information MDPI

Article

Generally Applicable Q-Table Compression Method and Its Application for Constrained Stochastic Graph Traversal Optimization Problems

Tamás Kegyes [1,2,†], Alex Kummer [1,†], Zoltán Süle [2,†] and János Abonyi [1,*,†]

[1] HUN-REN-PE Complex Systems Monitoring Research Group, University of Pannonia, Egyetem Utca 10, 8200 Veszprém, Hungary; kegyes.tamas@mik.uni-pannon.hu (T.K.); kummer.alex@mk.uni-pannon.hu (A.K.)
[2] Department of Computer Science and Systems Technology, University of Pannonia, Egyetem Utca 10, 8200 Veszprém, Hungary; sule.zoltan@mik.uni-pannon.hu
* Correspondence: janos@abonyilab.com
† These authors contributed equally to this work.

Abstract: We analyzed a special class of graph traversal problems, where the distances are stochastic, and the agent is restricted to take a limited range in one go. We showed that both constrained shortest Hamiltonian pathfinding problems and disassembly line balancing problems belong to the class of constrained shortest pathfinding problems, which can be represented as mixed-integer optimization problems. Reinforcement learning (RL) methods have proven their efficiency in multiple complex problems. However, researchers concluded that the learning time increases radically by growing the state- and action spaces. In continuous cases, approximation techniques are used, but these methods have several limitations in mixed-integer searching spaces. We present the Q-table compression method as a multistep method with dimension reduction, state fusion, and space compression techniques that project a mixed-integer optimization problem into a discrete one. The RL agent is then trained using an extended Q-value-based method to deliver a human-interpretable model for optimal action selection. Our approach was tested in selected constrained stochastic graph traversal use cases, and comparative results are shown to the simple grid-based discretization method.

Keywords: constrainedshortest path; constrained shortest Hamiltonian path; disassembly line balancing optimization; stochastic shortest path; stochastic graph-based routing; reinforcement learning; Q-compression method; mixed-integer state space

check for updates

Citation: Kegyes, T.; Kummer, A.; Süle, Z.; Abonyi, J. Generally Applicable Q-Table Compression Method and Its Application for Constrained Stochastic Graph Traversal Optimization Problems. *Information* **2024**, *15*, 193. https://doi.org/10.3390/info15040193

Academic Editor: Arkaitz Zubiaga

Received: 22 February 2024
Revised: 19 March 2024
Accepted: 30 March 2024
Published: 31 March 2024

1. Introduction

We developed a generally applicable Q-compression method for solving specific mixed-integer graph optimization problems. Our approach is based on a dynamic discrete representation of the mixed-integer state space and provides a human-interpretable solution to the curse of dimensionality issue. The efficiency of our method is demonstrated on selected use cases that belong to a common problem class of constrained stochastic graph traversal problems.

We identified a diverse set of problems that belong to the class of constrained graph traversal problems. Although the shortest pathfinding problem is a well-known combinatorial optimization problem, its parameters are difficult to define exactly [1]. It is easy to see that the optimal route problem of a truck is practically a constrained shortest pathfinding problem [2], where the constraints describe the working hours limits of the driver and the availability of parking or the fuel capacity limits [3]. Daily planning of the route of an electric delivery van is a constrained Hamiltonian pathfinding problem, where the constraints are based on the limits of battery capacity [4] and charging options [5] or battery exchange opportunities [6]. Furthermore, disassembling all components of a product after its life cycle is also a constrained graph traversal problem, where a precedence graph describes component removal dependencies, and the constraints limit the use of parallel workstations [7]. Finally, we would like to highlight that in real-life problems, these kinds of problems are rather stochastic optimization tasks than discrete [8,9].

The constrained shortest pathfinding problem (CSPP) is shown to be NP-complete, and furthermore, the Hamiltonian pathfinding problem can be transformed into an SPP in polynomial time [10]. It is also presented that disassembly optimization problems can be solved by formulating as an SPP [11]. It is easy to see that these problems originate from one common root, which is based on CSPP. Therefore, a solution for other types of problems within the problem class can be derived from a CSPP solution.

Traditional methods for the shortest pathfinding problem, such as Dijkstra's [12], Floyd-Warshall [13], and Bellman-Ford algorithms, deliver the optimal solution in a reasonable time but are not able to handle stochastic distance distributions and complex constraints. The exact mixed-integer linear or quadratic programming technique [14] can be an option in multiple cases because it supports the integration of constraints, but is sensitive to stochastic parameters and the size of the problem [15]. There are successful results in applying machine learning (ML) tools to improve the performance of branch & bound method by optimizing the plane cuts. However, the optimal strategy depends significantly on the nature of the problem, and hence, it is hard to generalize the method [16]. The genetic algorithm demonstrated its power in stochastic search problems and converges to the Pareto-optimal solution. It provides not only the best solution, but also alternatives, but requires built-in heuristics to retain cycles in the path, and its performance strongly depends on the right parameterization setup [17]. In summary, there are no general methods to solve constrained stochastic graph traversal problems, so we propose to use the reinforcement learning approach, which can handle both stochastic challenges and constraints well [18].

Reinforcement learning (RL) can be an obvious solution for sequential decision-making processes such as step-by-step pathfinding [19]. The objective function should be implemented in the reward function, as well as in violation of constraints [20]. However, by considering some stochastic effect or measurement uncertainty, additional difficulties arise: some continuous components need to be integrated into the state space, which becomes mixed discrete-continuous.

Mixed-integer programming covers a general framework for a wide range of optimization problems, such as scheduling, routing, production planning, and other graph optimization tasks [21]. As mixed-integer programs (MIPs) are hard to solve problems, their effective solvers rely on heuristics. Heuristics have variable performance in different types of problems, leading to a heavily human-supervised solution design. As the number of optimization tasks increases exponentially, there is a need for a higher level of autonomous processes for optimal decisions. Reinforcement learning (RL) can serve as a valuable tool in the development of self-optimizing solutions.

There are already several directions for applying RL techniques to solve MIPs. Deep reinforcement learning (DRL) can be used to find a feasible solution to MIPs applying the Smart Feasibility Pump method [22]. RL can be applied to determine the optimal cutting plane as a subroutine of a modern IP solver [23]. Gradient-based RL methods can be extended to use mixed-integer model predictive control (MPC) for an optimal policy approximation [24]. A general integer programming (IP) solver technique was developed to learn the large neighborhood search (LNS) policy for IP [25], but the method does not handle the stochastic constraints yet. Constrained combinatorial optimization problems can also be solved by RL techniques [26]. Moreover, stochastic shortest-path problems also have RL solutions [27]. In contrast with these solutions of mixed-integer optimization problems, we developed a new discretization approach for reducing the state-space into a purely discrete space representation by performing the DBSCAN clustering method and applied an iterative policy optimization method to that.

Our contributions are as follows.

- We define the class of constrained stochastic graph traversal problems by identifying several real-life problems that belong to that. Although the pairwise relationships of individual problems were already known, we recognized that their joint investigation is recommended.

- We present a general end-to-end process for collecting observations, creating and fine-tuning the discretization function based on DBSCAN clustering results, building a Q-table, and using it for optimal decisions. We call our framework the Q-table compression method.
- Our solution delivers a human-interpretable model without a complex pre-study of the correlation of distances or further hidden dependencies.
- We demonstrate the usability of our method in three selected use cases belonging to the problem class of constrained stochastic graph traversal problem: in a constrained stochastic shortest pathfinding problem, in a constrained stochastic Hamiltonian pathfinding problem, and in a stochastic disassembly line balancing problem. We also verify the performance of the Q compression method compared to a simple grid-based discretization method.

In Section 2, we define the class of constrained stochastic graph traversal problems and formulate the constrained shortest pathfinding problem (CSPP), the constrained Hamiltonian pathfinding problem (CHPP), and the disassembly line balancing problem (DLBP). We also show that a solution for CSPP can be used directly for CHPP and DLBP as well. In Section 3, we will present a new multistep method, called the Q-table compression method, which integrates different state-space reduction steps with a dynamic training sampling technique to deliver an adaptive policy iteration algorithm. In Section 4, we demonstrate the applicability of our algorithm to three selected use cases: a CSPP, a CHPP, and a DLBP example. In Section 5, we give a summary and conclusions, and we will describe some further research directions and open research questions.

2. Constrained Graph Traversal Problem Formulation

Due to the rapid development of technology and economics, optimization problems are increasingly being focussed on to further increase efficiency. There are numerous proven methods for finding optimal solutions for certain tasks, but there are cases where the classical methods cannot be applied directly or do not perform well. We present a special optimization problem class for which no general solution is known. We will formulate the problem and show that multiple real-world problems belong to the class.

Consider a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D})$. The vertex set is denoted by $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, while the edge set is described by $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ and finally the corresponding positive distances are in $\mathcal{D} = \{d_1, d_2, \ldots, d_m\}$. We can use the function $d : \mathcal{E} \to \mathcal{D}$ as an assignment to identify the corresponding distance of an edge: $d_i = d(e_i)$ for all $i \in 1, 2, \ldots, m$. The graph is assumed to be connected and simple in the context that there are no self-loops or multi-edges in it. The edge that connects the vertex i and the vertex j can be referred to as (v_i, v_j).

