mathematics

MDPI

Article

Optimizing Maintenance of Energy Supply Systems in City Logistics with Heuristics and Reinforcement Learning

Antoni Guerrero [1,2], Angel A. Juan [2,*], Alvaro Garcia-Sanchez [3] and Luis Pita-Romero [1]

[1] Baobab Soluciones, Jose Abascal 55, 28003 Madrid, Spain; antoni.guerrero@baobabsoluciones.es (A.G.)
[2] Research Center on Production Management and Engineering, Universitat Politècnica de València, Plaza Ferrandiz-Carbonell, 03801 Alcoy, Spain
[3] Department of Organization Engineering, Business Administration and Statistics, Universidad Politécnica de Madrid, Jose Abascal 2, 28006 Madrid, Spain; alvaro.garcia@upm.es
* Correspondence: ajuanp@upv.es

Abstract: In urban logistics, effective maintenance is crucial for maintaining the reliability and efficiency of energy supply systems, impacting both asset performance and operational stability. This paper addresses the scheduling and routing plans for maintenance of power generation assets over a multi-period horizon. We model this problem as a multi-period team orienteering problem. To address this multi-period challenge, we propose a dual approach: a novel reinforcement learning (RL) framework and a biased-randomized heuristic algorithm. The RL-based method dynamically learns from real-time operational data and evolving asset conditions, adapting to changes in asset health and failure probabilities to optimize decision making. In addition, we develop and apply a biased-randomized heuristic algorithm designed to provide effective solutions within practical computational limits. Our approach is validated through a series of computational experiments comparing the RL model and the heuristic algorithm. The results demonstrate that, when properly trained, the RL-based model is able to offer equivalent or even superior performance compared to the heuristic algorithm.

Keywords: optimization; energy supply systems; city logistics; team orienteering problem; biased-randomized algorithms; reinforcement learning

MSC: 90-10; 90C59; 90B06

Citation: Guerrero, A.; Juan, A.A.; Garcia-Sanchez, A.; Pita-Romero, L. Optimizing Maintenance of Energy Supply Systems in City Logistics with Heuristics and Reinforcement Learning. *Mathematics* **2024**, *12*, 3140. https://doi.org/10.3390/math12193140

Academic Editors: Kin Keung Lai, Lean Yu and Jian Chai

Received: 28 August 2024
Revised: 24 September 2024
Accepted: 5 October 2024
Published: 7 October 2024

1. Introduction

Effective maintenance scheduling and routing are critical components in the management of power generation assets. City logistics [1] rely heavily on a stable energy supply for everything from transportation and communication to residential and industrial power needs. Ensuring a reliable energy supply becomes increasingly challenging due to the complexity of infrastructure, the volume of demand, and the need for continuous operation. The industrial assets are inevitably affected by aging and degradation of their components, which strongly impact production availability and product quality, which in turn has led to to a surge of interest in maintenance optimization in recent years [2]. Effective maintenance optimization addresses these challenges by ensuring that corrective and preventive actions are strategically planned and executed, minimizing system failures and improving overall reliability. Predictive maintenance takes this a step further [3], using data to anticipate failures before they occur, reducing unnecessary downtime and enabling smoother city operations. Ensuring that maintenance tasks are performed efficiently is essential for minimizing downtime, optimizing operational efficiency, and maximizing energy production [4]. For example, in wind-energy farms, operations and maintenance costs constitute a significant portion of the overall life cycle cost, often accounting for approximately 20% to 25% [5].

The problem analyzed in this paper, which is based on a real-life case from a Spanish company, involves scheduling and routing for the maintenance of power generation assets. This problem is particularly complex when managing multiple assets spread across various locations, each requiring different types of maintenance at different times. Solving this challenge effectively can lead to significant improvements in operational reliability and financial performance [6]. In the context of city logistics, similar principles apply as urban environments demand efficient scheduling and routing of service tasks to maintain the power generation infrastructure [7]. Optimizing maintenance in urban power generation involves balancing immediate needs with long-term goals, which requires optimizing both the scheduling of urban maintenance tasks and the routing of service teams across a network of nodes. This dual optimization challenge is modeled as a variation of the team orienteering problem (TOP), a well-known combinatorial optimization problem [8]. Our problem extends the traditional TOP by incorporating a multi-horizon scenario where the solution must cover a period of five days, corresponding to a typical working week. This additional layer of complexity requires balancing short-term and long-term objectives, considering not only the immediate maintenance needs but also the operational constraints and priorities over the entire period. To the best of our knowledge, this is the first work to model a maintenance routing and scheduling problem as a multi-period TOP, which allows the use of well-known heuristics to solve it.

In our case, there will be several power generation facilities, each consisting of three main components that require maintenance. It will be assumed that the following types of maintenance can be performed: (i) corrective maintenance, where the asset is not functioning properly and needs repair; (ii) preventive maintenance, where the asset is functioning properly but requires inspection; and (iii) predictive maintenance, which is performed based on the probability of asset failure. In all cases, it is important to account for maintenance time, as facilities do not generate energy while undergoing maintenance. Moreover, it will be assumed that all maintenance teams will start their routes from a single depot, where they will also return as the last stop in their routes. The inclusion of the three types of maintenance—corrective, preventive, and predictive—is also a novel contribution, as most existing studies typically address at most two of them. Figure 1 provides a schematic representation of the problem for one day.

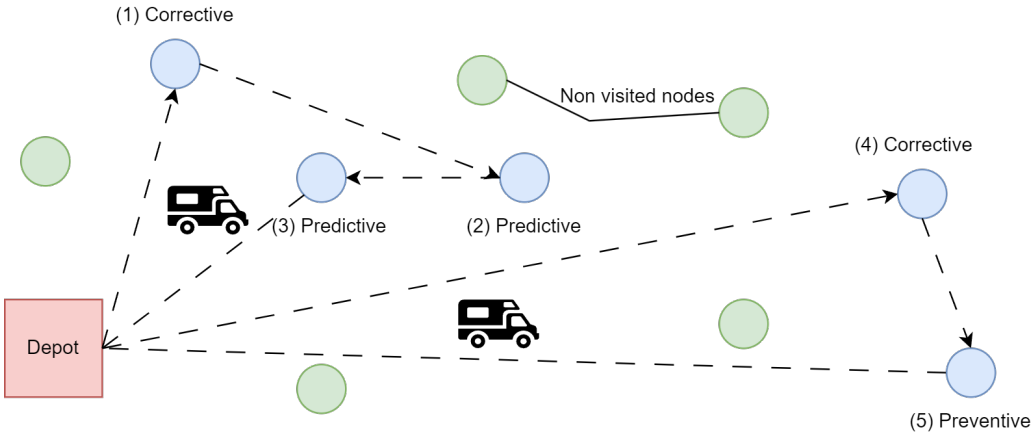

Figure 1. Schematic representation of the routing-scheduling problem for a single day. Green nodes indicate locations that have not been visited.

Each day is limited by time, as each maintenance team can only work for eight hours. Consequently, not all nodes can be visited, so both the travel distance between nodes and the maintenance times must be considered. Each maintenance task can only be performed once. Hence, the list of potential tasks should be updated at the end of each day. It is also assumed that multiple maintenance teams can work on the same facility and on different components simultaneously. The goal is to schedule all necessary maintenance for the

week (five days) to maximize profits. In addition, each asset supports only one type of maintenance. For example, with fifteen power generation facilities, there are forty-five assets to maintain. This approach is used because an asset requiring corrective maintenance will not need other types of maintenance in the short term, and vice versa. The positions of the depot and facilities are provided, allowing the computation of travel times in minutes between each pair of nodes. In this case, both travel times and maintenance times are multiples of fifteen minutes. If the travel time between two nodes or the maintenance duration is not a multiple of fifteen, it is rounded up to the next multiple to account for potential variability, as these variables are not deterministic in real-world scenarios. This implies that each day is divided into 32 timesteps, as 32×15 min equals 8 h of work. However, the number of days considered and the number of timesteps are parameters that can vary depending on the nature of the problem being modeled. An alternative approach to expand this work could involve introducing randomness and dynamism into travel times, which would better capture its inherent complexity. More sophisticated algorithms, such as simheuristics or learnheuristics, could then be employed [9]. This randomness can be attributed to factors like weather or traffic.

For each facility, the amount of electricity produced over the five days is known, and for each asset, the facility's production during maintenance is considered. For corrective maintenance, the maintenance duration is known, and for preventive maintenance, both the duration of which and the specific day on which it can be performed are known. Unlike corrective and predictive maintenance, preventive maintenance can only be performed after a certain day and must be completed within three days. This ensures that preventive maintenance is performed since assets requiring it are functioning properly and may not provide immediate benefit, which might otherwise lead to their maintenance being skipped. Finally, for predictive maintenance, the maintenance time and the probability of asset failure each day are given, which is then set to zero when the maintenance is performed. Additionally, the energy price for each day is also known.

