in Tables 2-5. The data on the quality of solutions (optimality gap) and the time needed to generate them (the sum of training and usage time) for the algorithm variants using the optimal hyperparameter configurations are shown in the graphs in Figures 5 and 6.

3.1. Q-learning

The ranking of the importance of the hyperparameters (Table 1) shows that the α coefficient has the most significant impact on the solution quality, followed by the number of episodes. The hyperparameters γ and ε have a smaller but still significant effect. Complementing this information with the SHAP plot (Figure 1), it can be observed that certain low values of the parameter α have a significant positive impact on the returned solutions (this applies to instances of small size - Table 2). In the case of the number of episodes, the trend is the opposite (but more consistent). Similarly, the γ and ε coefficients affect the solution, but to a lesser extent.

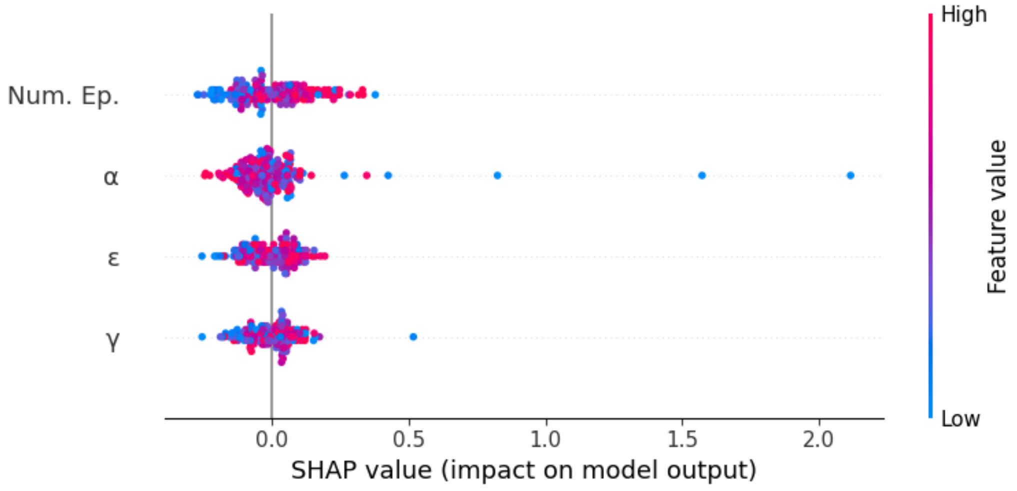

Fig. 1. SHAP Summary Plot for Q-learning algorithm

Tab. 2. Optimal hyperparameter configurations for the Q-learning algorithm

Instance dimension	α	γ	ε	Number of episodes
5	0.216735921	0.518455108	0.650662841	4387
10	0.412923089	0.715504491	0.884111398	1396
25	0.72206396	0.831738886	0.618261325	120
50	0.444388814	0.927633423	0.913247959	2082
100	0.438454581	0.732915014	0.559553977	3973
200	0.909431441	0.804891674	0.978954298	4911

Considering the most advantageous configurations of hyperparameters identified for the Q-learning algorithm (Table 1), it can be observed that in the case of small problem instances (dimension: 5, 10), the values of the learning coefficients (α) and discounting (γ) are relatively low. Therefore, the strategy assuming slower learning and considering a shorter horizon in terms of desired rewards is profitable for the agent. At the same time, the degree of exploration (ε) is quite high, the agent is focused on greater exploration. In the case of medium instances (25, 50, 100), the values of hyperparameters α and γ are higher, which means a change in the orientation of the agent towards faster learning and consideration of long-term benefits. The coefficientεis variable, implying a balance between exploration and exploitation, which is likely to be strongly influenced by the specificity of the test cases considered. Large problem instances require the agent to learn intensively, focus on long-term goals, and explore, as evidenced by high values of all hyperparameters. The number of learning episodes required to complete varies across problem instances, but is high with some exceptions (dimensions 10 and 25).