Suppose that the graph $\mathcal{G}$ has two privileged vertices: s and t are the starting vertex and the target vertex. A path is defined as a series of edges: $\mathcal{P} = (e_1, e_2, \ldots, e_k) = ((v_0, v_1), (v_1, v_2), \ldots (v_{k-1}, v_k) \subset \mathcal{E}$. The shortest pathfinding problem (SPP) is to find a path from s to t with the shortest total distance: $\min |\mathcal{P}| = \min \sum_{i=1}^{k} d_i$, where $v_0 = s$, $v_k = t$ and d_i is the distance value of the edge $e_i = (v_{i-1}, v_i)$.

We can add further limitations on the shortest pathfinding problem: assume that the traveler can cover only a limited distance in one go, hence the shortest path should be split into subroutes or ranges that do not exceed their distance limit L: $\mathcal{P} = \cup_{i=1}^{l} \mathcal{P}_i$, where $\sum_{i \in \mathcal{P}_|} d_i \leq L$ for all $j \in \{1, 2, \ldots, l\}$. It is easy to see that $|\mathcal{P}| = \sum_{j=1}^{l} |\mathcal{P}_j| = \sum_{i=1}^{k} d_i$.

An important relevance of a constrained shortest path problem in practice is the truck routing problem: in this case, the traveler should split his route into feasible ranges due to the driver’s working hours limits and parking availability or fuel capacity limits. A secondary goal is to minimize the number of ranges in the whole path. To reach this, it is mandatory to extend the objective of the classical shortest pathfinding problem, and declare the common scale for it, which can be the cost:

$$\min z = \left(c_d \sum_{j=1}^{l} |\mathcal{P}_j| + c_r l\right), \tag{1}$$

where c_d represents the distance proportional cost of the completed path and c_r covers the range cost (e.g., battery recharge cost). We refer to this problem type as the constrained shortest pathfinding problem (CSPP).

There are other problems that can be transformed into a CSPP. Consider a directed connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The Hamiltonian pathfinding problem is to find a continuous path in the graph that visits all the vertices exactly once. The problem can be reduced to a shortest pathfinding problem. We can perform a reversible transformation $\hat{t}$ on graph $\mathcal{G}$ to prepare graph $\hat{\mathcal{G}}$ as follows:

- Every possible loop-free route $v \subseteq \mathcal{V}$ declares a vertex $\hat{v}$ in the transformed graph, including $\hat{v}_0 = ()$ which represents the empty subset.
- $\hat{e}_{ij} = (\hat{v}_i, \hat{v}_j)$ is an edge in the transformed graph if and only if:
  - The path in $\mathcal{G}$ represented by $\hat{v}_i \subset \mathcal{V}$ is a sub-path of the one represented by $\hat{v}_j \subset \mathcal{V}$:
  $$\hat{v}_i \subset \hat{v}_j$$
  - The path of $\hat{v}_j$ is longer exactly by one edge than the path of $\hat{v}_i$:
  $$\hat{v}_i \cup (v_y) = \hat{v}_j, \text{ where } v_y \in \mathcal{V}$$
  - By marking with $v_x \in \mathcal{V}$ the last element of $\hat{v}_i$, the additional edge of path $\hat{v}_j$ is at its end compared to the path of $\hat{v}_j$:
  $$(v_x, v_y) \in \mathcal{E}$$
  - The initial vertex can be freely chosen from all vertex $v \in \mathcal{V}$
  $$((), (v)) \in \hat{\mathcal{E}} \text{ for all } v \in \mathcal{V}$$
- As in the classical Hamiltonian pathfinding problem in our formulation, there were no distances in the original graph $\mathcal{G}$, but the SPP formulation requires distances to the edges, so we can assign a constant value 1 to all existing $(\hat{v}_i, \hat{v}_j)$ edges of the transformed graph $\hat{\mathcal{G}}$. However, in real-world problems, there are given distances: $\hat{d}(\hat{v}_i, \hat{v}_j) = d(v_x, v_y)$ and we are interested in solving the shortest Hamiltonian pathfinding problem (SHPP).
- Finally, an optimal solution to the shortest path finding problem of $\hat{\mathcal{G}}$ will provide the shortest Hamiltonian path with an objective function of $\min |\mathcal{H}| = \min \sum_{i=1}^{k} d_i$.

We can simplify the graph $\hat{\mathcal{G}}$ by merging vertex pairs $\hat{v}_a$ and $\hat{v}_b$ into a single transformed vertex if both contain the same original vertices and their last elements are identical. (For example, (v_1, v_2, v_3) can be merged into (v_2, v_1, v_3), but cannot be merged into (v_3, v_2, v_1)). The merging concept comes from the observation that it does not matter in a route what the vertex visiting order was, only the fact whether a vertex was visited or not and the last visited vertex, which determines where the traveler can move on. Denote by $\hat{v}_{\max} \subset \hat{V}$ those transformed vertices that contain all $v \in \mathcal{V}$. Then it can be shown that finding a Hamiltonian route in graph $\mathcal{G}$ indicates a path from $\hat{v}_0$ to $\hat{v}_{\max}$ in $\hat{\mathcal{G}}$, and vice versa.

Similarly to CSPP, a further constraint can be introduced to limit the distance that the traveler can cover in one go. Formally, the Hamiltonian path $\mathcal{H}$ should be split into subroutes or ranges that do not exceed their distance limit L: $\mathcal{H} = \cup_{i=1}^{l} \mathcal{H}_i$, where $\sum_{i \in \mathcal{H}_j} d_i \leq L$ for all $j \in \{1, 2, \ldots, l\}$. It is easy to see that $|\mathcal{H}| = \sum_{j=1}^{l} |\mathcal{H}_j|$.

The objective of the constrained shortest Hamiltonian pathfinding problem (CSHPP) will contain a second term to minimize the number of required ranges:

$$\min z = \big(c_d \sum_{j=1}^{l} |\mathcal{H}_j| + c_r l\big),$$

where c_d represents the proportional cost of the distance of the completed path and c_r covers the range cost.

A relevant practical problem type is the daily route planning of an electric delivery van where all the target addresses should be visited by taking into account the limits of the battery capacity and the charging options or the battery exchange opportunities.

Another type of problem that can be transformed into a CSPP is the disassembly line balancing problem. In its simplest form, we consider a disassembly line for a single product with a finite supply. Each product has n elementary components to remove, represented by a vertex set $\mathcal{V}$. The task of eliminating the component $v_i \in \mathcal{V}$ is specified by its processing time $t_i \in \mathcal{T}$, while a boolean flag of $h_i \in \mathcal{H}$ indicates its hazardousness, and finally $d_i \in \mathcal{D}$ declares the demand value of it. The general problem is to determine the disassembly order of the components and assign every task to the workstations of the disassembly line to optimize the objective function. The pre-defined cycle time is denoted by $t_c \in \mathbf{R}$, and each workstation should complete its assigned removal tasks within the cycle time. A directed precedence graph $\mathcal{P} = (\mathcal{V}, \mathcal{E}, (\mathcal{T}, \mathcal{H}, \mathcal{D}))$ describes the logical dependencies of component removal tasks. As we mentioned above, the vertices represent the components to be removed. The removal process of a component v_i can be started only if all components from which a directed edge goes to the vertex v_i are already removed. Typically, the precedence graph is used in its transitive reduced form. Consider a feasible solution that declares the order of component removals: the component represented by the vertex v_i should be removed as the r_ith element, and w_i^j determines which component should be removed on the workstation i as the jth task. Then the objective function can be formulated as follows:

$$\min \Big(c_i \sum_{i=1}^{l} (t_c - \sum_{j=1}^{l_j} t_{w_i^j})^2 + c_h \sum_{i=1}^{n} h_i r_i + c_d \sum_{i=1}^{n} d_i r_i\Big)$$

The first term determines the quadratic idle times of the used workstations, which supports not only minimizing idle times but also decreasing imbalance. The second and third terms depend on the component property and disassembly sequence and aim to remove the hazardous components and the components with higher values earlier. The three terms are weighted by c_i, c_h, and c_d, which have multiple goals behind them: compensating for the scaling discrepancies of the objectives and determining the relative importance of the objective components based on external preferences.

It is easy to see that the disassembly line balancing problem is a special modified case of a constrained Hamiltonian pathfinding problem since all the components should be removed. However, on the one hand, the disassembly of the components depends not only on the last removed components but also on the previously removed components, and on the other hand, the objective function is more complex, with a quadratic term in it. Similarly to the Hamiltonian pathfinding problem, we can construct a reversible transformation $\tilde{t}$ to formulate a problem as a CSPP to produce graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, (\tilde{\mathcal{T}}, \tilde{\mathcal{H}}, \tilde{\mathcal{D}}))$, for which the objective function becomes to the following form:

$$\min z = \big(c_i \sum_{j=1}^{l} (t_c - |\tilde{\mathcal{H}}_j|)^2 + c_h \sum_{j=1}^{n} \tilde{h}_j j + c_d \sum_{j=1}^{n} \tilde{d}_j j\big) \tag{2}$$

In the real world, the previously described problems are rather stochastic than deterministic: in the routing problems, the distances are measured on a time scale that depends on the traffic, the weather, and further external conditions, while in the disassembly problems, the component removal times depend on the product conditions.