Given the combinatorial nature of the problem and the multi-day horizon, finding an optimal solution is computationally complex. The difficulty arises from the need to coordinate maintenance activities efficiently, prioritize tasks based on their urgency and impact, and plan routes that minimize travel time while maximizing the effectiveness of each maintenance operation. This paper makes several key contributions to tackle this complex problem: (i) modeling the problem as a TOP, which allows us to make use of established solution techniques and adapt them to the specific needs of maintenance scheduling in power generation facilities; (ii) proposing a biased-randomized heuristic algorithm, which balances exploration and exploitation to find effective solutions within reasonable computational time, providing a practical approach for real-world applications; (iii) proposing a RL algorithm, which learns from interactions with the problem environment to improve decision making over time, offering a dynamic approach to solving the scheduling challenge; and (iv) comparing the results of both solution algorithms, thus evaluating their performance in terms of solution quality, computational efficiency, and practical applicability. One of the main reasons for using a biased-randomized algorithm to solve this problem is its proven effectiveness [10]. This approach is capable of providing optimal or near-optimal solutions for most TOP benchmark instances and, more broadly, for NP-hard routing and scheduling problems. Additionally, since RL is increasingly being explored as a new solving methodology for NP-hard problems, the RL-based approach has been employed to evaluate its performance on a real-world scenario and compare it with the more widely established methods. This comparison was our main research goal.

The remaining of the paper is structured as follows: Section 2 provides additional details of the optimization problem being considered. Section 3 reviews some related work. Section 4 explains the heuristic-based approach for solving the problem, while Section 5 introduces the RL-based approach. Section 6 describes the computational experiments carried out and analyzes the results obtained by each solution approach, and also introduces

a new method for improving inference results. Finally, Section 7 summarizes the main findings of the paper.

2. Additional Modeling Details

The problem will be approached as a variation of the TOP, an NP-hard problem in which a set of nodes and vehicles are given, and the objective is to maximize the rewards obtained from the nodes visited. There is one starting and one ending depot, which in this case are the same. Typically, constraints related to time or distance traveled may prevent all nodes from being visited; in this case, the constraint is time. As this is a multi-period problem, it spans several days, meaning the scheduling and routing must be optimized for different time periods. Let $M = \{1, 2, \ldots, n\} \cup \{o, d\}$ be the set of all n maintenance tasks, along with the the origin and destination depots o and d, respectively. The problem is modeled using a directed graph $G = (M, E)$, where E is the set of edges, $E = \{(i, j) \in M \times M\}$, representing the path between nodes. Let V denote the set of all vehicles and D the set of all days. The variable x_{ijrk} is a binary decision variable that takes a value of 1 if vehicle r goes from node i to j in day k. Let d_{ik} be a parameter that equals 1 if maintenance task i can be performed on day k, and 0 otherwise, s_{ij} be the travel time between nodes i and j, and m_i represent the maintenance time of node i. Both depots are available each day and have a maintenance time of zero. Let also L be the maximum time allowed per vehicle each day. Then, the mathematical model is as follows:

$$\sum_{j \in M} x_{ojrk} \leq 1 \qquad \forall r \in V, k \in D \tag{1}$$

$$\sum_{i \in M} x_{idrk} = \sum_{j \in M} x_{ojrk} \qquad \forall r \in V, k \in D \tag{2}$$

$$\sum_{i \in M} \sum_{j \in M} x_{ijrk} \leq \sum_{j \in M} x_{ojrk} \qquad \forall r \in V, k \in D \tag{3}$$

$$\sum_{i \in M} x_{iork} + \sum_{j \in M} x_{djrk} = 0 \qquad \forall r \in V, k \in D \tag{4}$$

Equations (1) and (2) state that if a vehicle exits from the depot, it must also reach the destination depot. Equation (3) ensures that a vehicle can only visit other nodes if it has exited from the depot, while Equation (4) enforces the origin depot to be the first node of the route, and the destination depot the final one.

Meanwhile, Equation (5) ensures that each node is visited at most once, except the destination depot, regardless of the vehicle or the day:

$$\sum_{k \in D} \sum_{r \in V} \sum_{i \in M} x_{ijrk} \leq 1 \qquad \forall j \in N \setminus \{d\} \tag{5}$$

$$\sum_{j \in M} x_{ijrk} = \sum_{j \in M} x_{jirk} \qquad \forall i \in M \setminus \{o, d\}, r \in V, k \in D \tag{6}$$

$$\sum_{r \in V} \sum_{i \in M} x_{ijrk} \leq d_{jk} \qquad \forall j \in M, k \in D \tag{7}$$

Equation (6) states that if a vehicle arrives at a node, it must also depart from that node, except for the depots. Meanwhile, Equation (7) ensures that a node cannot be visited on day k if it is not available, i.e., if it falls outside its time window:

$$\sum_{i \in M} \sum_{j \in M} x_{ijrk}(s_{ij} + m_j) \leq L \qquad \forall r \in V, k \in D \tag{8}$$

Restriction (8) ensures that the travel time for each vehicle on a given day does not exceed the maximum time allowed. Note that this cost is not symmetric, since the cost of

going from i to j is not the same as going from j to i, since the maintenance time of i and j might not be the same. Finally, restriction (9) defines the binary nature of the variable:

$$x_{ijrk} \in \{0,1\} \qquad \forall i,j \in M, r \in V, k \in D \tag{9}$$

In this context, it is important to understand the priority of doing each type of maintenance, i.e, the impact each one of them has on the profits. For this problem, these priorities are as follows: (i) corrective maintenance, prioritized from highest to lowest energy production; (ii) preventive maintenance, prioritized from highest to lowest energy production, except on the last maintenance day, when its priority should exceed that of corrective maintenance; and (iii) predictive maintenance, pr ioritized from highest to lowest energy production and from highest to lowest failure probability. In this context, the objective function to maximize, as shown in Equation (10), is:

$$\sum_{\substack{d,k \in D \\ d \leq k}} \sum_{r \in V} \sum_{i,j \in M} x_{ijrd}\, p_j\, b_{jk}\, e_k \tag{10}$$

The summation over each day, as well as previous ones, ensures that the most important tasks are prioritized and completed as early as possible, as the reward for each node considers not only the day the maintenance is performed but also the preceding days. Moreover, each variable in the objective function has the following meaning: (i) p_i represents the energy production of the facility associated with maintenance i; (ii) e_k denotes the price at which energy is sold on day k; and (iii) b_{ik} is a value that depends on the type of maintenance i and the day k. This value ensures the correct prioritization of maintenance tasks. For example, assigning the highest value to corrective maintenance implies that these tasks should be performed as soon as possible, rather than focusing on other types of maintenance. However, other factors need to be considered, such as the variation in energy production across facilities and the differing maintenance times for each type, since the duration of maintenance for each asset can vary. Assets requiring corrective maintenance typically have longer durations compared to those needing preventive maintenance, and preventive maintenance usually takes more time than predictive maintenance. Therefore, finding the optimal values for each scenario is important to achieve the best possible solution. For predictive maintenance, b_{ik} equals the probability that the asset associated with maintenance i will fail on day k. For corrective and preventive maintenance, the following values are used: (i) corrective, where $b_{ik} = 3$; and (ii) preventive, where b_{ik} is equal to 0 except on the second day of the time window, when it is set to 6, and on the last day, when it is set to 18. The reason for varying this preventive value depending on the day is that, since assets requiring preventive maintenance are still operational, it is preferable to prioritize other types of maintenance first and schedule preventive maintenance for the latest possible day. The second day also has a reward to account for the edge case where maintenance cannot be performed on the last day.

Since preventive maintenance is prioritized over predictive maintenance, one way to determine initial values for preventive maintenance is to evaluate the reward contributions of tasks with the highest maintenance time and lowest production compared to those with the highest production and lowest maintenance time for predictive maintenance. This can be achieved by calculating the ratio between the maximum production of predictive maintenance and the minimum production of preventive maintenance, and then multiplying it by the ratio between the maximum maintenance time of preventive maintenance and the minimum maintenance time of predictive maintenance. It is also important to consider all time windows. While the values mentioned above work well with the data used in our analysis, they might be fine-tuned for each instance of the problem.

3. Related Work

Pinciroli et al. [11] optimized the maintenance of renewable energy systems using reinforcement learning, specifically employing the PPO algorithm for training. The study

focused on two types of maintenance: corrective and predictive, with the failure rate modeled by a density function. This research was conducted as a long-term study, where the proposed model was compared against a heuristic and other different models. Their optimization was based on a single global indicator rather than multiple components as in our case. Dong et al. [12] developed a deep neural network to determine the optimal timing and location for preventive maintenance of wind turbines, taking into account the condition of components like blades and gearboxes. Similarly, Ilić and Žarković [13] tested various machine learning models to schedule maintenance tasks. However, to the best of our knowledge, no existing paper addresses the scheduling and routing problem using RL and compares the results with those obtained using a well-established heuristic.