3.2. Q-learning with normalization min-max

Considering the most advantageous configurations of hyperparameters identified for the Q-learning algorithm (Table 1), it can be observed that in the case of small problem instances (dimension: 5, 10), the values of the learning coefficients (α) and discounting (γ) are relatively low. Therefore, the strategy that assumes slower learning and considers a shorter horizon in terms of desired rewards is profitable for the agent. At the same time, the degree of exploration (ε) is quite high, the agent is focused on greater exploration. In the case of medium instances (25, 50, 100), the values of hyperparameters α and γ are higher, which means a change in the orientation of the agent towards faster learning and consideration of long-term benefits. The coefficient ε is variable, implying a balance between exploration and exploitation, which is likely to be strongly influenced by the specificity of the test cases considered. Large problem instances require the agent to learn intensively, focus on long-term goals, and explore, as evidenced by high values of all hyperparameters. The number of learning episodes required for completion varies across problem instances, but is high with some exceptions (dimensions 10 and 25).

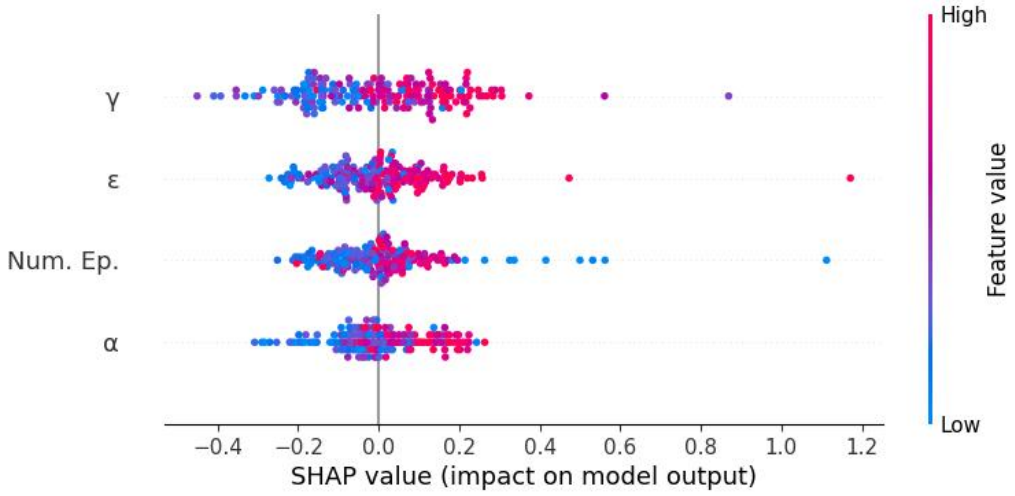

Fig. 2. SHAP Summary Plot for Q-learning algorithm with normalization min-max

Tab. 3. Optimal hyperparameter configurations for the Q-learning algorithm with Normalization min max

Instance dimension	α	γ	ε	Number of episodes
5	0.429508475	0.708271683	0.238358583	134
10	0.516055941	0.659979997	0.846639153	458
25	0.706780548	0.513684641	0.315526634	1166
50	0.077643814	0.915251736	0.90335125	1412
100	0.796441444	0.951788881	0.813607005	2779
200	0.956187044	0.978377178	0.526697065	3568

The most favorable hyperparameter configurations determined for the Q-learning algorithm using the normalized cost matrix (Table 2) indicate that it allows the agent to reach convergence faster, as evidenced by the lower number of episodes required to run. The learning process is more stable. The number of episodes is obviously influenced by the complexity of the problem. The algorithm shows improved performance when optimizing for long-term rewards, as indicated by higher values of the discount factor γ. While the agent still shows variability in the balance between exploration and exploitation, the adopted policy shows increased stability. For small problem instances (dimensions: 5, 10), the values of the learning rate α and the discount factor γ remain moderate, while the exploration rate ε shows high variability-indicating that the agent is able to quickly switch to exploitation or engage in intensive exploration. The number of episodes remains very low, suggesting that normalization has a positive impact on the agent's efficiency. Medium-sized instances (25, 50, 100) require higher values of both α and γ, suggesting that it is advantageous to emphasize intensive learning and a focus on long-term rewards. However, some deviations are observed (for instances of dimension 50, the coefficient α takes a low value, similar to γ for problems of size 25), which may be caused by noise in the data resulting from normalization. The coefficient ε still indicates the need to explore the space of possible solutions. The largest problem instances force the agent to focus on intensive learning and long-term goals, and exploration is not as intensive.

3.3. Q-learning with long-term profitability

The ranking of the influence of the hyperparameters (Table 1) shows a strong influence of ε on the quality of the returned solutions. The number of episodes, γ and α show a much lower ability to influence the generated solutions. According to Figure (3), in general, high values of ε show the ability to exert a strong positive influence on the returned solutions. In some cases (large instances), low and medium values of this parameter show a similar ability. This is related to the modified mapping strategy. High values of γ are beneficial. In the case of number of episodes and α, the influence trend is strongly mixed. However, in some cases, low numbers of episodes showed the ability to exert a strong positive influence.