In the formulation of stochastic shortest pathfinding problems, the directed graph has the same structure as in the deterministic case, so the vertex set and the edge set are identical, but the related distances are probability variables: $\mathbf{D}_i \sim \text{LogNormal}(\mu_{d_i}, \sigma^2_{d_i})$ for all $i = 1, \ldots, m$.

Classical solution methods are not directly applicable to the types of problems formulated above, or their resource requirements increase radically for larger problems. This led us to turn to the reinforcement learning method which demonstrated its ability to approach the optimal solution efficiently.

3. Q-Table Compression Method

In this section, we present a discretization method integrated into the Q-table-based policy iteration algorithm to solve mixed continuous-discrete problems with a reinforcement learning approach. First, we summarize the key properties of reinforcement learning by highlighting the action-value-based policy iteration methods, especially the every-visit Monte Carlo method and the Q-learning concept. Then we declare the basic requirements for a state compression solution. Finally, we describe the Q-table compression process in detail by providing the pseudo-algorithm to support its implementation.

3.1. Reinforcement Learning (RL)

Reinforcement learning (RL) solves problems due to sequential learning. An agent takes observations (O_t) of the environment and, based on that, executes an action (A_t). As a result of the action in the environment, the agent will receive a reward (R_t), and it can take a new observation (O_{t+1}) from the environment, and the cycle is repeated. The problem is to let the agent learn to maximize the total expected reward.

Preliminary, we should highlight that graph traversal problems can be approached as a sequential decision problem: the agent observes the traveled path, which is a sequence of the visited vertices, and needs to decide on the next action that should define the next vertex to visit. Therefore, RL techniques can provide an obvious solution for graph traversal problems by determining a sequence of vertices to find an optimal (shortest) path.

Reinforcement learning is based on the reward hypothesis, which states that maximization of expected cumulative rewards can describe all goals. Formally, the history is the sequence of observations, actions, and rewards: $H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$. The state is the information used to determine what happens next. Formally, the state is a function of the history: $S_t = f(H_t)$. A state is Markov if and only if $\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$. Markov property is fundamental to the theoretical basis of RL methods. G_t denotes the total discounted reward of the time step t: $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. According to these, for a graph traversal problem, the action space is practically the vertex set: $A_t \in \mathcal{V}$, the observation space is the subroute performed: $O_t = ((v_0, v_1), (v_1, v_2), \ldots (v_{t-1}, v_t) \subset \mathcal{E}$, while the reward is the increment of the partial objectives: $R_t \in \mathcal{R} = z_t - z_{t-1}$

The state value function $v(s)$ gives the expected total discounted return if starting from state s: $v(s) = \mathbb{E}[G_t \mid S_t = s]$. The Bellman equation practically states that the state value function can be decomposed into two parts: immediate reward (R_{t+1}) and the discounted value of successor states $\gamma v(S_{t+1})$.

The policy covers the agent's behavior in all possible cases, so it is essentially a map from states to actions. There are two main categories in it: deterministic policy ($a = \pi(s)$) and stochastic policy ($\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$).

We will focus on using an action-value function to determine the current optimal action. However, it can be a very slow process for large state- and/or action spaces to keep the value function updated (and hence optimal). Denote by $N(s, a)$ the counter of visiting the state s with an action selection of a. Then consider the function $Q(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$, which accumulates the expected total discounted reward starting from state s and choosing action a. According to the every-visit Monte Carlo policy evaluation process, if the agent performs a new episode and receives rewards accordingly, then the counter must be incremented: $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$, and the Q-value function must be updated for all visited (S_t, A_t) pairs: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t,A_t)}\big(G_t - Q(S_t, A_t)\big)$. It is proven that if the agent follows the update of the Q value function combined with a simple ϵ-Greedy action selection method, then the Q-value function approaches the optimal policy as the number of observations tends to infinity.

There are several situations where the learning process is not based only on the agent's own experience. Formally, this means that action-value function $q_\pi(s; a)$ is determined by observing the results of an external behavior policy $\mu(a|s)$.

A possible way to handle the difference between the target and behavior policies is to modify the update logic of the value function as Q-learning does (Section 6.5 in [28]). Assume that in state S_t the very next action is derived by using the behavior policy: $A_{t+1} \sim \mu(\cdot|S_t)$. By taking action A_{t+1} immediate reward R_{t+1} and the next state S_{t+1} will be determined. But for the update of the value-function, let us consider an alternative successor action based on target policy: $A' \sim \pi(\cdot|S_t)$. Then the Q-learning value-function update will look like: $Q(S_t; A_t) \leftarrow Q(S_t; A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}; A') - Q(S_t; A_t)\big)$.

In a special case, if the target policy π is chosen as a pure greedy policy and the behavior policy μ follows ϵ-greedy policy, then the so-called SARSAMAX update can be defined as follows: $Q(S; A) \leftarrow Q(S; A) + \alpha\big(R + \gamma \max_{a'} Q(S'; a') - Q(S; A)\big)$. Last, but not least, it was proven that Q-learning control converges to the optimal action-value function: $Q(s; a) \rightarrow q_*(s; a)$.

In our solution, we used the every-visit Monte Carlo method to determine the optimal policy. However, this can be easily replaced by Q compression, but in that case, it is necessary to choose a suitable α-strategy.

3.2. *Q-Table Compression Process*

In this section, we collect the basic requirements of a discretization function, then present the Q-table compression process in detail, and highlight some hints and guidelines for implementation.

In general, Q-table methods are used for discrete problems. However, there are cases where the discrete problems partially become continuous by considering some stochastic effect or measurement uncertainty. One option to decrease the complexity of such a problem is to divide it into sub-problems. It can be done by introducing sub-goals and solving these problems separately [29]. Another option is to merge those observed states into a common one that differs from each other only insignificantly. This can be done by introducing a grid representation over the continuous observation space [30]. In this section, we will present an integrated method to dynamically redefine the state space and transform the Q-table to align it to the changes. We assume a stochastic Markov Decision Process with a mixed-integer state space and discrete action space.

o^i_j denotes the jth observation in episode i, and it is an element of the observation space O. Our goal is to define a discrete representation S that will be used as a state space in the reinforcement learning process. Formally, we are looking for an iteratively refined mapping function $\mathcal{F}^i : O \rightarrow S$.

We can declare the following requirements for the functions $\mathcal{F}^i$.

- Let's assume that the optimal action is determined for each and every o^i_j and it is denoted by a^i_j. If $a^i_j \neq a^k_j$ then $\mathcal{F}(a^i_j) \neq \mathcal{F}(a^k_j)$. This practically means that the mapping function merges observation states only if their optimal actions do not differ.
- The mapping function $\mathcal{F}$ should be consistent in that sense that if action a^i_j moves the agent from the observation state o^i_j to o^i_{j+1} then action a^i_j should move the agent from $\mathcal{F}(o^i_j)$ to $\mathcal{F}(o^i_{j+1})$.
- $|S| \ll |O|$, or in other words: the size of the state representation S should be significantly smaller than the size of observation space O. The smaller S is the better representation.

The first requirement prevents merging states that have different optimal actions. The second defines the transitivity of the mapping function. The last requirement states that the potential of using the described representation function depends on the efficiency of dimension reduction: if no significant reduction can be reached, then the method cannot raise the efficiency of the learning process.

According to the above requirements, we design a modified Q-table-based policy iteration method by integrating a discretization step into it. In the simplest RL framework, the agent observes its states directly and takes its actions accordingly. In our approach, the agent takes observations, determines the simplified state, and takes actions based on it. So observations determine the state, but observation space differs from state space: the first one is a mixed-integer space, while the last one is purely discrete. The key component in the process is the state space definition and its projection.

We suggest maintaining a compression-based Q-table learning process due to the following steps, which are summarized in Figure 1.