As discussed earlier, the problem has been modeled as a modified TOP [14], where time windows [15] and the objective of maximizing rewards over multiple days must be taken into account. The multi-period TOP with time windows was first introduced by Tricoire et al. [16], who proposed two heuristics: the first being a constructive heuristic that serves as the foundation for a variable neighborhood search based on a stochastic local search. Other algorithms have also been proposed, such as an iterated local search approach by Zucchi et al. [17], which generates an initial solution that is iteratively improved through perturbations and local search techniques until a time limit is reached, applied in their case to a car patrol company. Various other methods, including genetic algorithms [18], particle swarm optimization [19], and colony search algorithms [20], have also been proposed in different contexts. In the latter, the colony search metaheuristic mimics early human behavior, where humans expanded settlements in search of better living conditions. The work by Wen et al. [21] builds upon this concept by proposing the crow-colony search optimization algorithm. For other TOP variants, efforts have been made to combine heuristics and RL. For instance, Vincent et al. [22] introduced a TOP with time windows and a set orienteering problem, which they solve by combining simulated annealing with RL, using the ε-greedy approach to learn fitness values based on neighborhood moves.

Furthermore, for solving combinatorial optimization problems using exclusively RL, a common approach to handling the data are to use an encoder–decoder architecture. This architecture has two main components: an encoder, which processes a variable-length sequence as input, and a decoder, which uses the encoded input along with the current state of the solution to predict the next step in the sequence. One of the earliest instances where this architecture and reinforcement learning was constructively applied to solve combinatorial optimization problems was in the work by Bello et al. [23]. They trained a pointer network based on the LSTM network [24] to solve the Traveling Salesman Problem (TSP) and the Knapsack Problem, producing promising results. This model was later refined and adapted to other problem types, such as the Vehicle Routing Problem (VRP), as demonstrated by Nazari et al. [25]. Although other types of neural networks, such as recurrent neural networks [26] and graph neural networks [27], have been used, the introduction of the transformer architecture by Vaswani [28] led to the development of more robust models. These models leverage the transformer's capability to effectively capture relationships between data points. One of the first of these was the attention model introduced by Kool et al. [29], which achieved near-optimal results when solving the TSP. Since then, the transformer architecture has become the foundation for most models, enabling the application of RL to increasingly complex problems. For instance, the AMARL model proposed by Gao et al. [30] was designed to solve the multi-vehicle TSP, which introduces challenges in coordinating the different agents. Another approach to address the multi-vehicle TSP was introduced by Cao et al. [31], with the development of the DAN model. Additionally, new constraints, such as time windows, have been incorporated into these models. For example, Zhang et al. [32] tackled the multi-vehicle TSP with time windows and rejections, where the primary challenge was ensuring that the model could generate feasible solutions that adhere to the time windows. Similarly, Zong et al. [33] developed a model for the VRP with time windows, featuring a task handler to facilitate cooperation between vehicles. In all the cases mentioned above, the performance of the RL

algorithms was comparable to that of the heuristics and consistently achieved near-optimal results. Moreover, instead of RL, other approaches have also been employed, such as unsupervised learning [34] and supervised learning [35], although the latter requires a large amount of high-quality solution data, which is often unrealistic to obtain for these type of problems. For instance, the original pointer network was trained using supervised learning, as can be seen in the work by Vinyals et al. [36]. For learning more about neural combinatorial optimization problems, the work by Berto et al. [37] offers a comprehensive resource. This unified benchmark includes a library containing numerous state-of-the-art methods and various combinatorial optimization problems. It allows for modifications to be made to the baseline used in REINFORCE algorithms or changes to the training approach, such as switching to PPO [38]. Additionally, the library also supports the development of improvement methods, as seen in the works by Ma et al. [39] and Ma et al. [40].

4. Heuristic Approach

The first approach used to solve the problem described in the previous section involves a savings-based heuristic. This methodology, proposed by Panadero et al. [41], has been adapted to suit this particular problem. This approach employs a biased-randomized technique with a geometric probability distribution and a self-tuning mechanism, allowing the algorithm to determine the optimal parameter β to use in the geometric distribution. As shown in Figure 2, since the problem is solved using a randomized approach, the heuristic is given a maximum time to run, during which multiple solutions are generated. Each time a solution is obtained, it is compared to the current best solution. If the new solution is better, it replaces the current best solution. If not, the new solution is discarded, and the process continues. This cycle repeats until the time runs out, at which point the script stops and returns the best solution found. Additionally, to account for daily variability in rewards and nodes, the following steps are implemented for each day in the horizon period: (i) compute the rewards; (ii) solve the problem for that day; (iii) update what nodes can be visited; and (iv) proceed to the next day.

Algorithm 1 outlines the heuristic process in detail. Initially, the global solution, which stores the solutions for each day, is initialized (line 1). For each day, the node's rewards are computed (lines 2 and 3), and a dummy solution is created (line 4). This dummy solution consists of as many routes as there are nodes, with each route being a depot–node–depot route. Additionally, each pair of nodes has two savings values, one for each edge direction. The savings associated with each pair of nodes are then computed, and a savings list is created, containing the savings for each edge, sorted from highest to lowest (line 5). The savings $s_{i,j}$ for each pair of nodes i and j are obtained as follows:

$$s_{i,j} = \alpha(c_{0,j} + c_{i,n+1} - c_{i,j}) + (1 - \alpha)(r_i + r_j) \tag{11}$$

where $c_{0,j}$ represents the cost of traveling from the initial depot to node j, $c_{i,n+1}$ represents the cost of traveling from node i to the final depot (which in this case is the same as the starting one), and $c_{i,j}$ represents the cost of traveling from node i to node j. Additionally, r_i and r_j denote the rewards associated with nodes i and j, respectively. Note that the cost of traveling from node i to node j is not necessarily the same as the cost of traveling from j to i, as the cost from i to j includes the travel time from i to j plus the maintenance time at node j, which can be different from the maintenance time of node i. While the savings list is not empty, a new edge is selected based on the savings, either in a greedy manner (choosing the edge with the highest savings) or through a biased-randomized approach (lines 6 and 7). Sometimes, the best short-term move may not be the best in the long run, so introducing randomness can lead to better solutions. The current routes containing nodes i and j are then checked to see if they can be merged, ensuring that the route time does not exceed the maximum time. If they can be merged, the original routes are removed from the solution, and the new merged route is added (lines 8 to 16). The selected edge is subsequently removed from the savings list (line 17). Once all edges have been processed, the routes are sorted by their rewards, from highest to lowest, and only the routes necessary

to meet the vehicle requirements are retained. The nodes that can be visited are updated, and the solution is added to the global solution. The process is repeated until all days are completed, when the final solution is returned (lines 18 to 24).

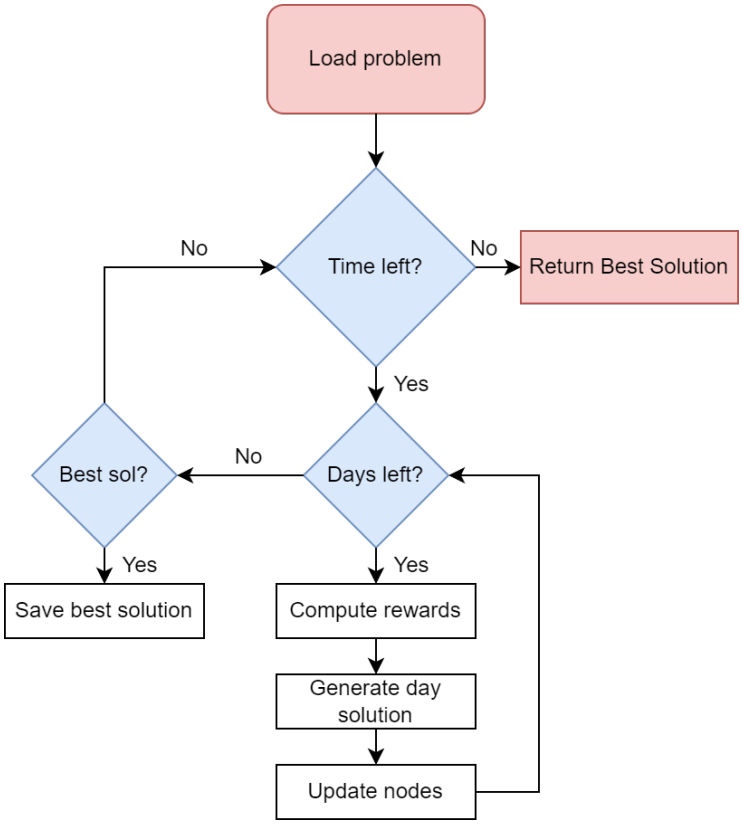

Figure 2. Schematic approach for solving the multi-period TOP with the biased-randomized heuristic.

Instead of manually fine-tuning the optimal value of α for each problem to obtain the savings shown in Equation (11), a function is created to find the best value automatically, as outlined in Algorithm 2. Initially, the best value of α is set to zero and the best reward to minus infinity (lines 1 and 2). Then, for each possible value of α to be tested, a solution is computed using the non-biased version of the heuristic (lines 3 and 4). If the solution's reward with the current α is better than the previously recorded best reward, both the best reward and the best value of α are updated accordingly (lines 5 to 7). Finally, the best value of α is returned.