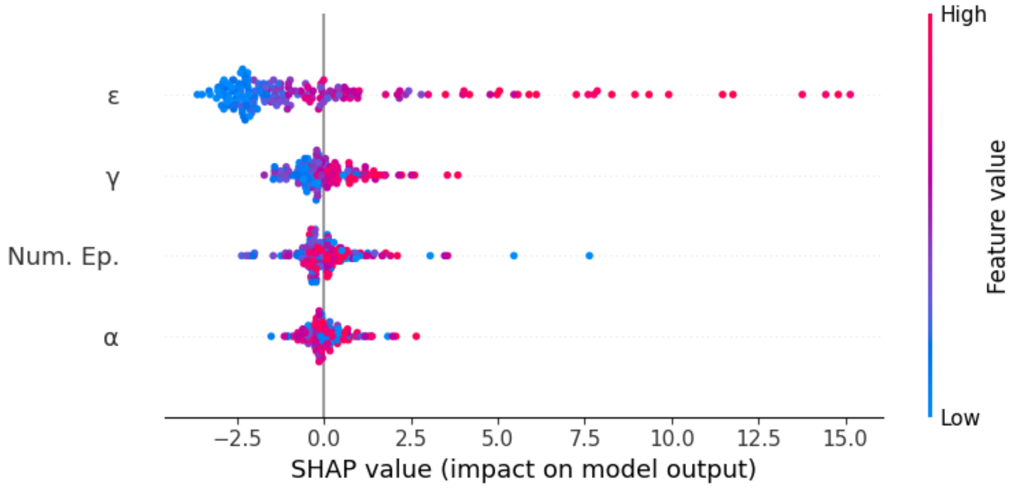

Fig. 3. SHAP Summary Plot for Q-learning algorithm with long term profitability

Tab. 4. Optimal hyperparameter configurations for the Q-learning algorithm with long term profitability

Instance dimension	α	γ	ε	Number of episodes
5	0.602595856	0.591650661	0.947070837	1552
10	0.887286286	0.742201905	0.440510978	4529
25	0.871624447	0.589830703	0.090013667	4994
50	0.909138517	0.730348748	0.012323453	2648
100	0.358076148	0.612671906	0.020874408	1696
200	0.805077677	0.843362572	0.017584689	4343

Looking at the data in Table 4, we can see that the values of the α coefficient are generally high (the approach requires intensive learning). In turn, the hyperparameter γ is moderate, which is an unusual situation from the point of view of focusing on obtaining globally optimal solutions and indicates the impact of the implemented modifications. The values of the ε coefficient are high for small instances and very low for larger instances, so the implemented long-term profitability strategy is more beneficial in the case of faster narrowing of the exploration policy. The number of episodes is generally high.

3.4. Q-learning with backward optimization

The ranking of the influence of the hyperparameters (Table 1) shows a strong influence of ε on the quality of the returned solutions. The coefficients γ, number of episodes and α have a much smaller influence. Looking at the data from the SHAP plot (Figure 4), it can be seen that the improvement of the solutions is obtained with high values of ε. The influence of high and low number of episodes and values of γ and α is strongly diversified, with a tendency of positive influence of some lower values of number of episodes in some cases.