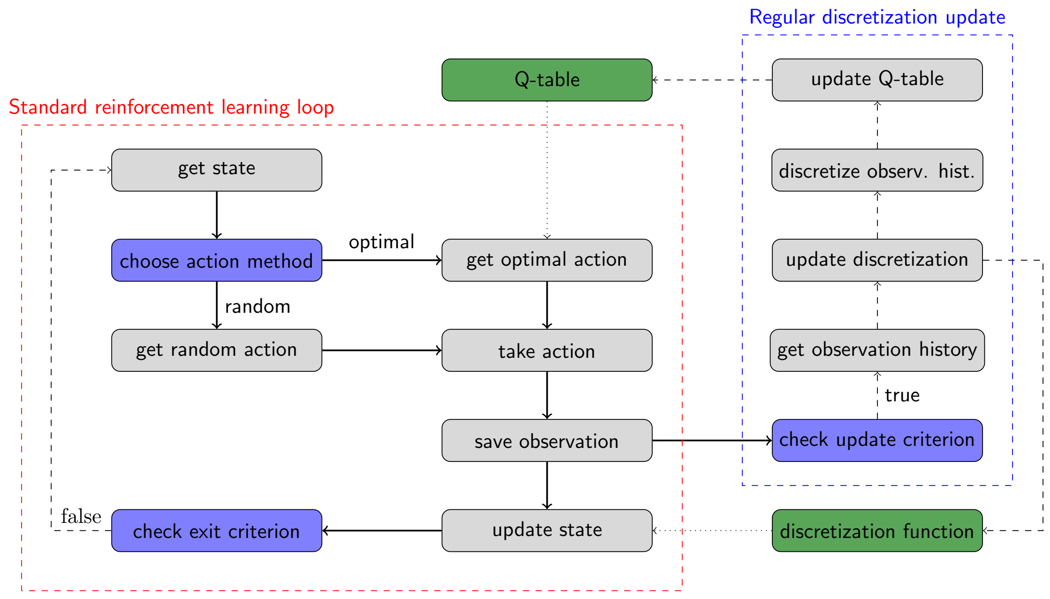

Figure 1. Q-compression process flow.

There is a standard reinforcement learning loop in the process. The agent queries its current discretized state (`get state`). The learning process is based on the ϵ-Greedy algorithm: the agent decides whether to take a random action or take the best-known action (`choose action method`). The agent randomly chooses an action with ϵ probability of the feasible actions (`get random action`), or chooses the optimal action from the feasible actions with $(1-\epsilon)$ probability (`get optimal action`). In this case, the agent determines the expected cumulative rewards to reach the target state for all feasible actions first and then chooses the action with the best total expected reward (or if there are multiple actions with the same total expected reward, then randomly choose one out of them). If the Q-table does not contain a relevant entry for the current state because the trajectory is undiscovered, then the action selection falls back to the random method. The value of the parameter ϵ decreases from 1 to 0 linearly as the number of episodes goes to its predefined limit. We would like to highlight that the restriction on potential actions improves the efficiency of the learning process compared to enabling any action and producing a bad reward for unfeasible actions.

Once the action is chosen, the agent performs the selected action, determines the new observation state, and registers the reward (`take action`). After that, the agent saves the quadlet (old state, action, new state, reward) into the observation history (`save observation`), and determines the new discretized state by applying the discretization function for the observation (`update state`). Finally, the agent checks whether the target state has been reached (`check exit criterion`). If not, then the cycle repeats.

Next to the standard reinforcement learning loop, the agent needs to update the discretization regularly, and on top of that, compress the state space size. The discretization update process is quite resource-intensive. Therefore, we suggest performing it after a batch of episodes. Once, when the update criterion is satisfied (`check update criterion`) the agent queries the observation history (`get observation history`). To let the agent act adaptively, the history can be accessible due to a moving observation window to get the most relevant part of the history and not process the old, invalid records from there. The major goal at this stage is to create a new/updated discretization ruleset that fits the requirements as described above in this subsection and projects the mixed discrete-continuous observation space to a discrete state space (`update discretization`). Denote by $s_t = (v_t, u_t) \in \mathcal{V} \times \mathbf{R}$ an observation, where v_t shows the currently visited vertex of the graph, and u_t describes the current range utilization. The agent takes the observations in a loop: It fixes the discrete part of the observation space (practically a vertex) and queries all matching records: $(v_i, u_i)|v_i = v_{fixed}$. Then it uses the DBSCAN algorithm to assign the continuous range utilization values of the observations into clusters. Denote R_i^v the ith range of the corresponding cluster and k_v the number of identified clusters of the vertex v. The ranges should be nonoverlapping: $R_i \bigcap R_j = \emptyset$ for all $i \neq j \in \{1, \dots, k_v\}$, and they need to be widened to completely cover the potential value range: $\bigcup_{i=1}^{k_v} R_i = \mathbf{R}$. Figure 2 demonstrate the result of applying DBSCAN algorithm to determine the clusters for two different verices.

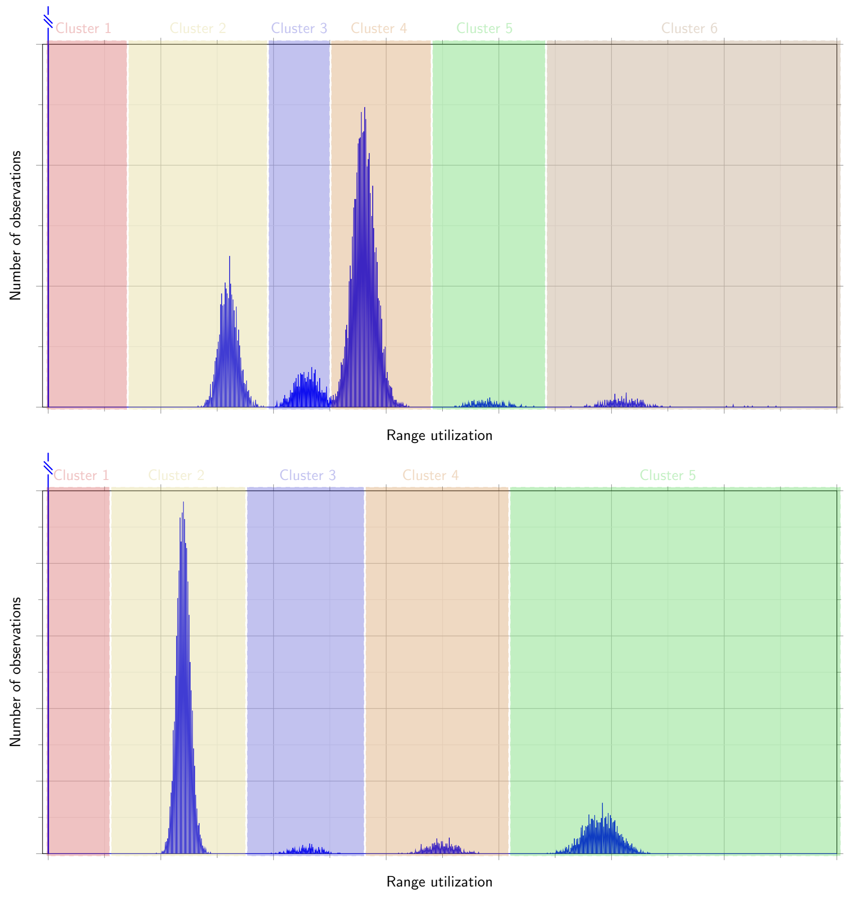

Figure 2. Applying DBSCAN for clustering range utilization component of a selected vertex.

The agent repeats the process for all elements in the loop. After updating the discretization function for the ith time by assigning the centroid value to each elements of the widened ranges, the agent applies it to the observation history to produce a discretized version of it: $\mathcal{F}^i : \mathcal{V} \times \mathbf{R} \rightarrow \mathcal{V} \times \{R_1, R_2, \dots R_{v_k}\}$ (`discretize observation history`). Finally, the agent truncates the Q-table and rebuilds it from the discretized observation history using the discretized states (`update Q-table`). The Algorithm 1 presents the Q-compression method in pseudo-code format .

Algorithm 1 Q-table compression reinforcement learning method

```
function Q-COMPRESSION($D$, $l_{sim}$, $l_{range}$)
inputs:
    $D$: $n \times n$ matrix                                   ▷ represents the graph distances
    $l_{sim}$: constant parameter                              ▷ define simulation length
    $l_{range}$: constant parameter                            ▷ define single range limit
                                                               ▷ initialize learning parameters
    $Q() \leftarrow [.]$                                       ▷ initialize Q-table
    $O() \leftarrow [.]$                                       ▷ initialize observation history
    for $i = 1, 2, \dots, l_{sim}$ do                          ▷ loop for iterating episodes
        $(v, u)_{curr} \leftarrow (v_{start}, 0)$              ▷ reset state (current vertex, range util.)
        while $v_{curr} \neq v_{target}$ do                    ▷ check episode exit criterion
            $\xi \leftarrow \backsim U(0, 1)$                  ▷ generate standard uniform random number
            $\mathbf{a} \leftarrow \{v | D(v_{curr}, v) > 0\}$ ▷ get feasible action options list
            if $\xi < \frac{i}{l_{sim}}$ then                  ▷ optimal action selection criterion
                $m \leftarrow \max\left(r = Q(s, a) | s = (v_{curr}, \mathcal{F}_{v_{curr}}(u))\}\right)$ ▷ get maximal expected cumulative reward from Q-table
                $\mathbf{v_{max}} \leftarrow \left\{a | Q(s, a) = m, s = (v_{curr}, \mathcal{F}_{v_{curr}}(u))\right\}$ ▷ get all maximal-reward actions from Q-table
                if $|\mathbf{v_{max}} > \mathbf{0}|$ then      ▷ If no applicable action found then fallback to random action
                    $\mathbf{a} \leftarrow \mathbf{a} \cap \mathbf{v_{max}}$ ▷ restrict optimal action list to optimals
                end if
            end if
            $a \leftarrow \backsim U(\mathbf{a})$              ▷ choose next action from options
            $v_{next} \leftarrow a, d \backsim D(s_{curr}, a), u_{next} \leftarrow u - d$ ▷ determine next obs.
            $\mathbf{r} \leftarrow$ REWARD$(s_{curr}, a, d)$   ▷ get reward
            $O \xleftarrow{+} ((s_{curr}, a, s_{next}, r)$     ▷ save observation into history
            $v_{curr} \leftarrow v_{next}, u_{curr} \leftarrow u_{next}$ ▷ update state
        end while
        if  mod $(i, l_{range}) == 0$ then                     ▷ Q-table update due criterion
            UPDATE-Q($O$)                                      ▷ call discret. and Q-table update sub-process
        end if
    end for
end function
function REWARD($s_{curr}$, $a$, $d$, $l_{range}$)
inputs:
    $s_{curr}$: pair of current vertex and range utilization
    $a$: single vertex                                         ▷ action $a$ determines the next vertex to visit
    $d$: dynamic value                                         ▷ measures distance realized on performed action $a$
    $l_{range}$: constant parameter                            ▷ define utilization limit
    $r_d \leftarrow c_d \cdot d$                               ▷ get reward term of distance proportional cost
    $r_r \leftarrow c_r(d == u_{curr})$                        ▷ get reward term of range proportional cost
    $r_o \leftarrow c_o(l_{range} < u_{curr})$                 ▷ get reward term of range overutilization cost
    return $r_d + r_r + r_o$
end function
```