Since the value of α is used to compute a linear combination, it should take a value between 0 and 1, where 0 means all weight is given to cost, and 1 means all weight is given to rewards. In this case, ten equidistant values of α are tested from 0 to 0.95. These specific values are chosen to avoid selecting one, which might lead to non-optimal results given the complexity of the problem and the significant importance of rewards in this problem. Finally, Algorithm 3 outlines the main loop of the approach.

Algorithm 1 Heuristic algorithm

```
globalSol ← GlobalSolution()
for day ∈ days do
    computeRewards(inputs, day)
    daySol ← dummyDaySolution(inputs)
    savingsList ← sortedSavingsList(inputs, α)
    while savingsList ≠ ∅ do
        ijEdge ← selectEdge(savingsList)
        iRoute ← getRoute(i)
        jRoute ← getRoute(j)
        isMergePossible ← checkMergeConditions(iRoute, jRoute)
        if isMergePossible then
            newRoute ← mergeRoutes(iRoute, jRoute)
            deleteRoute(daySol, iRoute)
            deleteRoute(daySol, jRoute)
            addRoute(daySol, newRoute)
        end if
        deleteEdge(savingsList, ijEdge)
    end while
    sortRoutesByReward(daySol)
    deleteRoutesByReward(daySol, numberVehicles)
    addDaySolution(globalSol, daySol)
    updateInputs(inputs, daySol)
end for
return globalSol
```

Algorithm 2 Find best value of α

```
bestα ← 0
bestReward ← −∞
for α ∈ possibleValues do
    sol ← heuristic(input, α)
    if reward(sol) > bestReward then
        bestReward ← reward(sol)
        bestα ← α
    end if
end for
return bestα
```

Algorithm 3 Main loop

```
initSol ← heuristic(inputs, α)
bestSol ← initSol
β ← 0.5
bestβ ← β
time ← 0
while time ≤ maxTime do
    newSol ← heuristicBR(inputs, α, β)
    if reward(newSol) > reward(bestSol) then
        bestSol ← newSol
        bestβ ← β
    end if
    β ← randomTriangular(bestβ)
    updateTime(time)
end while
return bestSol
```

First, Algorithm 3 receives the problem as an input and the best value of α determined by Algorithm 2. An initial solution is generated using the non-biased-randomized heuristic and is assigned as the best solution (lines 1 and 2). Additionally, β and best β, the parameters required for the biased randomization, are initialized, along with the time counter, which is set to 0 (lines 3 to 5). While β is the current parameter used in the biased randomization, best β is the one that has produced the highest reward and is used in the self-tuning part of the algorithm. Moreover, since β varies between 0 and 1, it is initially set to 0.5 and then it is adjusted automatically in order to find the best possible value. While the maximum time has not been exceeded, the algorithm computes a solution using the biased randomization process described in [10] with the current β. If this solution's reward surpasses the best solution's reward so far, both the best solution and best β are updated (lines 6 to 11). Subsequently, β is updated to a new value drawn randomly from a triangular distribution with a mode equal to best β (line 12). This ensures that β values are close to the best β ones while also allowing the exploration of the entire search space of the parameter. The elapsed time since the start of the loop is also updated each iteration and, once the maximum time is reached, the loop terminates, and the best solution found is returned (lines 13 to 15).

Each day, the reward for each node is computed as shown in Equation (10): it is the summation over the day and the following days of the energy production, multiplied by the energy price and the bonus. However, if the reward is computed this way preventive maintenance tends to be performed on the second day, as the reward is higher at that time, without accounting for the fact that the reward might be highest on the third day. To address this issue, there is a 33% chance that the reward for the second day is randomly reduced by 85% to 100%. This adjustment encourages the algorithm to consider performing the maintenance on the last day, while still keeping the option of doing it on the second day in case it cannot be done on the last one. This step is not performed in Algorithm 2 when determining the optimal value of α, to avoid the randomness in the rewards. Once the problem is solved, the objective function is calculated to get the final reward.

5. Reinforcement Learning Approach

For the RL approach, a constructive strategy has been implemented. Each day, the algorithm addresses the problem by sequentially building the route for each vehicle. The process begins with the vehicle starting at the depot. The RL algorithm then determines which nodes can be visited by applying masks, taking into account factors like time windows for preventive maintenance or avoiding already-visited nodes. Based on the current context and these masked nodes, the algorithm selects the next node to visit. This process repeats until the available time runs out, at which point the vehicle returns to the depot. This process is repeated for each vehicle, and once all routes are completed, the algorithm moves on to the next day. The flowchart in Figure 3 illustrates the structure of the algorithm.

The problem is first loaded, and all necessary data are generated to initiate the process. The data from different types of maintenance and the depot are then processed through graph embeddings, consisting of learnable linear functions. This embedded data are passed to the encoder, which is based on the architecture used in transformer models. The encoder comprises n_{enc} layers, each featuring batch normalization, residual connections, multi-head attention, and feed-forward networks. For each day and vehicle, a route is constructed through a series of steps:

- Update state: The current state of the problem is updated to reflect the vehicle's progress and the current situation.
- Context and mask generation: Relevant data are generated to provide context for the algorithm, and a mask is created to know which nodes can be visited.
- Decoding: The context, mask, and initial embeddings are passed through the decoder, which produces a probability distribution that indicates the likelihood of visiting each node.

- Node selection: The next node is selected based on the probability distribution. This process repeats until the route is complete.

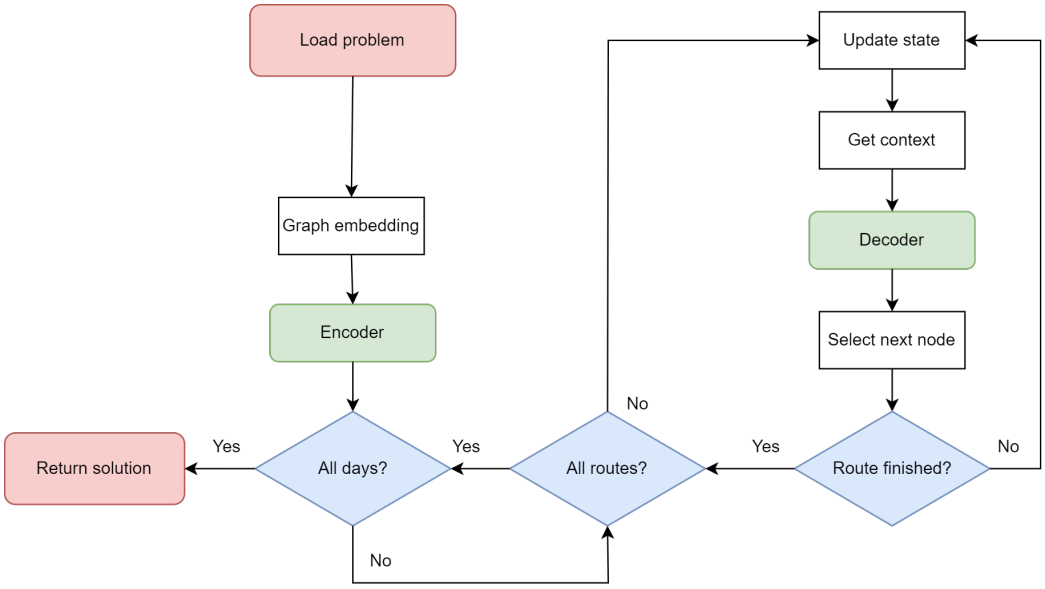

Figure 3. Overall structure of the RL algorithm.

Nodes are assigned a probability of 0, preventing them from being visited if they: (i) have already been visited; (ii) do not fit within the time windows (applicable for preventive maintenance); and (iii) would result in the vehicle exceeding the allowed time to return to the depot (taking into account travel and maintenance time). This ensures that each vehicle can complete its route within the day's time constraints.

5.1. Encoder–Decoder

As has been explained before, a common approach is to use an encoder–decoder architecture. Figure 4 illustrates the encoder component of this model. Initially, the different characteristics of the three types of maintenance (corrective, preventive, and predictive) are embedded using learnable linear functions. The data related to the depot are also included and processed through the transformer encoder.

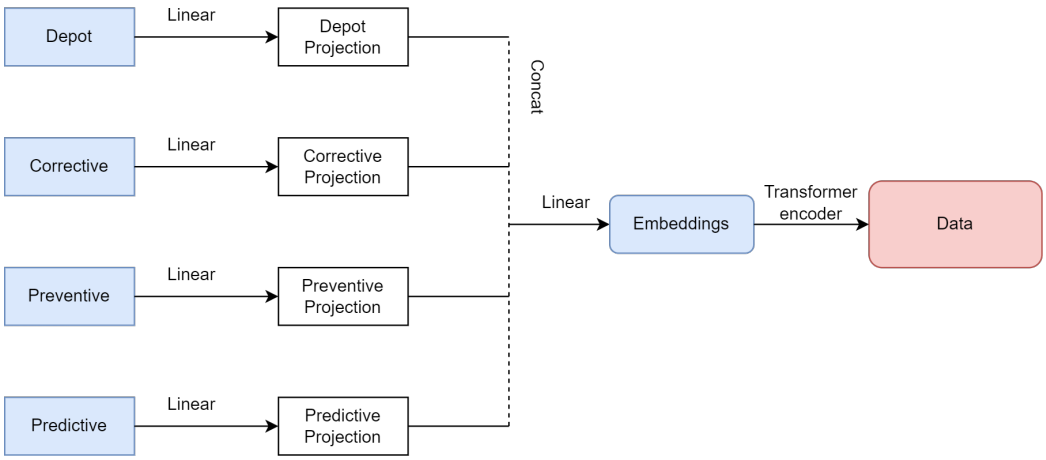

Figure 4. Schema of the encoder component.