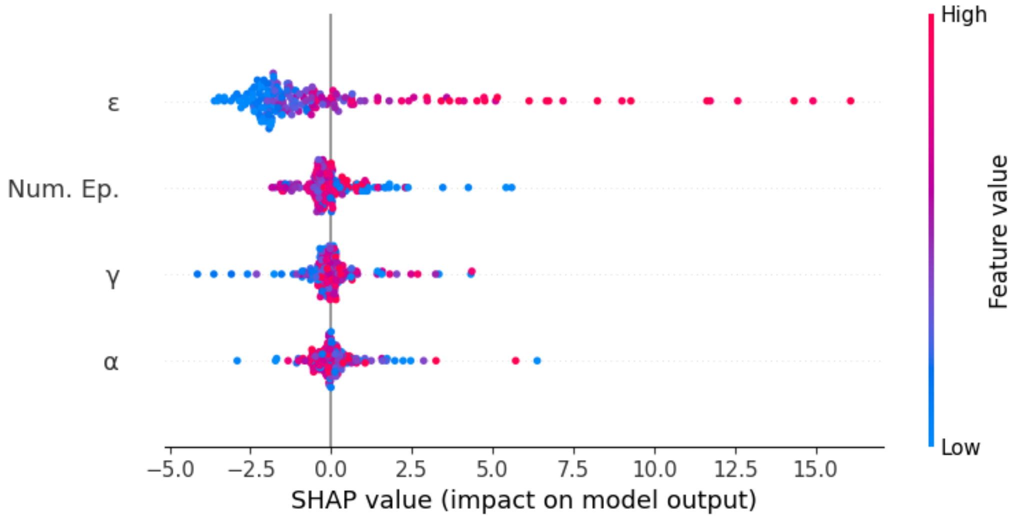

Fig. 4. SHAP Summary Plot for Q-learning algorithm with backward optimization

Tab. 5. Optimal hyperparameter configurations for the Q-learning algorithm with backward optimization

Instance dimension	α	γ	ε	Number of episodes
5	0.480665636	0.532148941	0.452874689	4686
10	0.975338809	0.650906724	0.344045493	4249
25	0.180388867	0.504711696	0.117021317	3871
50	0.845071277	0.669749132	0.051060022	2048
100	0.560788166	0.913400068	0.011397173	4420
200	0.530869001	0.525611789	0.010909041	4492

Based on the data in Table 5, we can conclude that in the case of Q-learning with backward optimization, the values of the α coefficient do not show a clear trend. The learning efficiency does not depend on the size of the problem, which is related to the implemented backward optimization strategy. Similarly to the case of the Q-learning with Long-Term Profitability algorithm, the values of the γ hyperparameter are moderate, which is due to the use of an additional strategy of considering long-term rewards. Similarly to the case of Q-learning with Long-Term Profitability, the value of the ε coefficient is higher for small instances and very low for larger ones.

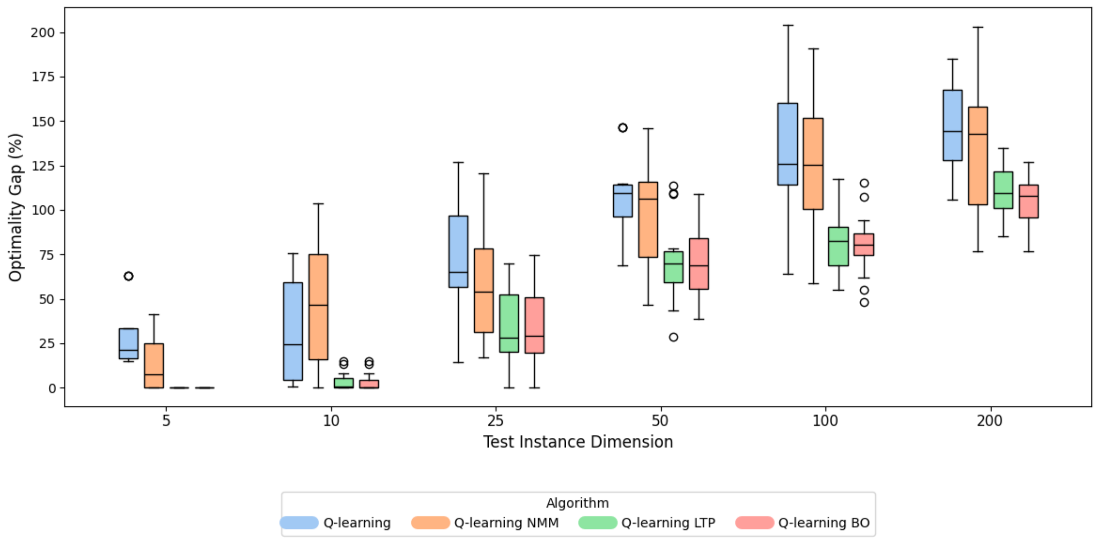

Fig. 5. Distribution of the optimality gap (%) across Q-learning algorithm variants

Looking at the data presented in Figure 5, we can see that the basic version of the Q-learning algorithm is characterized by the lowest efficiency and stability in almost all test cases. In particular, for larger instances (e.g., size 200), the median degradation reached 144.52%, with a wide interquartile range and several extreme outliers observed, indicating high volatility and poor convergence. Even for small problems, such as instance size 5, the algorithm showed a median degradation of 21.47% with significant outliers, indicating poor suitability for this class of problems.

