Solving Hard Bi-objective Knapsack Problems Using Deep Reinforcement Learning

Hadi Charkhgard[a], Hanieh Rastegar Moghaddam[a], Ali Eshragh[b], Sasan Mahmoudinazlou[a]

[a] *Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL, USA*
[b] *Carey Business School, Johns Hopkins University, Washington, D.C., USA*

Abstract

We study a class of bi-objective integer programs known as bi-objective knapsack problems (BOKPs). Our research focuses on the development of innovative exact and approximate solution methods for BOKPs by synergizing algorithmic concepts from two distinct domains: multi-objective integer programming and (deep) reinforcement learning. While novel reinforcement learning techniques have been applied successfully to single-objective integer programming in recent years, a corresponding body of work is yet to be explored in the field of multi-objective integer programming. This study is an effort to bridge this existing gap in the literature. Through a computational study, we demonstrate that although it is feasible to develop exact reinforcement learning-based methods for solving BOKPs, they come with significant computational costs. Consequently, we recommend an alternative research direction: approximating the entire nondominated frontier using deep reinforcement learning-based methods. We introduce two such methods, which extend classical methods from the multi-objective integer programming literature, and illustrate their ability to rapidly produce high-quality approximations.

Keywords: Multi-objective optimization; Bi-objective Knapsack Problem; Deep Reinforcement Learning; Q-learning

1. Introduction

Multi-objective optimization is an active area of research that primarily focuses on achieving a comprehensive understanding of the trade-offs between conflicting objective functions of an optimization problem. Its main aim is to identify Pareto-optimal solutions, which are solutions where it is impossible to improve one objective without adversely affecting at least one other objective. Over the past two decades, significant attention has been directed toward the development of numerous exact and heuristic solution methods designed to address a specific class of multi-objective optimization problems known as multi-objective integer linear programs [4, 5, 8, 9, 11, 14, 18, 20, 27, 28]. In this class of optimization problems, all constraints and objective functions are linear, and all decision variables are constrained to integer values exclusively.

The heightened interest in this area is, in part, a result of the substantial advancements in single-objective optimization solvers and the fact that a vast number of practical problems can be formulated as multi-objective integer programs. While substantial progress has been made in algorithmic advancements within multi-objective integer programming, there remain several underexplored research areas, primarily driven by the rapid evolution of machine learning. This trend is exemplified by the recent surge in efforts to apply machine learning techniques to enhance

single-objective integer linear programming methods [3]. In contrast, such endeavors have been relatively limited in the field of multi-objective integer linear programming. This discrepancy is not surprising, considering that similar initiatives typically reach the multi-objective optimization domain with some delay due to its inherently greater complexity when compared to its single-objective optimization counterparts. In fact, even the existing, albeit limited, efforts have primarily been in the opposite direction, i.e., utilizing multi-objective optimization techniques to improve machine learning methods [1, 23].

The gap becomes even more pronounced within a prominent machine learning paradigm known as (deep) Reinforcement Learning (RL), which naturally aligns with sequential single-objective decision-making problems involving uncertainty. Deep RL emerged approximately a decade ago, demonstrating its efficacy in conquering Atari games [15]. Several years after its emergence, researchers began to explore the prospect of employing RL to solve specific classes of single-objective integer programming problems without uncertainty. Such optimization problems need to be transformed into a sequential decision-making problem before being solved. One of the initial endeavors in this direction was undertaken by [16], who sought to solve vehicle routing problems. Since then, numerous studies have explored this research avenue [12, 21]. However, comparable endeavors in the domain of multi-objective integer programming have been scarce, and this study aims to bridge this gap.

In the context of single-objective integer linear programming, the absence of uncertainty presents a considerable challenge when designing deep RL methods: Achieving superior performance compared to state-of-the-art 'custom-built' heuristics in this domain can be hard, i.e., evaluations typically involve comparisons with other RL-based methods or general-purpose solution methods. Consequently, existing studies have predominantly focused on evaluating the effectiveness of these methods in solving single-objective integer linear programming problems rather than their competitive edge [10, 13]. The hope is that as this research domain matures, deep RL methods can either stand on their own as competitive methods or contribute to the development of hybrid methods that exhibit competitiveness. A similar trajectory is anticipated in the domain of multi-objective integer linear programming.