The embedded features, denoted as $h_i^{(0)}$, $\hat{h}_i^{(0)}$, and $\tilde{h}_i^{(0)}$, correspond to the embedded features of each maintenance type: depot and corrective, preventive, and predictive maintenance, respectively. Together, these represent the entire graph instance. Effective feature

mapping in this context requires understanding the relationships between features in the graph. For example, a node's representation needs to include enough information to be chosen over its neighboring nodes in a specific situation. To capture these feature relationships, multi-head self-attention is used, which creates more detailed feature embeddings. This self-attention mechanism enables the encoder to evaluate and prioritize the importance of different features within the input graph in each situation. The data utilized are as follows:

- x_i, y_i represent the coordinates of maintenance i;
- t_i denotes the duration of maintenance i;
- p_i indicates the energy production of the facility associated with maintenance i;
- $\hat{t}_i^{start}, \hat{t}_i^{end}$ define the start and end of the time window for preventive maintenance i;
- π_i^k represents the probability of failure for asset i on day k, relevant for predictive maintenance.

A hat above a variable signifies it pertains to preventive maintenance, while a tilde indicates it relates to predictive maintenance. Suppose there are N corrective maintenances, M preventive maintenances and S predictive maintenances. Then, the encoding steps are formally defined as follows, where all the matrices are learneable parameters and produce an output of $d_k = 128$ values. Increasing this parameter did not yield better results.

$$h_0^{(0)} = [x_0, y_0] W_0^{emb} \tag{12}$$

$$h_i^{(0)} = [x_i, y_i, t_i, p_i] W_1^{emb} \quad \forall i \in \{1, 2, \ldots, N\} \tag{13}$$

$$\hat{h}_i^{(0)} = \left[\hat{x}_i, \hat{y}_i, \hat{t}_i, \hat{p}_i, \hat{t}_i^{start}, \hat{t}_i^{end}\right] \hat{W}_0^{emb} \quad \forall i \in \{1, 2, \ldots, M\} \tag{14}$$

$$\tilde{h}_i^{(0)} = \left[\tilde{x}_i, \tilde{y}_i, \tilde{t}_i, \tilde{p}_i, \pi_i^0, \pi_i^1, \pi_i^2, \pi_i^3, \pi_i^4\right] \tilde{W}_0^{emb} \quad \forall i \in \{1, 2, \ldots, S\} \tag{15}$$

$$h^{(0)} = \left[h_0^{(0)}, h_1^{(0)}, \ldots, h_N^{(0)}, \hat{h}_1^{(0)}, \ldots, \hat{h}_M^{(0)}, \hat{h}_1^{(0)}, \ldots, \hat{h}_S^{(0)}\right] W^{emb} \tag{16}$$

The encoder follows the same architecture as a transformer, comprising n_{enc} layers, each consisting of multi-head attention, a feed-forward network, batch normalization, and residual connections. Let $h^{(l)}$ represent the data produced by the encoder at layer $l-1$, and $h^{(0)}$ denote the initial data obtained from the graph embedding. The encoder's input is initially set to $h^{(0)}$. In each layer l, $h^{(l)}$ passes through the multi-head attention mechanism, where keys, queries, and values are computed for each head:

$$Q_{l,h} = h^{(l)} W_{l,h}^Q \quad K_{l,h} = h^{(l)} W_{l,h}^K \quad V_{l,h} = h^{(l)} W_{l,h}^V \tag{17}$$

All the matrices involved are also learnable parameters, and each produces an output of dimension d_k/H, where H is the number of heads in the multi-head attention layer. In this implementation, $H = 8$, as H must be an integer that divides d_k and is used to have a balance between the number of heads and the output dimension. Changing this value did not improve results.

$$Z_l^h = Softmax\left(\frac{Q_{l,h} K_{l,h}^T}{\sqrt{d_k}}\right) V_{l,h} \tag{18}$$

Finally, the multi-head attention output is obtained by concatenating the attention results from each head:

$$MHA^l\left(h^{(l)}\right) = \left[Z_l^1, Z_l^2, \ldots, Z_l^H\right] W_l^{out} \tag{19}$$

The result produced by the attention mechanism goes through a batch normalization and residual connection layer, obtaining $\hat{h}^{(l)}$:

$$\hat{h}^{(l)} = BN^l\left(MHA^l\left(h^{(l)} + h^{(l)}\right)\right) \tag{20}$$

Finally, $\hat{h}^{(l)}$ passes through an additional layer consisting of batch normalization, residual connections, and a feed-forward network. The feed-forward network is composed of a linear transformation followed by a ReLU activation function, and then another linear transformation. The ReLU function, defined as $f(x) = \max(0, x)$, is extensively used to introduce non-linearity into neural networks. However, despite the existence of numerous activation functions, as shown by Nguyen et al. [42], ReLU is one of the most popular due to its simplicity and effectiveness.

$$h^{(l+1)} = BN^l\left(FF^l\left(\hat{h}^{(l)}\right) + \hat{h}^{(l)}\right) = BN^l\left(W_{l,1}^{ff} ReLU\left(W_{l,0}^{ff}\hat{h}^{(l)}\right) + \hat{h}^{(l)}\right) \tag{21}$$

where $W_{l,0}, W_{l,1}$ are the matrices corresponding to the learneable lineal networks of the feed-forward one and $h^{(l+1)}$ represents the inputs passed to the next layer of the encoder. Equations (17) and (21) are then repeated n_{enc} times. The encoder returns $h^{(n_{enc}+1)}$ and its mean, that is:

$$\overline{h}^{(n_{enc})} = \frac{1}{1+N+M+S}\left(\sum_{i=0}^{N} h_i^{(n_{enc}+1)} + \sum_{i=1}^{M} \hat{h}_i^{(n_{enc}+1)} + \sum_{i=1}^{S} \tilde{h}_i^{(n_{enc}+1)}\right) \tag{22}$$

These encoder data are computed only once, at the beginning. Moreover, when the model selects the next node, it requires context information about the current state of the problem. Three different context data are generated:

- Graph context: This is obtained by applying a linear network to a vector that includes the mean graph embeddings $\overline{h}^{(n_{enc})}$, the mean node embeddings of the current partial route, the mean node embeddings of the nodes that have not been visited, and the node embedding of the depot. These data collectively provide the network with all necessary information related to the current state of the nodes.
- Day context: This is obtained by applying a linear network to a vector that contains the current day, the current timestep of the day, the current global timestep, and the remaining timesteps until the end of the day.
- Problem context: This includes the energy prices for each day and the number of timesteps in each day.

Figure 5 illustrates the steps of the decoder. First, a multi-head attention mechanism is applied between the current node embeddings and the graph context. This allows the neural network to understand the vehicle's current location and the available nodes. This process follows Equations (17) to (19), but instead of using the same vector for keys, queries, and values (as in self-attention), those vectors mentioned earlier are employed for keys, queries, and values. Next, the route context is obtained and a multi-head attention is performed with the day context to determine which node might be the best choice considering the current day and timestep. The result of this is then used in another multi-head attention operation with the problem context to provide a broader context about the problem. Finally, a single-head attention (multi-head attention with $H = 1$) is computed using the node embeddings produced by the encoder, which produces the final logits.

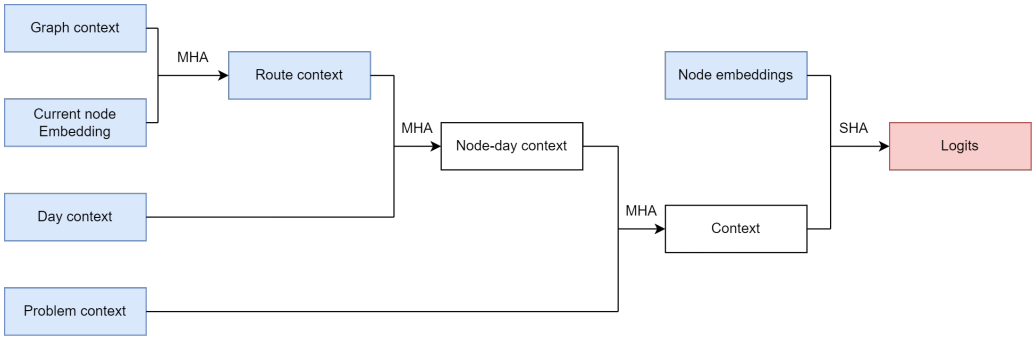

Figure 5. Schema of the decoder component.