Q-learning with min-max normalization is characterized by an improvement of the median solution quality, especially for small instances (e.g. 7.35% for size 5), and the complete elimination of outliers over all sizes. However, a limited scalability is still observed, as larger instances show degradation values similar to the basic version of the algorithm (e.g. 142.84% median for size 200), suggesting that normalization alone does not sufficiently improve decision policies for complex tasks.

The Q-learning with Long-Term Profitability variant yielded significant performance gains, especially for small and medium-sized instances. For example, the median degradation was 0% for size 5 and 0.43% for size 10, indicating a strong ability to identify near-optimal assignments. While performance deteriorated with increasing problem size (e.g., 82.49% for size 100), the algorithm still showed lower variance and lower frequency of outliers, demonstrating improved stability.

Q-learning with backward optimization was the best performing variant across all metrics. This method maintained exceptional quality for small instances (e.g., 0.32% median for size 10), while also outperforming all other variants for large instances (e.g., 107.64% median for size 200). Notably, the algorithm consistently exhibited minimal interquartile ranges and very few outliers, confirming its superior robustness and resistance to training noise. The backward search mechanism appears to be particularly effective in exploiting late-stage decisions in long assignment sequences. The algorithm exhibits the most desirable properties among the variants evaluated: high efficiency, low degradation, and strong stability, making it the most suitable choice for solving the assignment problem over a wide range of instance sizes.

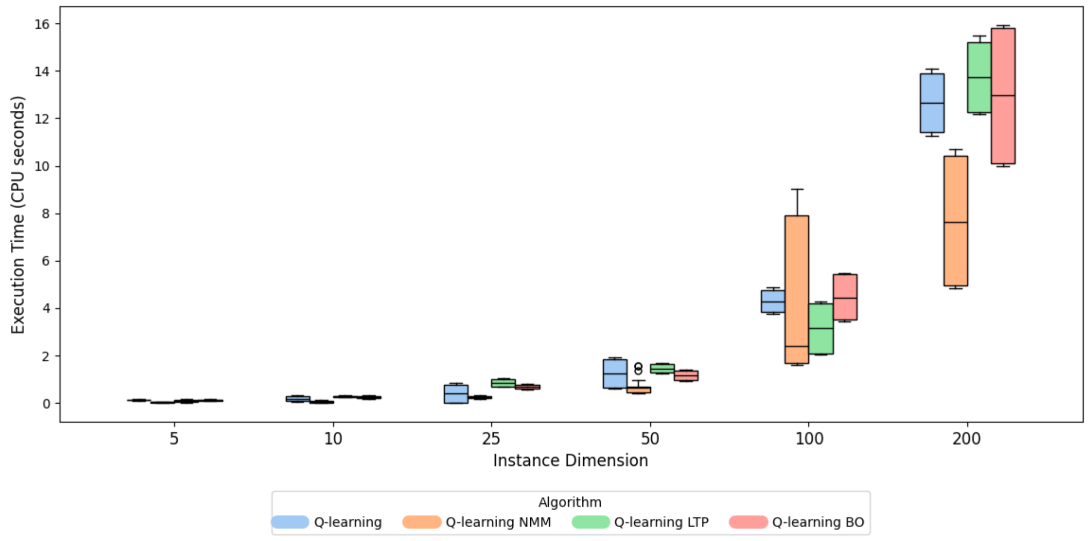

Fig. 6. Execution time distribution across Q-learning algorithm variants

Looking at the data presented in Figure 6, it is evident that for all algorithm variants, the execution time increases significantly with the problem size, which is expected due to the exponential growth of the decision space. This phenomenon is characteristic of learning-based methods that explore action spaces through iterative updates, especially when using exploration-exploitation strategies.

The basic Q-learning variant exhibits moderate execution time for small problem sizes (e.g., median ≈ 0.12s for size 5), but the runtime increases rapidly for larger instances. At size 200, the median exceeds 12.65s, with values approaching 14.1s. The algorithm shows greater variability for small problems (wider interquartile range), but becomes more consistent for larger problems, albeit at a high computational cost. This suggests an inefficient learning dynamic that stabilizes only after the algorithm reaches saturation in high-dimensional spaces.

Q-learning with min-max normalization shows superior computational efficiency across all instance sizes. For size 5, the median execution time is only 0.02s, and even for size 200 it remains below 7.65s, significantly outperforming all other variants. The method shows tight distributions for small instances and greater variability for larger instances, while still maintaining a favorable upper bound. This is due to improved convergence and reduced learning noise, making this variant the most computationally efficient.