With this consideration, in this study, we investigate how fundamental ideas from both multi-objective optimization and reinforcement learning can be effectively combined to solve a specific class of multi-objective integer programs known as Bi-objective Knapsack Problems (BOKPs). Our aim is to present our proposed methods in a clear and understandable manner, making them accessible even to readers who may not be familiar with multi-objective integer programming and/or RL. This will also help readers envision potential directions for future research. To achieve this goal, we begin by demonstrating the development of an exact RL-based method by blending a customized adaptation of the ε-constraint method from multi-objective optimization with Q-learning from the RL field. Unfortunately, as we show through a small computational experiment, exact RL-based methods tend to be computationally slow and require individualized training for each problem instance. Nevertheless, this detailed presentation of the exact RL-based approach serves as a foundation for introducing the essential principles required for our proposed approximate RL-based techniques.

Specifically, we introduce two approximate RL-based methods that enable the effective generation of a single deep neural network for approximating nondominated frontiers instead of training many networks to produce points from different regions in the criterion space [25]. The first method is an extension of the proposed exact RL-based approach which combines a customized variation of

the ε-constraint method with the *deep* Q-learning. The second method combines deep Q-learning with the weighted sum method. A notable advantage of the proposed approximate RL-based methods is their ability to be trained for a class of instances without the need for instance-specific training. As mentioned earlier, it is not practical to benchmark RL-based methods against state-of-the-art solutions, as these methods are still in their early stages of development. However, we can illustrate their potential. To demonstrate this potential, we focus on a hard class of BOKPs, which are known to pose significant challenges for existing solution methods even when there is only a single objective [26, 22]. These hard BOKPs serve as an ideal testing ground for evaluating the effectiveness of our proposed approximate RL-based methods. In a computational analysis, we first demonstrate that our methods provide high-quality approximations for small-sized instances, where the exact nondominated frontier can be computed. For larger instances, we compare the performance of our methods against a state-of-the-art heuristic [30], and show that it can rapidly generate better approximations.

The rest of this paper is organized as follows: In Section 2, we provide some preliminary concepts. Section 3 elaborates on the proposed exact RL-based method. Section 4 provides an in-depth explanation of the proposed approximate RL-based methods. Section 5 encompasses a computational study. Finally, in Section 6, we offer concluding remarks.

2. Preliminaries

A biobjective optimization problem can be stated as follows,

$$\begin{aligned} &\max \{z_1(\boldsymbol{x}), z_2(\boldsymbol{x})\} \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned} \tag{1}$$

where $\mathcal{X}$ represents the *feasible set in the decision space*, and $z_k(\boldsymbol{x})$ for each $k = 1, 2$ represents an objective function. We denote all the vectors using bold fonts in this study. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $\boldsymbol{z} = \{z_1, z_2\}$ represents the *feasible set in the objective/criterion space*, i.e., $\mathcal{Y} := z(\mathcal{X}) := \{\boldsymbol{y} \in \mathbb{R}^2 : \boldsymbol{y} = \boldsymbol{z}(\boldsymbol{x}) \text{ for some } \boldsymbol{x} \in \mathcal{X}\}$.

Definition 1. *A feasible solution $\boldsymbol{x}' \in \mathcal{X}$ is called efficient or Pareto optimal, if there is no other $\boldsymbol{x} \in \mathcal{X}$ such that $z_k(\boldsymbol{x}) \geq z_k(\boldsymbol{x}')$ for $k = 1, 2$ and $\boldsymbol{z}(\boldsymbol{x}) \neq \boldsymbol{z}(\boldsymbol{x}')$. If $\boldsymbol{x}'$ is efficient, then $\boldsymbol{z}(\boldsymbol{x}')$ is called a nondominated point. The set of all efficient solutions $\boldsymbol{x}' \in \mathcal{X}$ is denoted by $\mathcal{X}_E$. The set of all nondominated points $\boldsymbol{z}(\boldsymbol{x}') \in \mathcal{X}$ for some $\boldsymbol{x}' \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N$ and referred to as the nondominated frontier.*

The focus of this study is on a class of bi-objective optimization problems known as Bi-Objective Knapsack Problem (BOKPs) where

$$\mathcal{X} := \{\boldsymbol{x} \in \{0,1\}^n : \sum_{i=1}^{n} w_i x_i \leq W\},$$

and

$$z_k(\boldsymbol{x}) := \sum_{i=1}^{n} c_i^k x_i \qquad \forall k \in \{1, 2\}.$$