Once the logits are obtained, the mask is applied, setting the logits whose nodes can not be visited to minus infinity, which ensures that Equation (23) returns 0. Then, the final probabilities are computed as shown in Equation (23), where C is a learneable parameter:

$$p_t = Softmax(C \cdot tanh(logits)) \tag{23}$$

5.2. Training

The solution obtained by the algorithm is a permutation $\pi = (\pi_1, \pi_2, \ldots)$ of a subset of the nodes, where only the depot can be repeated and not all nodes need to be visited. Policy-gradient methods learn the policy directly through gradient-based optimization. In this context, a stochastic policy $p(\pi|s)$ is defined for selecting a solution π given a problem s. This policy is factorized and parameterized by θ as follows:

$$p_\theta(\pi|s) = \prod_{t=1}^{N} p_\theta(\pi_t|s, \pi_{1:t-1}) \tag{24}$$

Williams [43] proposed a estimator for the policy gradient using Monte Carlo sampling, assuming that the rewards are independent of θ. This method is known as the REINFORCE algorithm:

$$\nabla_\theta \mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[R(\pi)\nabla_\theta \log p_\theta(\pi|s)] \tag{25}$$

However, this method suffers from high variance, which can make it difficult to train models effectively. To address this issue, a baseline value is introduced in Equation (25) to reduce variance, leading to improved training stability and better results:

$$\nabla_\theta \mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(R(\pi) - b(s))\nabla_\theta \log p_\theta(\pi|s)] \tag{26}$$

For the training phase, the process is divided into epochs, each consisting of 2000 training steps with a batch size of 256, for speed and memory limitations. At the end of each epoch, the model is evaluated using 100,000 randomly generated validation problems. This extensive validation set is used to account for the problem's difficulty and variability, which aims to reflect the model's performance as accurately as possible. If the mean reward from the validation set exceeds that of the previous best model, a t-test with $\alpha = 0.05$ is conducted to verify that the new model is statistically better. If the new model is confirmed to be superior, it replaces the previous best model. Additionally, the training phase tracks the mean reward over the last 200 training steps. If this mean reward improves or is within 0.3% of the mean reward of the best model, the training model is also tested against the validation data to potentially accelerate convergence. Furthermore, if the best model is updated, new validation data are generated to prevent overfitting.

For the training hyperparameters, a learning rate of 10^{-4} was employed, along with the Adam optimizer proposed by Kingma and Ba [44]. Higher learning rates were also tested, but the model failed to converge. We found that with a learning rate of 10^{-4} and the Adam optimizer, the training phase behaved as expected. Additionally, the choice of the

baseline for training is important, as the final results can significantly depend on it. In this case, the baseline is defined as follows: (i) the mean reward of the training data obtained by the training model during the first epoch; and (ii) the mean reward of the training data achieved by the best found model in other case, allowing the model to compete against its own best performance. Since the model generates a probability distribution over the nodes at each step, three inference methods are utilized to derive the solution:

- Greedy: The node with the highest probability is selected and visited;
- Sampling: A node is chosen randomly according to the probability distribution;
- Augmentation: Proposed by Kwon et al. [45], this method generates eight equivalent problem instances (they have an equivalent solution) from each original problem. Since the positional data passed to the network ranges from 0 to 1, there are 8 transformations that preserve distances between nodes, effectively producing equivalent solutions. These problems are then solved in a greedy way. The transformations are displayed in Figure 6.

$$\begin{array}{c|c} (x,y) & (y,x) \\ (1-x,y) & (y,1-x) \\ (x,1-y) & (1-y,x) \\ (1-x,1-y) & (1-y,1-x) \end{array}$$

Figure 6. Transformations used for generating equivalent problem instances.

The results obtained with the model for the baseline utilize the greedy method, while the rewards are calculated using the sampling method from the training model. Moreover, the choice of the baseline in the REINFORCE algorithm significantly impacts the training speed of the algorithm. For example, in the work by Bello et al. [23], an actor–critic method was employed; however, later research by Kool et al. [29] found that this approach generally underperforms compared to simpler baselines. In this case, the training data and baseline used were proposed by Lee and Ahn [46], where the data are derived from augmenting a problem to generate eight different ones with equivalent solutions. According to the authors, the model trained with this augmented data can perform as well as or better than one trained on completely random data. This approach only requires generating one-eighth of the total problems, as the remaining problems are simply augmented versions of the original. This method significantly speeds up training, with improvements of approximately 20% to 30% observed in this case. Given that the batch size is 256, only 32 original problems need to be generated, since each problem yields eight equivalent ones through augmentation.

6. Computational Experiments

For the computational experiments, the model was trained for 100 epochs using the parameters previously mentioned. Inference was performed on a Chinese-made personal laptop (HP Omen 15-en1004ns) with 16 GB of RAM, an AMD Ryzen 5800H processor, and a mobile Nvidia RTX 3070 GPU. The model was trained on randomly generated problems consisting of 45 maintenance tasks: 40 predictive, 2 corrective, and 3 preventive. For inference, four methods were tested, three of which were described in Section 5.2: the greedy (argmax) method, the sampling method, and augmentation. Additionally, this paper introduces a new inference method. In cases where vehicles start and end at different nodes, augmenting solutions and also permuting vehicle routes can improve results, as shown by Lee and Ahn [46]. However, since this problem requires vehicles to start and end at the same node, this approach is not applicable. Instead, we employed a similar approach to augmentation by creating equivalent versions of the original problem and selecting the solution with the highest reward. In this case, before applying the augmentation, we first performed node coordinate rotations to generate equivalent problems. This is because the distance between each pair of nodes remains unchanged. The nodes were rotated at equal intervals from 0 to $\pi/2$ degrees, with a total of 32 rotations. After applying the

8 augmentations for each rotation, 256 equivalent problems were created, and the solution with the highest greedy reward was chosen. Limiting the rotation range from 0 to $\pi/2$ prevents duplicate problems after augmentation.

The reinforcement learning model's performance was compared to the heuristic approach on 256 randomly generated instances with the same number of nodes and maintenance types as used in the training phase of the model, allowing the heuristic 0.5, 2, and 120 s to solve each problem. Meanwhile, the four inference methods mentioned earlier were tested for the RL model. The sampling method solved each time 256 equivalent problems, while the rotation method used 32 rotations, as increasing the number of rotations past this point did not yield better results. As shown in Table 1, the method that combines rotations and augmentations obtains the best results for the reinforcement learning model, achieving performance comparable to the heuristic with 2 s of computation time while using fewer computational resources. Usually, the model takes approximately 0.7 s to get the solution for each instance using this method. The percentage of corrective and preventive maintenances performed by both the heuristic and RL models is also presented. It is important to note that not all preventive maintenances have to be completed, as the algorithms may determine that it is better to postpone them to the following week.

Table 1. Mean results of the RL and heuristic method in 256 random generated instances.

		Reward	**Corr. Maint. (%)**	**Prev. Maint. (%)**
Heuristic	0.5 s	19.42	100%	76.04%
	2 s	19.55	100%	76.69%
	120 s	19.65	100%	76.43%
Reinforcement Learning Model	Greedy	19.09	100%	76.17%
	Sampling	19.43	100%	77.08%
	Augment.	19.50	100%	76.69%
	Rot. + Augment.	19.56	100%	76.30%

However, although the mean reward of the reinforcement learning model is approximately the same as that of the 2-s heuristic, the heuristic outperforms the model on a greater number of individual problems, producing a better reward in 57% of the cases. This suggests that the reinforcement learning model is better at scheduling high-reward corrective and preventive maintenances within the given timeframe. When both methods successfully schedule these high-priority maintenances, the heuristic tends to have a slight advantage. For instance, Figure 7 presents the reward distribution across the 256 instances, demonstrating that the RL model is more effective at scheduling high-reward maintenance tasks.

In Table 2, the heuristic and the RL model are tested under different conditions. The comparison involved the RL model using the greedy, sampling, and rotation with augmentation methods, compared against the heuristic method, allowing it 2 s to solve each problem, which is the time where the performance of both methods was similar. In total, 1536 randomly generated instances were solved, with 256 instances for each configuration of nodes and maintenance types. Three different numbers of maintenance tasks were used: 30, 45, and 75, representing small, medium, and large problem sizes, respectively. The model remains the same as before, trained on a medium configuration with instances consisting of 40 predictive, 2 corrective, and 3 preventive maintenance tasks. In this context, N represents the number of corrective maintenance operations, M the number of preventive maintenance operations, and S the number of predictive maintenance operations. The rows marked with a star indicate that the number of preventive and predictive maintenance operations matches the numbers used during the model's training phase. The heuristic shows better scalability with varying numbers of nodes and maintenance tasks. The RL model performs well under the same conditions used during the training phase and slightly underperforms compared to the heuristic when conditions change slightly. Moreover, the method that combines rotations and augmentations typically yields the best results for the

RL model. However, when the number of nodes and maintenance types is significantly different, the heuristic proves to be superior.