While Q-learning with Long-Term Profitability provides high solution quality, it comes at the cost of increased execution time, especially for large instances. For instance size 200, the median runtime exceeds 13.70s, with maximum values exceeding 15s. Compared to the basic version of the algorithm, it is more stable for small instances, with consistent runtimes. For large problems, however, its computational cost increases more steeply, reflecting the overhead of maintaining and evaluating long-term action histories.

Q-learning with backward optimization has similar execution time characteristics to Long-Term Profitability, but with slightly better scalability. At size 200, its median execution time is about 12.95 s, and while still computationally expensive, it maintains a lower variance compared to Q-learning with Long-Term Profitability on larger problems. The algorithm benefits from reverse history traversal, which seems to provide more efficient processing of accumulated knowledge as complexity increases.

4. CONCLUSIONS

The basic approach of Q-learning in the context of solving optimization problems has certain limitations. Among them, we can mention the susceptibility of the values in the Q-table to be dominated by certain elements (minima or maxima) of the cost matrix. This is particularly detrimental in the case of small instances of the considered problems, the inability to precisely adapt the algorithm to the specifics of the problem, or difficulties in identifying appropriate hyperparameter values. The Q-learning algorithm also tends to deviate from the specified Q-values when making assignments (worsening the solution). The Q-learning algorithm in its natural form is not adapted to the specifics of the assignment problem. The final solution is determined based on the Q-table, and this process takes the form of iterating over subsequent rows and selecting the most favorable Q-values in them. In practice, this approach is a greedy heuristic that does not take into account the impact of the current decision on future ones. In the learning process, the algorithm in this form does not take into account previous assignments. The procedure is based only on the computed (based on the cost and reward/penalty matrix) values of Q. It should be emphasized that a more accurate representation of the problem would require including in the state information about the previous assignments and the current total cost. However, such a change would make the Q table too large (the dimension of the table would be 2n). With n equal to 25, the Q table would have to store information about more than 33 million states (this would require about 156 GB of RAM).

A certain form of limiting the negative impact of the assignment process is provided by the Q-learning with Long-Term Profitability and with Backward Optimization variants, in which the full policy and long-term profitability of assignments are taken into account. A different function is responsible for returning the final solution than in the Q-learning and Q-learning with Normalization Min-Max versions. As a result, a better fit to the specifics of the problem is achieved, which is usually reflected in better quality solutions.

The size of the Q-table depends on the size of the problem. Thus, large instances cause the generation of a large Q-table, which is the reason for the longer time to return solutions and problems with scalability and generalization. Another significant limitation is the fact that the values in the Q-table are closely linked to the cost matrix (Q-values are adapted to a specific environment). An attempt to apply the learned Q-table to a different problem will not yield the expected result. Therefore, it is necessary to train the algorithm each time the input data changes (changes in the cost matrix). The situation is similar when the problem size changes (the algorithm is not able to generalize effectively).

In practice, the Q-learning algorithm represents an approach that deviates significantly from the structural characteristics of the Assignment Problem, as well as from the nature of many other combinatorial optimization problems, such as the Travelling Salesman Problem or those belonging to the broader class of Vehicle Routing Problems. Despite its relative flexibility and considerable theoretical potential, it remains a challenge to adapt Q-learning in a way that consistently ensures high quality solutions. Nevertheless, it should be noted that in many cases the solutions generated by the analyzed algorithm variants can be considered acceptable, especially in scenarios where no superior methods are available. However, it is advisable to pursue improvements and refine the structure of the algorithm, even if this requires departing from its classical formulation. This is supported by the superior performance observed in variants that incorporate strategies such as long-term profitability and backward optimization. Moreover, hyperparameter tuning should be treated as an essential step in the application of Q-learning, since the chosen hyperparameter values have a significant influence on the quality of the resulting solutions. The optimal configuration of these parameters is highly dependent on the specific characteristics of the problem instance, as well as on the internal mechanics of each algorithmic variant. Relying on default or randomly selected hyperparameter values often leads to a significant reduction in the algorithm's ability to generate high-quality solutions. However, it is important to recognize that hyperparameter optimization is generally a computationally intensive process, often requiring significant time and computational resources.

Conflicts of interest

This work does not have any conflict of interest to declare.