Algorithm 1 *Cont.*

```
43: function UPDATE-Q($O$)
44: inputs:
45:     $O$: list of quad-tuples                    ▷ stores the observation history up to episode $i$
46:     for $v \in \mathcal{V}$ do                  ▷ loop for iterating vertices
47:         $O_v \leftarrow \{O(s_{curr}, a, s_{next}, r) | s_{curr} = (v, .)\}$    ▷ filter for relevant obs.
48:         $C_v \leftarrow \text{DBSCAN}(O_v(r))$   ▷ determine clusters by DBSCAN
49:         $R_v \leftarrow C_v$                     ▷ widen the clusters to get complete disjoint covering ranges
50:         for $i = 1, 2, \ldots, k_v$ do           ▷ loop on identified clusters of vertex $v$
51:             $\mathcal{F}_v(u \in R_v^i) \leftarrow \text{AVERAGE}(R_v^i)$   ▷ update discretization function by assigning cluster's centroid value
52:         end for
53:     end for
54:     $Q() \leftarrow [.]$                          ▷ reset Q-table
55:     for $j = 1, 2, \ldots, |O|$ do               ▷ loop for iterating obs. history
56:         $((v, u)_{curr}, a, r, (v, u)_{next}) \leftarrow O_j(s_{curr}, a, r, s_{next})$   ▷ read $j$th obs.
57:         $\tilde{s}_{curr} \leftarrow (v_{curr}, \mathcal{F}(u_{curr}))$   ▷ determine discretized current state
58:         $\tilde{s}_{next} \leftarrow (v_{next}, \mathcal{F}(u_{next}))$   ▷ determine discretized next state
59:         $Q(\tilde{s}_{curr}, a) \overset{+}{\leftarrow} \frac{1}{|O(\tilde{s}_{curr}, a, ., .)|} (r + \gamma \max_{a_{next}} Q(\tilde{s}_{next}, a_{next}) - Q(\tilde{s}_{curr}, a))$   ▷ calculate expected reward for the intervals discretized by DBSCAN
60:     end for
61: end function
```

4. Results and Discussion

In this section, we present three application examples in which we demonstrate the usability of the Q-compression method. The first use case is a constrained shortest pathfinding problem (CSPP), the second one is a constrained shortest Hamiltonian pathfinding problem (CSHPP), and the last one is a disassembly line balancing problem (DLBP). We present the performance of Q-compression method in these examples and compare it to a simple grid-based discretization method and the optimal solution of the deterministic version of the problems.

If we assume that the constrained graph traversal process is a sequence of decisions to determine which vertex should be visited next, then two pieces of information influence the decision: the currently visited vertex and the range utilization. In Table 1 we summarize the key components of the constrained graph traversal problem class.

As we described in Section 2 the Hamiltonian pathfinding problems (including the disassembly problems) can be transformed into the shortest pathfinding problem. The Q-compression method is executed on the original graph in the CSPP case and on a transformed graph in the other two cases. In CSHPP the transformed graph vertices correspond to a feasible subpath that ends in a specific vertex in the original graph. In DLBP the last vertex of the sub-path has no influence, therefore, the transformed graph's vertices correspond to the feasible subpaths only independently from the last vertex.

The edges represent the connectivity of two vertices: whether or not a feasible transition from one vertex to the other exists. Edges integrate information about vertex connectivity in the CSPP and CSHPP cases and the precedence graph in the DLBP case.

The constraints describe the restrictions on range utilization in the CSPP and CSHPP cases and the utilization of the workstation in the DLBP case.

In the CSPP and CSHPP cases, the objective stands for a weighted sum of the total distances of the traveled path and the number of performed ranges. In the DLBP case, the objective is more complex: it is a weighted sum of the quadratic idle time of the workstations and the products of component disassembly order and their hazardousness and demand values.

Aligning to the traditional setup of RL solutions, we use the opposite of objectives for reward. Therefore, optimization problems turn to maximization tasks. Our results are presented by retransforming rewards to original objectives.

Table 1. Constrained graph traversal problem components by problem types.

	Vertex Setup (State Space)
CSPP	current vertex
CSHPP	visited vertices + current vertex
DLBP	removed components
	Edge context (action space)
CSPP	integrates the vertex connectivity
CSHPP	integrates the vertex connectivity
DLBP	integrates the precedence graph
	Constraints (restrictions for action selection)
CSPP	range utilization $\leq$ battery capacity
CSHPP	range utilization $\leq$ battery capacity
DLBP	workstation utilization $\leq$ cycle time
	Objective
CSPP	$\min\left(c_d \sum_{j=1}^{l} \lvert\mathcal{P}_j\rvert + c_r l\right)$
CSHPP	$\min\left(c_d \sum_{j=1}^{l} \lvert\mathcal{H}_j\rvert + c_r l\right)$
DLBP	$\min\left(c_i \sum_{i=1}^{l} (t_c - \sum_{j=1}^{l_j} t_{w_i^j})^2 + c_h \sum_{i=1}^{n} h_i r_i + c_d \sum_{i=1}^{n} d_i r_i\right)$

4.1. Constrained Shortest Path Finding Use Case

Our first use case is a simple shortest pathfinding problem. Figure 3 presents a directed graph.

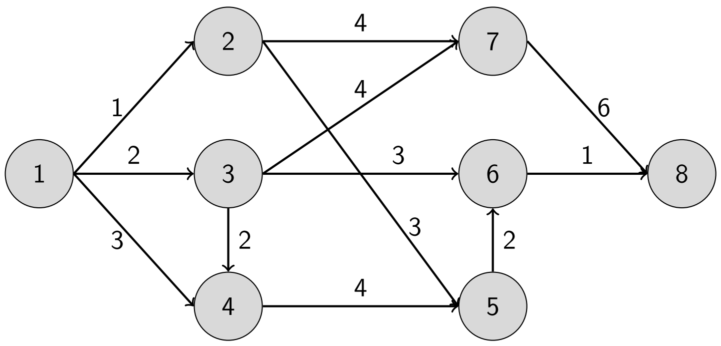

Figure 3. Distance graph of shortest pathfinding use case.

The edges are labeled with their average lengths. However, these distances are stochastic, which was simulated by multiplying random values from a log-normal distribution with parameter values of $\mu = 0$ and $\sigma = 0.01$. The goal is to find the shortest path from vertex 1 to vertex 8, taking into account an additional constraint: the traveler can only go through $L = 4.5$ units in one go. We can easily see that the expected overall shortest path has a length of 7 units. The path $1 \rightarrow 3 \rightarrow 6 \rightarrow 8$ should be divided into three sub-paths to remain within the range limit, while the path $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 8$ needs to split only into two.

The objective function is a slightly modified version of Equation (1) by standing for three terms:

$$\min z = \left(c_d \sum_{j=1}^{l} |\mathcal{P}_j| + c_o \sum_{j=1}^{l} (|\mathcal{P}_j > L) + c_r l\right) \tag{3}$$

The first term measures the total length of performed distances. The second term counts the ranges that exceed the limit value of L. Converting the range-limit constraint into an objective term transforms it into a soft condition. The third term describes the number of ranges into which the entire route was divided. The final objective function is a weighted sum of these three terms. In our use case, we set the c_d distance proportional cost to 1, the c_o overutilization cost to 100, and the c_r range cost to 10.

We performed 20 simulations with 500 episodes in each for both the Q-compression method and the grid-based method. The granularity of the grid was set to 0.01. Figure 4 shows the distance distribution histogram by edges with a granularity of 0.01 unit.