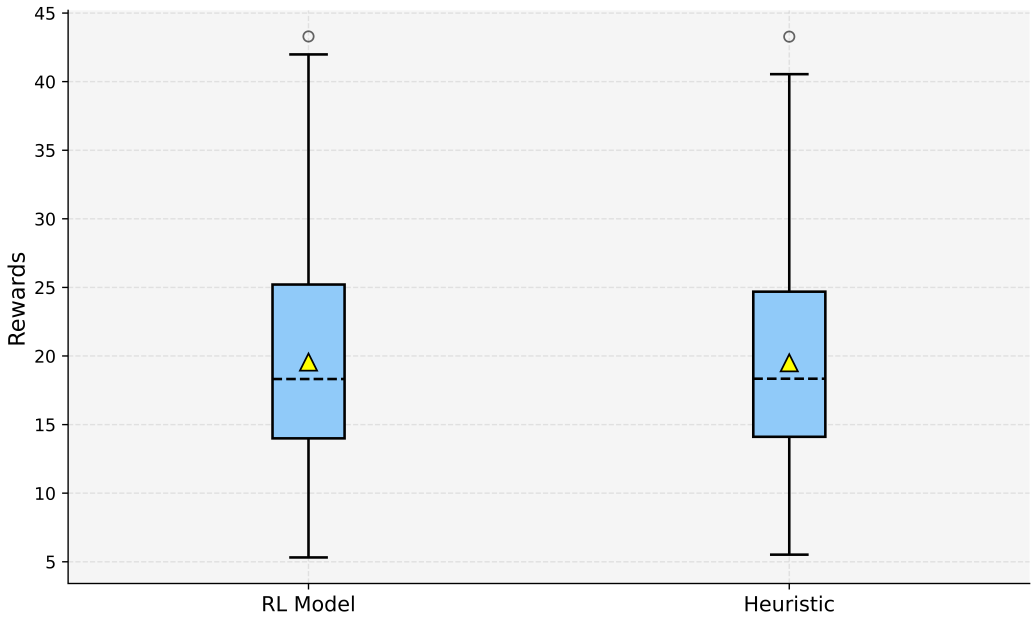

Figure 7. Distribution of the results obtained by both methods across the 256 instances. The triangle indicates the mean, while the circle represents the outliers.

Table 2. Mean results of the RL and heuristic method under different conditions. The rows marked with a star indicate that the number of preventive and predictive maintenance operations matches the numbers used during the model's training phase.

		Heuristic	**RL Model**		
		2 s	**Greedy**	**Sampling**	**Rot. + Augment.**
30 Maintenances	N = 2 * M = 3 S = 25	19.05	13.35	13.87	15.22
	N = 1 M = 2 S = 27	12.76	7.60	7.84	8.47
45 Maintenances	N = 2 * M = 3 S = 40	19.55	19.09	19.43	19.56
	N = 3 M = 3 S = 39	21.68	19.62	20.61	21.06
75 Maintenances	N = 2 * M = 3 S = 70	20.71	8.43	8.7	8.82
	N = 3 M = 5 S = 67	29.81	10.23	10.54	10.44

One drawback of the RL model is its lengthy training time; in this case, training the RL model for 100 epochs took 2 days. Although performance improvements plateau beyond this point, it is possible that the model had not yet fully converged. Additionally, the RL model requires a large amount of data for training. However, a key advantage of RL is

that the exact solution for each problem in the dataset does not need to be known, as the model continuously tries to improve the reward it receives. Another strength is that it does not require real-world data, since it can be trained on randomly generated problems that follow the same logic as real-world scenarios, but generating this data and ensuring the RL model performs as expected can be challenging.

Although additional work is needed to improve the RL model's performance under varying conditions, it can achieve performance and efficiency comparable to that of a heuristic model. In this study, the RL model outperformed the heuristic when computational time was constrained, which means that RL can be suitable for scenarios where low computational time is crucial and immediate solutions are required, making it ideal for real-time problem solving. Future research could also focus on integrating real-time data to further improve the model's effectiveness on handling unpredictable changes.

7. Conclusions

This paper investigates the problem of optimizing maintenance schedules and routes for power generation assets over multiple days. It models this challenge as a multi-period team orienteering problem. The optimization problem involves multiple assets across various locations, different types of maintenance (corrective, preventive, and predictive), and a need to maximize profits while adhering to time and resource constraints. For solving the problem, it introduces a dual approach: a biased-randomized heuristic algorithm and a reinforcement learning framework. The heuristic approach is a savings-based method with a biased-randomized technique. This approach adapts a geometric probability distribution and includes a self-tuning mechanism. The process involves generating multiple solutions within a set time frame, evaluating each one, and updating the best solution if a better one is found. For each day in the planning period, the algorithm computes rewards, solves the problem, and updates node visitation options. The savings for each node pair are calculated and used to create and merge routes. A function automatically finds the best parameter values for the savings calculation, which balances cost and rewards. Additionally, to handle potential reward variability, there is a random adjustment to encourage maintenance scheduling flexibility. The RL approach constructs routes for each vehicle starting from the depot, selecting the next node to visit based on a probability distribution, which is influenced by time windows, visited nodes, and the vehicle's remaining time. The RL model uses an encoder–decoder architecture with transformer-like layers to embed data, calculate contexts, and generate probabilities for node selection. The training of this model employs policy-gradient methods with techniques like Monte Carlo sampling and baselines to reduce variance and improve stability. The model is trained over multiple epochs with a large validation set to ensure robustness. During inference, different methods, including greedy, sampling, and augmentation, are used to select the best routes. The performance of the RL model is evaluated and compared with a heuristic approach on randomly generated test instances of different sizes and conditions.

The computational experiments involved training a model for 100 epochs on a laptop with an Nvidia RTX 3070 GPU (Santa Clara, CA, USA). The model, tested on 45 maintenance tasks, evaluated four inference methods: greedy, sampling, augmentation, and a new rotation-based method. The rotation method involved generating 256 equivalent problems through node coordinate rotations and was found to yield the best results, closely matching the heuristic's performance with 2 s of computation time. Comparisons between the RL model and the heuristic method showed that while the RL model's average reward was similar to the heuristic's with 2 s of computation, the heuristic performed better on more individual problems. The RL model was particularly strong in scheduling high-reward maintenance tasks, but the heuristic demonstrated superior scalability and overall performance in different problem conditions, especially when applying the model to solve instances that differ substantially from those encountered during training. This highlights the need for more research to ensure the scalability of the RL model. Additionally, RL models require advanced hardware with significant computational power, and the vast

amount of training data needed often rely on randomly generated instances that mimic real-world conditions, making validation against actual scenarios more difficult. Nevertheless, the RL method was trained successfully and achieved results comparable to those of the heuristics, as found by other researchers, effectively meeting our initial goal.

These findings have important implications for city logistics planning and design, as the optimization of maintenance schedules for power generation assets might affect various city services. The approaches developed in this study, including the heuristic and RL methods, could be adapted to improve the efficiency of scheduling and routing for other city maintenance tasks, such as waste collection, infrastructure repair, and emergency response. The following research lines can be considered for future work: (i) enhancing the performance of RL models under diverse and dynamic conditions, including better adaptation to varying numbers of maintenance tasks and nodes; (ii) integrating real-time data and adaptive mechanisms to handle unpredictable changes in maintenance schedules and resources; (iii) validating the results in real-world scenarios; and (iv) exploring hybrid approaches that combine the strengths of heuristic and machine learning methods could lead to more robust and scalable solutions.

Author Contributions: Conceptualization, L.P.-R. and A.G.-S.; methodology, A.G. and A.A.J.; software, A.G.; writing—original draft preparation, A.G. and A.A.J.; writing—review and editing, A.G.-S. and L.P.-R.; supervision, A.A.J. and A.G.-S. All authors have read and agreed to the published version of the manuscript.

Funding: The present work was carried out as part of the IA4TES project "Artificial Intelligence for Sustainable Energy Transition". The project belongs to the "Misiones de I+D en Inteligencia Artificial 2021" program funded by the Spanish Government through the "Plan de Recuperación". This research has also been partially funded by the Spanish Ministry of Science (PID2022-138860NB-I00 and RED2022-134703-T).