REFERENCES

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 2623–2631). Association for Computing Machinery. https://doi.org/10.1145/3292500.3330701

Asef-Vaziri, A., Kazemi, M., & Radman, M. (2022). The facility layout instances of the generalised travelling salesman problem. *International Journal of Production Research, 60*(19), 5794–5811. https://doi.org/10.5604/01.3001.0015.8269

Baller, A. C., Dabia, S., Dullaert, W. E., & Vigo, D. (2020). The vehicle routing problem with partial outsourcing. *Transportation Science*, *54*(4), 1034–1052. https://doi.org/10.1287/TRSC.2019.0940

Bladen, K., & Cutler, D. R. (2024). Assessing agreement between permutation and dropout variable importance methods for regression and random forest models. *Electronic Research Archive*, *32*(7), 4495–4514. https://doi.org/10.3934/ERA.2024.7.4495

Boccia, M., Masone, A., Sforza, A., & Sterle, C. (2021). A column-and-row generation approach for the flying sidekick travelling salesman problem. *Transportation Research Part C: Emerging Technologies*, *124*, 102913. https://doi.org/10.1016/j.trc.2021.102913

Cárdenas-Montes, M. (2018). Creating hard-to-solve instances of travelling salesman problem. *Applied Soft Computing*, *71*, 268–276. https://doi.org/10.1016/j.asoc.2018.07.010

Chen, T., Chu, F., Zhang, J., & Sun, J. (2024). Sustainable collaborative strategy in pharmaceutical refrigerated logistics routing problem. *International Journal of Production Research*, *62*(14), 5036–5060. https://doi.org/10.1080/00207543.2023.2283566

Chládek, P., Smetanová, D., & Krile, S. (2018). On some aspects of graph theory for optimal transport among marine ports. *Zeszyty Naukowe. Transport/Politechnika Śląska*, *101*, 37–45. https://doi.org/10.20858/sjsutst.2018.101.4

Clifton, J., & Laber, E. (2020). Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, *7*(1), 279–301. https://doi.org/10.1146/annurev-statistics-031219-041220

Eimer, T., Lindauer, M., & Raileanu, R. (2023, July). Hyperparameters in reinforcement learning and how to tune them. *International Conference on Machine Learning* (pp. 9104–9149). PMLR. https://doi.org/10.1109/ICML.2023.9149494

Fumagalli, F., Muschalik, M., Hüllermeier, E., & Hammer, B. (2023). Incremental permutation feature importance (iPFI): Towards online explanations on data streams. *Machine Learning*, *112*(12), 4863–4903. https://doi.org/10.1007/s10994-023-06385-y

Hu, M. (2023). *The art of reinforcement learning: Fundamentals, mathematics, and implementations with Python*. Apress.

Ilosvay, V., & Iaccarino, E. (2023). Exploring and optimizing reinforcement learning algorithms in the Frozen Lake environment (in deterministic and stochastic env) and hyperparameters optimization through *Optuna*. https://doi.org/10.13140/RG.2.2.35989.09445

Jarrett, D., Stride, E., Vallis, K., & Gooding, M. J. (2019). Applications and limitations of machine learning in radiation oncology. *The British Journal of Radiology*, *92*(1100), 20190001. https://doi.org/10.1259/bjr.20190001

Loecher, M. (2024). Debiasing SHAP scores in random forests. *AStA Advances in Statistical Analysis*, *108*(2), 427–440. https://doi.org/10.1007/s10182-024-00452-w

Małek, A., Caban, J., Dudziak, A., Marciniak, A., & Vrábel, J. (2023). The concept of determining route signatures in urban and extra-urban driving conditions using artificial intelligence methods. *Machines*, *11*(5), 575. https://doi.org/10.3390/machines11050575

Miller, T. (2024). *Mastering reinforcement learning*. The University of Queensland.