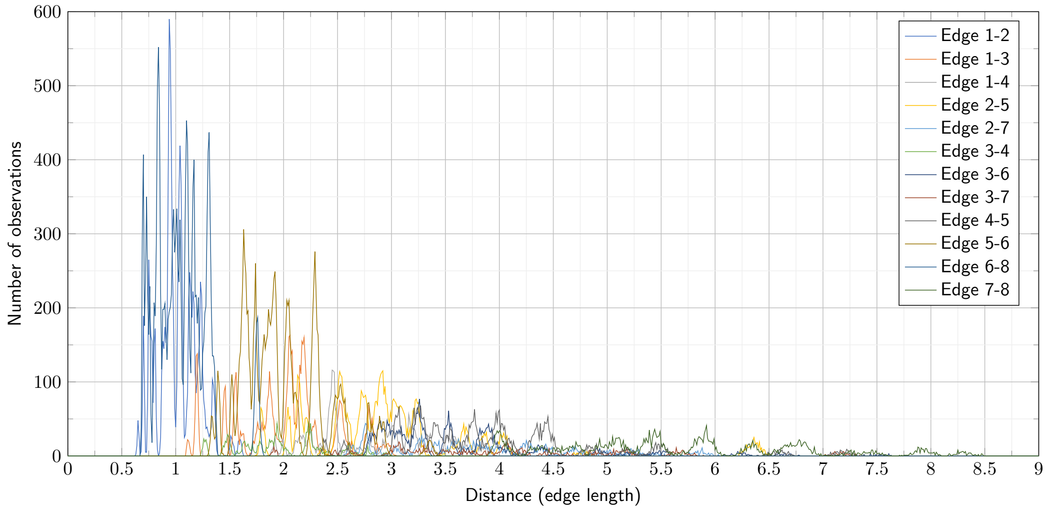

Figure 4. Distances by edges of constrained shortest pathfinding use case.

We applied the ϵ-Greedy decision mechanism with a linearly decreasing ϵ-strategy. The frequency of updating the discretization and the Q-table was set to 50 episodes. Figure 5 presents the results of our simulation.

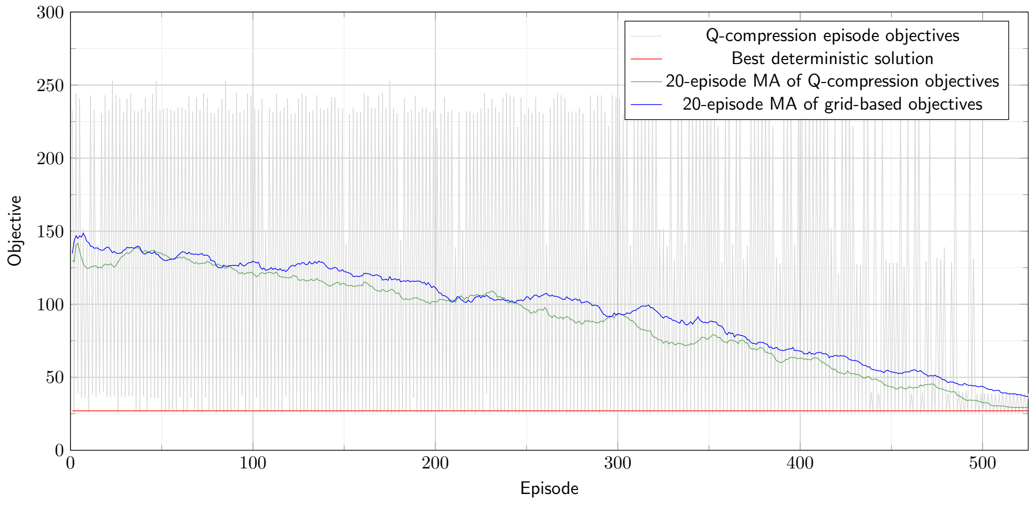

Figure 5. Q-compression method performance on constrained shortest pathfinding use case.

The optimal solution of the deterministic problem has an objective value of 27. We can observe that in some cases the empirical episode reward is lower than this, which comes from the stochastic property of the problem. Both the grid-based discretization method and the Q-compression method improve their performance as ϵ approaches 0, but the Q-compression finds a better objective value on average than the grid-based method (29.26 vs. 36.95). Table 2 summarizes the increase in Q-table size by episodes for the grid-based method and the Q-compression method. It shows that the Q-table continuously grows using the grid-based discretization method, even if with decreasing momentum. In contrast, the Q-compression method keeps the Q-table compact, which definitely supports the better learning capability that we observed in the achieved objective values.

Table 2. Q-table size comparison for CSPP by episodes.

Discretization	**Episode**									
Method	**50**	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
Grid-based	135	203	254	292	320	340	356	366	372	375
Q-compression	39	42	43	43	43	43	43	43	43	43

4.2. Constrained Hamiltonian Path Finding Use Case

Our second use case is a Hamiltonian pathfinding problem [31]. Figure 6 presents a directed graph.

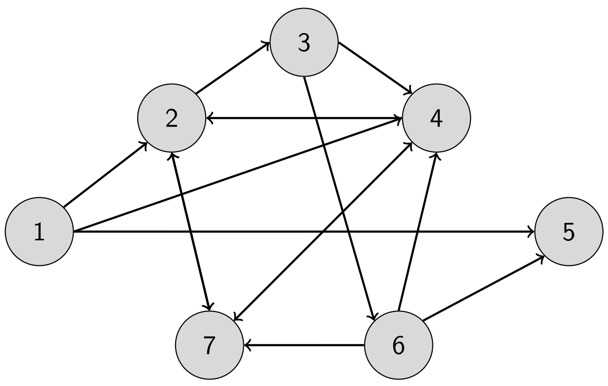

Figure 6. Directed graph of Hamiltonian pathfinding use case.

Since there are no distances declared to the edges, we set the average length of all the edges to 1. To ensure the feasibility of the problem, non-existing edges are replaced with new edges with an average length of 100. Similarly to the CHPP use case, we apply multiplicative stochastic noise from a log-normal distribution with parameter values of $\mu = 0$ and $\sigma = 0.05$. The goal is to find the continuous path from vertex 1 to vertex 5 by visiting all other vertices exactly once. The range limit is set to $L = 10$. We should note that an entire feasible Hamiltonian path does not exceed the range limit. Furthermore, it is easy to validate that $1 \rightarrow 4 \rightarrow 7 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5$ is a suitable solution (and the only one).

The objective function is identical to the previous one formulated in Equation (3). The weighting parameters are set as follows: $c_d = 1$, $c_u = 100$, $c_r = 10$. Further settings are identical to the descriptions of the CSPP use case.

We performed 20 simulations with 2000 episodes in each for both the Q-compression method and the grid-based method. The granularity of the grid was set to 0.1. The frequency of updating the discretization and the Q-table was set to 100 episodes. Figure 7 presents the results of our simulation.

The optimal solution of the deterministic problem has an objective value of 16. The Q-compression method approaches the optimal solution better by reaching an average objective value of 17.01 than the grid-based method, which produces an average objective value of 224.38. However, both show constant convergence to it. Table 3 shows the size of the Q-table by episodes. We can observe that the grid-based method grows the Q-table to an average size of 3647 records, while the Q-compression method leads to less than half the size by having 1735 records on average.

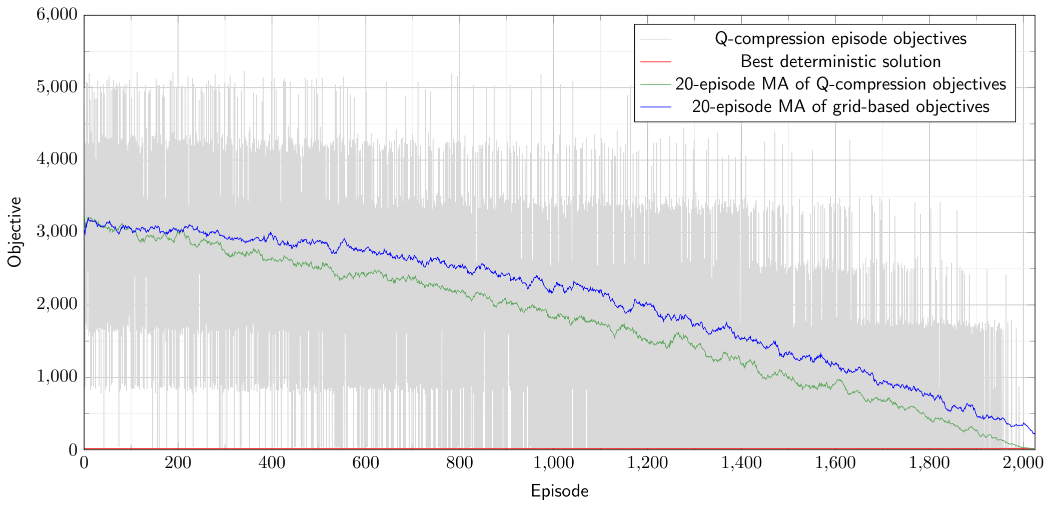

Figure 7. Q-compression method performance on constrained Hamiltonian pathfinding use case.

Table 3. Q-table size comparison for CHPP by episodes.