Data Availability Statement: Restrictions apply to the availability of these data.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Fontaine, P.; Minner, S.; Schiffer, M. Smart and sustainable city logistics: Design, consolidation, and regulation. *Eur. J. Oper. Res.* **2023**, *307*, 1071–1084. [CrossRef]
2. Pinciroli, L.; Baraldi, P.; Zio, E. Maintenance optimization in industry 4.0. *Reliab. Eng. Syst. Saf.* **2023**, *234*, 109204. [CrossRef]
3. Molęda, M.; Małysiak-Mrozek, B.; Ding, W.; Sunderam, V.; Mrozek, D. From corrective to predictive maintenance—A review of maintenance approaches for the power industry. *Sensors* **2023**, *23*, 5970. [CrossRef]
4. Mirhosseini, M.; Keynia, F. Asset management and maintenance programming for power distribution systems: A review. *IET Gener. Transm. Distrib.* **2021**, *15*, 2287–2297. [CrossRef]
5. Li, Y.F.; Valla, S.; Zio, E. Reliability assessment of generic geared wind turbines by GTST-MLD model and Monte Carlo simulation. *Renew. Energy* **2015**, *83*, 222–233. [CrossRef]
6. Ayu, K.; Yunusa-Kaltungo, A. A holistic framework for supporting maintenance and asset management life cycle decisions for power systems. *Energies* **2020**, *13*, 1937. [CrossRef]
7. Ferrari, A.; Mangano, G.; Cagliano, A.C.; De Marco, A. 4.0 technologies in city logistics: an empirical investigation of contextual factors. *Oper. Manag. Res.* **2023**, *16*, 345–362. [CrossRef]
8. Panadero, J.; Juan, A.A.; Ghorbani, E.; Faulin, J.; Pagès-Bernaus, A. Solving the stochastic team orienteering problem: comparing simheuristics with the sample average approximation method. *Int. Trans. Oper. Res.* **2024**, *31*, 3036–3060. [CrossRef]
9. Juan, A.A.; Keenan, P.; Martí, R.; McGarraghy, S.; Panadero, J.; Carroll, P.; Oliva, D. A review of the role of heuristics in stochastic optimisation: From metaheuristics to learnheuristics. *Ann. Oper. Res.* **2023**, *320*, 831–861. [CrossRef]
10. Grasas, A.; Juan, A.A.; Faulin, J.; De Armas, J.; Ramalhinho, H. Biased randomization of heuristics using skewed probability distributions: A survey and some applications. *Comput. Ind. Eng.* **2017**, *110*, 216–228. [CrossRef]
11. Pinciroli, L.; Baraldi, P.; Ballabio, G.; Compare, M.; Zio, E. Optimization of the operation and maintenance of renewable energy systems by deep reinforcement learning. *Renew. Energy* **2022**, *183*, 752–763. [CrossRef]
12. Dong, W.; Zhao, T.; Wu, Y. Deep reinforcement learning based preventive maintenance for wind turbines. In Proceedings of the 2021 IEEE 5th Conference on Energy Internet and Energy System Integration (EI2), Taiyuan, China, 22–24 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 2860–2865.
13. Ilić, D.; Žarković, M. Machine learning for power generator condition assessment. *Electr. Eng.* **2024**, *106*, 2691–2703. [CrossRef]
14. Chao, I.M.; Golden, B.L.; Wasil, E.A. The team orienteering problem. *Eur. J. Oper. Res.* **1996**, *88*, 464–474. [CrossRef]

15. Vansteenwegen, P.; Souffriau, W.; Van Oudheusden, D. The orienteering problem: A survey. *Eur. J. Oper. Res.* **2011**, *209*, 1–10. [CrossRef]
16. Tricoire, F.; Romauch, M.; Doerner, K.F.; Hartl, R.F. Heuristics for the multi-period orienteering problem with multiple time windows. *Comput. Oper. Res.* **2010**, *37*, 351–367. [CrossRef]
17. Zucchi, G.; Corrêa, V.H.; Santos, A.G.; Iori, M.; Yagiura, M. A Metaheuristic Algorithm for a Multi-period Orienteering Problem arising in a Car Patrolling Application. In Proceedings of the INOC, Aachen, Germany, 7–10 June 2022; pp. 1–6.
18. Ferreira, J.; Quintas, A.; Oliveira, J.A.; Pereira, G.A.; Dias, L. Solving the team orienteering problem: developing a solution tool using a genetic algorithm approach. In Proceedings of the Soft Computing in Industrial Applications: Proceedings of the 17th Online World Conference on Soft Computing in Industrial Applications, Online, 10–21 December 2012; Springer: Berlin/Heidelberg, Germany, 2014; pp. 365–375.
19. Dang, D.C.; Guibadj, R.N.; Moukrim, A. An effective PSO-inspired algorithm for the team orienteering problem. *Eur. J. Oper. Res.* **2013**, *229*, 332–344. [CrossRef]
20. Wen, H.; Wang, S.X.; Lu, F.Q.; Feng, M.; Wang, L.Z.; Xiong, J.K.; Si, M.C. Colony search optimization algorithm using global optimization. *J. Supercomput.* **2022**, *78*, 6567–6611. [CrossRef]
21. Wen, H.; Song, S.; Peng, Z.; Guo, X.; Xu, C.; Wang, Y.; Huo, L.; Cui, J.H. Power Control Based Energy-Efficient Deployment for Underwater Wireless Sensor Networks With Asymmetric Links. *IEEE Internet Things J.* **2024**, *11*, 29742–29756. [CrossRef]
22. Vincent, F.Y.; Salsabila, N.Y.; Lin, S.W.; Gunawan, A. Simulated annealing with reinforcement learning for the set team orienteering problem with time windows. *Expert Syst. Appl.* **2024**, *238*, 121996.
23. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv* **2016**, arXiv:1611.09940.
24. Hochreiter, S. *Long Short-Term Memory*; Neural Computation MIT-Press: Cambridge, MA, USA, 1997.
25. Nazari, M.; Oroojlooy, A.; Snyder, L.; Takác, M. Reinforcement learning for solving the vehicle routing problem. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 9839–9849 .
26. Salehinejad, H.; Sankar, S.; Barfett, J.; Colak, E.; Valaee, S. Recent advances in recurrent neural networks. *arXiv* **2017**, arXiv:1801.01078.
27. Joshi, C.K.; Laurent, T.; Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv* **2019**, arXiv:1906.01227.
28. Vaswani, A. Attention is all you need. *arXiv* **2017**, arXiv:1706.03762.
29. Kool, W.; Van Hoof, H.; Welling, M. Attention, learn to solve routing problems! *arXiv* **2018**, arXiv:1803.08475.
30. Gao, H.; Zhou, X.; Xu, X.; Lan, Y.; Xiao, Y. AMARL: An attention-based multiagent reinforcement learning approach to the min-max multiple traveling salesmen problem. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *35*, 9758–9772. [CrossRef]
31. Cao, Y.; Sun, Z.; Sartoretti, G. Dan: Decentralized attention-based neural network for the minmax multiple traveling salesman problem. In Proceedings of the International Symposium on Distributed Autonomous Robotic Systems, Montbéliard, France, 28–30 November 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 202–215.
32. Zhang, R.; Zhang, C.; Cao, Z.; Song, W.; Tan, P.S.; Zhang, J.; Wen, B.; Dauwels, J. Learning to solve multiple-TSP with time window and rejections via deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 1325–1336. [CrossRef]
33. Zong, Z.; Tong, X.; Zheng, M.; Li, Y. Reinforcement Learning for Solving Multiple Vehicle Routing Problem with Time Window. *ACM Trans. Intell. Syst. Technol.* **2024**, *15*, 1–19. [CrossRef]
34. Min, Y.; Bai, Y.; Gomes, C.P. Unsupervised learning for solving the travelling salesman problem. *Adv. Neural Inf. Process. Syst.* **2024**, *36*.
35. Drakulic, D.; Michel, S.; Mai, F.; Sors, A.; Andreoli, J.M. Bq-nco: Bisimulation quotienting for generalizable neural combinatorial optimization. *arXiv* **2023**, arXiv:2301.03313.
36. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*.
37. Berto, F.; Hua, C.; Park, J.; Kim, M.; Kim, H.; Son, J.; Kim, H.; Kim, J.; Park, J. RL4CO: A unified reinforcement learning for combinatorial optimization library. In Proceedings of the NeurIPS 2023 Workshop: New Frontiers in Graph Learning, New Orleans, LA, USA, 15 December 2023.
38. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
39. Ma, Y.; Li, J.; Cao, Z.; Song, W.; Zhang, L.; Chen, Z.; Tang, J. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 11096–11107.
40. Ma, Y.; Li, J.; Cao, Z.; Song, W.; Guo, H.; Gong, Y.; Chee, Y.M. Efficient neural neighborhood search for pickup and delivery problems. *arXiv* **2022**, arXiv:2204.11399.
41. Panadero, J.; Currie, C.; Juan, A.; Bayliss, C. Maximizing reward from a team of surveillance drones under uncertainty conditions: a simheuristic approach. *Eur. J. Ind. Eng* **2020**, *14*, 1–23. [CrossRef]
42. Nguyen, A.; Pham, K.; Ngo, D.; Ngo, T.; Pham, L. An analysis of state-of-the-art activation functions for supervised deep neural network. In Proceedings of the 2021 International conference on system science and engineering (ICSSE), Ho Chi Minh City, Vietnam, 26–28 August 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 215–220.
43. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]

44. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
45. Kwon, Y.D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; Min, S. Pomo: Policy optimization with multiple optima for reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 21188–21198.
46. Lee, D.H.; Ahn, J. Multi-start team orienteering problem for UAS mission re-planning with data-efficient deep reinforcement learning. *Appl. Intell.* **2024**, *54*, 4467–4489. [CrossRef]