Nichols, J. A., Herbert Chan, H. W., & Baker, M. A. (2019). Machine learning: Applications of artificial intelligence to imaging and diagnosis. *Biophysical Reviews*, *11*, 111–118. https://doi.org/10.1007/s12551-018-0500-7

Pečený, L., Meško, P., Kampf, R., & Gašparík, J. (2020). Optimisation in transport and logistic processes. *Transportation Research Procedia*, *44*, 15–22. https://doi.org/10.1016/j.trpro.2020.02.003

Rabbani, Q., Khan, A., & Quddoos, A. (2019). Modified Hungarian method for unbalanced assignment problem with multiple jobs. *Applied Mathematics and Computation*, *361*, 493–498. https://doi.org/10.1016/j.amc.2019.07.022

Salih, A. M., Raisi-Estabragh, Z., Galazzo, I. B., Radeva, P., Petersen, S. E., Lekadir, K., & Menegaz, G. (2025). A perspective on explainable artificial intelligence methods: SHAP and LIME. *Advanced Intelligent Systems*, *7*(1), 2400304. https://doi.org/10.1002/aisy.202400304

Scikit-learn. (2024). *Permutation feature importance*. Retrieved May 4, 2025 from https://scikit-learn.org/dev/modules/permutation_importance.html

Sharma, A., Jain, A., Gupta, P., & Chowdary, V. (2020). Machine learning applications for precision agriculture: A comprehensive review. *IEEE Access*, *9*, 4843–4873. https://doi.org/10.1109/ACCESS.2020.2962538

Sharma, N., Sharma, R., & Jindal, N. (2021). Machine learning and deep learning applications - a vision. *Global Transitions Proceedings, 2*(1), 24–28. https://doi.org/10.1016/j.gtp.2021.05.004

Shekhar, S., Bansode, A., & Salim, A. (2021, December). A comparative study of hyper-parameter optimization tools. *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1–6). IEEE. https://doi.org/10.1109/CSDE52545.2021.9761710

Shinde, P. P., & Shah, S. (2018, August). A review of machine learning and deep learning applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICCUBEA.2018.00025

Shopov, V. K., & Markova, V. D. (2021, September). Application of Hungarian algorithm for assignment problem. *In 2021 International Conference on Information Technologies (InfoTech)* (pp. 1–4). IEEE. https://doi.org/10.1109/InfoTech52438.2021.9548600

Shramenko, N., Merkisz-Guranowska, A., Kiciński, M., & Shramenko, V. (2022). Model of operational planning of freight transportation by tram as part of a green logistics system. *Archives of Transport*, *63*(3), 113–122. https://doi.org/10.5604/01.3001.0015.9929

Surblys, V., Kozłowski, E., Matijošius, J., Gołda, P., Laskowska, A., & Kilikevičius, A. (2024). Accelerometer-based pavement classification for vehicle dynamics analysis using neural networks. *Applied Sciences*, *14*(21), 10027. https://doi.org/10.3390/app142110027

Taha, H. A. (2017). *Operations research: An introduction* (10th ed.). Pearson.

Taran, I., Bikhimova, G., Danchuk, V., Toktamyssova, A., Tursymbekova, Z., & Oliskevych, M. (2024). Improving the methodology for optimizing multimodal transportation delivery routes and cyclic schedules in a transnational direction. *Transport Problems: An International Scientific Journal*, *19*(1). https://doi.org/10.20858/tp.2024.19.1.13

Tarapata, Z., Kulas, W., & Antkiewicz, R. (2022). Machine learning algorithms for the problem of optimizing the distribution of parcels in time-dependent networks: The case study. *Archives of Transport*, *61*(1), 133–147. https://doi.org/10.5604/01.3001.0015.8269

Vadseth, S. T., Andersson, H., Stålhane, M., & Chitsaz, M. (2023). A multi-start route improving matheuristic for the production routeing problem. *International Journal of Production Research*, *61*(22), 7608–7629. https://doi.org/10.1080/00207543.2022.2154402

Wang, H., Liang, Q., Hancock, J. T., & Khoshgoftaar, T. M. (2024). Feature selection strategies: A comparative analysis of SHAP-value and importance-based methods. *Journal of Big Data*, *11*, 44. https://doi.org/10.1186/s40537-024-00874-7

Zhang, J., Liu, C., Li, X., Zhen, H. L., Yuan, M., Li, Y., & Yan, J. (2023). A survey for solving mixed integer programming via machine learning. *Neurocomputing*, *519*, 205–217. https://doi.org/10.48550/arXiv.2203.02878

Zhu, C., Dastani, M., & Wang, S. (2024). A survey of multi-agent deep reinforcement learning with communication. *Autonomous Agents and Multi-Agent Systems*, *38*, 4. https://doi.org/10.1007/s10458-023-096

Zhu, L., & Hu, D. (2019). Study on the vehicle routing problem considering congestion and emission factors. *International Journal of Production Research*, *57*(19), 6115–6129. https://doi.org/10.1080/00207543.2018.1533260