Discretization Method	**Episode 100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Grid-based	649	1127	1526	1863	2167	2425	2650	2844	3008	3145
Q-compression	411	615	788	944	1093	1223	1339	1428	1504	1563
Discretization method	Episode 1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Grid-based	3266	3362	3442	3504	3554	3589	3613	3630	3640	3647
Q-compression	1615	1651	1678	1698	1713	1723	1730	1733	1735	1735

4.3. Disassembly Line Balancing Use Case

Our third use case is a computer disassembling problem from the literature [32,33]. Similarly to the previous use cases, we apply multiplicative stochastic noise from a log-normal distribution with parameter values of $\mu = 0$ and $\sigma = 0.05$. There are identified 8 salvageable components of a PC. The parts themselves, their removal times, demand values, and hazardousness indicators are collected in Table 4.

Table 4. Personal computer disassembly tasks and parameters.

Task No.	**Disassembly Task**	**Removal Time**	**Demand**	**Hazardousness**
1	PC top cover	14	360	No
2	Floppy drive	10	500	No
3	Hard drive	12	620	No
4	Backplane	18	480	No
5	PCI cards	23	540	No
6	RAM modules (2)	16	750	No
7	Power supply	20	295	Yes
8	Motherboard	36	720	No

A precedence graph describes the logical dependencies of the order of the disassembly task in Figure 8.

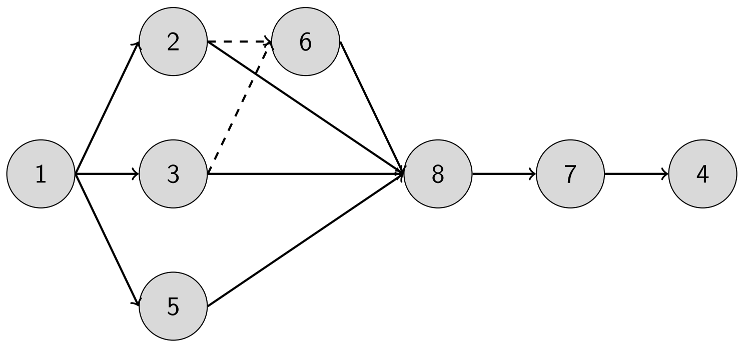

Figure 8. Precedence graph of personal computer disassembly problem.

We can distinguish two types of edges: solid edges represent predecessor AND relations, while dashed edges represent predecessor OR relations. The predecessor AND relation ($P_{AND}(i)$) declares a set of predecessor tasks that all must be completed before starting the task i. The predecessor OR relation ($P_{OR}(i)$) declares a set of predecessor tasks of which at least one needs to be completed before starting the task i.

The objective function comes from Equation (2) with a minor modification for sanctioning workstation over-utilization:

$$\min z = (c_i \sum_{j=1}^{l} (t_c - |\tilde{\mathcal{H}}_j|)^2 + c_o \cdot l \min(\max(|\tilde{\mathcal{H}}_j| - t_c), 0) + c_h \sum_{j=1}^{n} \tilde{h}_j j + c_d \sum_{j=1}^{n} \tilde{d}_j j$$

The first term is the quadratic idle time. It minimizes both the number of workstations used for disassembling the products and their imbalance. The second term is the soft criterion to keep the utilization of the workstation under the limit. If any of the workstations breaks the limit, then a significant penalty will be imposed. The third and fourth terms force the removal of hazardous components and components with higher demand value earlier. Following the referenced literature, the weights of c_i, c_h, and c_d are set at 1, while c_o has a value of 100. The cycle time (which corresponds to the range limit L of previous use cases) is declared externally: $t_c = 40$.

We performed 20 simulations with 1000 episodes each for both the Q-compression method and the grid-based method. The granularity of the grid was set to 0.1. The frequency of updating the discretization and the Q-table was set to 100 episodes. Figure 9 shows the removal time distribution histogram with 0.1 unit granularity by components of the personal computer.

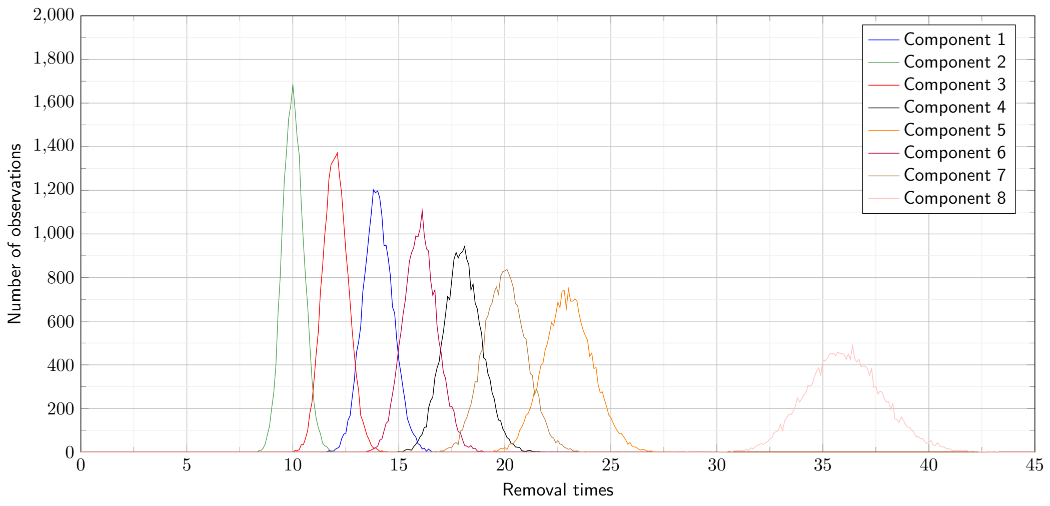

Figure 9. Removal time distributions by personal computer components.

The optimal solution to the deterministic problem has an objective value of 19,065. In this use case, the reward curves of the two discretization methods diverge. The Q compression method significantly better approaches the optimal solution by reaching an average objective value of 19,122.29. In contrast, the grid-based method produces an average objective value of 22,747.21 (Figure 10).

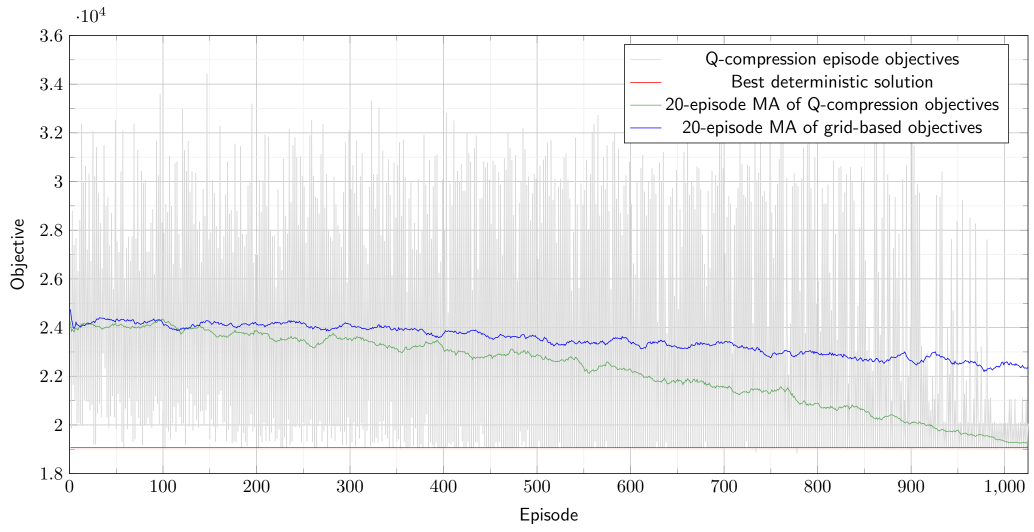

Figure 10. Q-compression method performance on personal computer disassembly use case.

As Table 5 shows, the grid-based method grows the Q-table to an average size of 2672 records, while the Q-compression method keeps the Q-table very compact by having 146 records on average, which definitely supports the better learning capability that we observed in the achieved objective values.

Table 5. Q-table size comparison for DLBP by episodes.

Discretization	**Episode**									
Method	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Grid-based	702	1158	1522	1834	2087	2320	2485	2609	2716	2807
Q-compression	144	151	152	150	147	147	146	146	146	146

5. Conclusions

In our article, we gave an overview of some graph traversal problems and showed that they have a common root, namely the constrained shortest path finding problem. Then we presented a general multistep Q-table compression method to provide a human-interpretable solution for mixed-integer optimization problems. The main result of our method is that it provides an algorithm to find an optimal abstraction level by deriving a reduced state space with significantly smaller number of elements. We demonstrated the ability of our method on three selected use cases of the constrained stochastic graph traversal problem class, namely, a constrained stochastic shortest pathfinding problem, a constrained stochastic Hamiltonian pathfinding problem, and a stochastic disassembly line balancing problem. In all three use cases, we compared the performance of our Q-compression method for discretizing the mixed continuous-discrete state-space to the classical grid-based partitioning approach. We found that our DBSCAN-based Q-table compression method achieves a better objective function value while producing a more compact Q-table than the reference technique. As the optimization problem is more complex, the difference in performance is more significant.

In future research, on the one hand, we desire to test our method on larger graph traversal problems as well and to analyze the options for transferring the knowledge from a Q-table with a structure of a previous iteration to a new one decreasing the training period.

On the other hand, as the Q-compression method is a general technique for solving mixed-integer sequential optimization tasks, we are working on identifying further problem types for which it can be applicable, such as constrained stochastic backpack problems and constrained stochastic production scheduling tasks.

Author Contributions: Conceptualization, A.K. and J.A.; methodology, T.K., A.K., Z.S. and J.A.; software, T.K.; validation, A.K., Z.S. and J.A.; formal analysis, T.K. and A.K.; investigation, T.K.; resources, J.A.; data curation, T.K.; writing—original draft preparation, T.K.; writing—review and editing, A.K. and Z.S.; visualization, T.K.; supervision, J.A.; project administration, J.A.; funding acquisition, J.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the TKP2021-NVA-10 project with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the 2021 Thematic Excellence Programme funding scheme.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data and source code will be shared on request.

Conflicts of Interest: The authors declare no conflicts of interest regarding the publication of this article. However, it should be noted that János Abonyi serves as a guest editor for the journal, therefore he has not been involved in the editorial handling or decision-making process for this manuscript to ensure impartiality and transparency in the review process.

Abbreviations

The following abbreviations are used in this manuscript:

RL	Reinforcement Learning
SPP	Shortest Pathfinding Problem
CSPP	Constrained Shortest Pathfinding Problem
HPP	Hamiltonian Pathfinding Problem
CSHPP	Constrained Shortest Hamiltonian Pathfinding Problem
DLBP	Disassembly Line Balancing Problem

References

1. Liao, X.; Wang, J.; Ma, L. An algorithmic approach for finding the fuzzy constrained shortest paths in a fuzzy graph. *Complex Intell. Syst.* **2021**, *7*, 17–27. [CrossRef]
2. Qin, H.; Su, X.; Ren, T.; Luo, Z. A review on the electric vehicle routing problems: Variants and algorithms. *Front. Eng. Manag.* **2021**, *8*, 370–389. [CrossRef]
3. Vital, F.; Ioannou, P. Scheduling and shortest path for trucks with working hours and parking availability constraints. *Transp. Res. Part B Methodol.* **2021**, *148*, 1–37. [CrossRef]
4. Baum, M.; Dibbelt, J.; Gemsa, A.; Wagner, D. Towards route planning algorithms for electric vehicles with realistic constraints. *Comput.-Sci.-Res. Dev.* **2016**, *31*, 105–109. [CrossRef]
5. Baum, M.; Dibbelt, J.; Gemsa, A.; Wagner, D.; Zündorf, T. Shortest feasible paths with charging stops for battery electric vehicles. *Transp. Sci.* **2019**, *53*, 1627–1655. [CrossRef]
6. Adler, J.D.; Mirchandani, P.B.; Xue, G.; Xia, M. The electric vehicle shortest-walk problem with battery exchanges. *Netw. Spat. Econ.* **2016**, *16*, 155–173. [CrossRef]
7. Çil, Z.A.; Öztop, H.; Kenger, Z.D.; Kizilay, D. Integrating distributed disassembly line balancing and vehicle routing problem in supply chain: Integer programming, constraint programming, and heuristic algorithms. *Int. J. Prod. Econ.* **2023**, *265*, 109014. [CrossRef]
8. Ulmer, M.W.; Goodson, J.C.; Mattfeld, D.C.; Thomas, B.W. On modeling stochastic dynamic vehicle routing problems. *EURO J. Transp. Logist.* **2020**, *9*, 100008. [CrossRef]
9. Slama, I.; Ben-Ammar, O.; Masmoudi, F.; Dolgui, A. Disassembly scheduling problem: Literature review and future research directions. *IFAC-PapersOnLine* **2019**, *52*, 601–606. [CrossRef]
10. Ferone, D.; Festa, P.; Guerriero, F.; Laganà, D. The constrained shortest path tour problem. *Comput. Oper. Res.* **2016**, *74*, 64–77. [CrossRef]
11. Kang, J.G.; Lee, D.H.; Xirouchakis, P.; Persson, J.G. Parallel disassembly sequencing with sequence-dependent operation times. *CIRP Ann.* **2001**, *50*, 343–346. [CrossRef]

12. AbuSalim, S.W.; Ibrahim, R.; Saringat, M.Z.; Jamel, S.; Wahab, J.A. Comparative analysis between Dijkstra and Bellman-Ford algorithms in shortest path optimization. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2020; Volume 917, p. 012077.
13. Toroslu, I.H. Improving the floyd-warshall all pairs shortest paths algorithm. *arXiv* **2021**, arXiv:2109.01872.
14. Ferone, D.; Festa, P.; Guerriero, F. An efficient exact approach for the constrained shortest path tour problem. *Optim. Methods Softw.* **2020**, *35*, 1–20. [CrossRef]
15. Dondo, R. A new formulation to the shortest path problem with time windows and capacity constraints. *Lat. Am. Appl. Res.* **2012**, *42*, 257–265.
16. Zhang, J.; Liu, C.; Li, X.; Zhen, H.L.; Yuan, M.; Li, Y.; Yan, J. A survey for solving mixed integer programming via machine learning. *Neurocomputing* **2023**, *519*, 205–217. [CrossRef]
17. Magzhan, K.; Jani, H.M. A review and evaluations of shortest path algorithms. *Int. J. Sci. Technol. Res* **2013**, *2*, 99–104.
18. Hildebrandt, F.D.; Thomas, B.W.; Ulmer, M.W. Opportunities for reinforcement learning in stochastic dynamic vehicle routing. *Comput. Oper. Res.* **2023**, *150*, 106071. [CrossRef]
19. Li, S.E. *Reinforcement Learning for Sequential Decision and Optimal Control*; Springer: Berlin/Heidelberg, Germany, 2023.
20. Dong, W.; Zhang, W.; Yang, W. Node constraint routing algorithm based on reinforcement learning. In Proceedings of the 2016 IEEE 13th International Conference on Signal Processing (ICSP), Chengdu, China, 6–10 November 2016; pp. 1752–1756.
21. Kallrath, J. Solving planning and design problems in the process industry using mixed integer and global optimization. *Ann. Oper. Res.* **2005**, *140*, 339–373. [CrossRef]
22. Qi, M.; Wang, M.; Shen, Z.J. Smart feasibility pump: Reinforcement learning for (mixed) integer programming. *arXiv* **2021**, arXiv:2102.09663.
23. Tang, Y.; Agrawal, S.; Faenza, Y. Reinforcement learning for integer programming: Learning to cut. In Proceedings of the 37th International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 9367–9376.
24. Gros, S.; Zanon, M. Reinforcement learning for mixed-integer problems based on mpc. *IFAC-PapersOnLine* **2020**, *53*, 5219–5224. [CrossRef]
25. Wu, Y.; Song, W.; Cao, Z.; Zhang, J. Learning large neighborhood search policy for integer programming. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 30075–30087.
26. Cappart, Q.; Moisan, T.; Rousseau, L.M.; Prémont-Schwarz, I.; Cire, A.A. Combining reinforcement learning and constraint programming for combinatorial optimization. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; Volume 35, pp. 3677–3687.
27. Xia, W.; Di, C.; Guo, H.; Li, S. Reinforcement learning based stochastic shortest path finding in wireless sensor networks. *IEEE Access* **2019**, *7*, 157807–157817. [CrossRef]
28. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018.
29. Arts, L.; Heskes, T.; de Vries, A.P. Comparing Discretization Methods for Applying Q-Learning in Continuous State-Action Space. 2017. Available online : https://www.cs.ru.nl/bachelors-theses/2017/Luuk_Arts___4396863___Comparing_Discretization_Methods_for_Applying_Q-learning_in_Continuous_State-Action_Space.pdf (accessed on 2 February 2024).
30. Sinclair, S.R.; Banerjee, S.; Yu, C.L. Adaptive discretization for episodic reinforcement learning in metric spaces. In Proceedings of the ACM on Measurement and Analysis of Computing Systems, Boston, MA, USA, 8–12 June 2020; Volume 3, pp. 1–44.
31. Baumgardner, J.; Acker, K.; Adefuye, O.; Crowley, S.T.; DeLoache, W.; Dickson, J.O.; Heard, L.; Martens, A.T.; Morton, N.; Ritter, M.; et al. Solving a Hamiltonian Path Problem with a bacterial computer. *J. Biol. Eng.* **2009**, *3*, 1–11. [CrossRef]
32. Tuncel, E.; Zeid, A.; Kamarthi, S. Solving large scale disassembly line balancing problem with uncertainty using reinforcement learning. *J. Intell. Manuf.* **2014**, *25*, 647–659. [CrossRef]
33. Lambert, A.; Gupta, S. *Disassembly Modeling for Assembly, Maintenance, Reuse and Recycling*; CRC Press: Boca Raton, FL, USA, 2004.