It is assumed that all coefficients/parameters are non-negative integers, i.e., $\boldsymbol{w} \in \mathbb{Z}_+^n$, $W \in \mathbb{Z}_+$, and $\boldsymbol{c}^k \in \mathbb{Z}_+^n$ for each $k = 1, 2$. A desirable characteristic of any Binary Knapsack Optimization Problem (BOKP) lies in its finite nondominated frontier, leading to the availability of numerous algorithms for solving any BOKP. In the remaining of this section, we introduce two classical algorithms for solving BOKPs, which we later adapt to develop reinforcement learning-based algorithms aimed at solving BOKPs. The first approach is an exact method, while the second is an approximation method.

A well-known classical exact method to generate the nondominated frontier of a BOKP is the augmented ε-constraint method [7, 17]. The algorithm solves the following *scalarization* problem, i.e., single-objective integer linear program, at each iteration,

$$\begin{aligned} \max_{\boldsymbol{x} \in \mathcal{X}} \ & \lambda z_1(\boldsymbol{x}) + z_2(\boldsymbol{x}) = \sum_{i=1}^{n} (\lambda c_i^1 + c_i^2) x_i \\ \text{s.t.} \ & \sum_{i=1}^{n} c_i^2 x_i \geq C_2^* + \varepsilon, \end{aligned} \tag{2}$$

where C_2^* represents a parameter that acts as a lower bound on the second objective function's value, while ε takes on a sufficiently small value, and λ adopts a sufficiently large value. It is important to note that the value of C_2^* undergoes changes after each iteration, whereas the other two parameters are fixed from the beginning. Specifically, considering our assumption that all parameters are integers, it is safe to assign $\varepsilon = 1$ and $\lambda = 1 + \sum_{i=1}^{n} c_i^2$. The latter choice implies that the second objective function should only be optimized if there is no further way to enhance the first objective function's value, even by a single unit. Essentially, the primary objective of the method during each iteration is to optimize the first objective function.

Algorithm 1: The augmented ε-constraint method

```
Inputs: An instance of BOKP
C_2^* ← −1
Y_N ← ∅
SearchDone← False
while SearchDone is False do
    x^* ← Eps(C_2^*)
    if x^* = Null then
        SearchDone← True
    else
        z^* ← (Σ_{i=1}^n c_i^1 x_i^*, Σ_{i=1}^n c_i^2 x_i^*)
        Add z^* to Y_N
        C_2^* ← z_2^*
return Y_N
```

Algorithm 1 details the procedure of the augmented ε-constraint method. In this algorithm, we use the operation $\texttt{Eps}(C_2^*)$ to show that Problem (2) is solved for a given C_2^*. The output of the operation $\texttt{Eps}(C_2^*)$ is an optimal solution $\boldsymbol{x}^*$ for Problem (2) in each iteration. If Problem (2) is *infeasible*, the operation $\texttt{Eps}(C_2^*)$ sets $\boldsymbol{x}^*$ to *null*. Importantly, during any iteration, if $\boldsymbol{x}^* \neq$ *null*, then $\boldsymbol{x}^*$ is known to be an efficient solution. Thus, in such cases, $\boldsymbol{z}^* := (\sum_{i=1}^{n} c_i^1 x_i^*, \sum_{i=1}^{n} c_i^2 x_i^*)$ is a

nondominated point, which is subsequently appended to the list of previously identified nondominated points. Initially, C_2^* can be set to -1 (or $-\infty$). Following the discovery of a nondominated point $\boldsymbol{z}^*$ within an iteration, the ensuing task involves locating the subsequent closest nondominated point to it in the next iteration. This succeeding point's second objective function value is strictly greater than z_2^*. Consequently, at the end of each iteration, C_2^* can be updated to z_2^*. The algorithm terminates as soon as $\texttt{Eps}(C_2^*)$ returns *null*, denoting the successful identification of all nondominated points.

Algorithm 2: The weighted sum method

```
Inputs: An instance of BOKP
Ŷ_N ← ∅
SearchDone← False
while SearchDone is False do
    if there is no other value for θ that needs to be explored then
        SearchDone← True
    else
        Choose a new value for θ
        x* ← WSM(θ)
        z* ← (Σ_{i=1}^n c_i^1 x_i^*, Σ_{i=1}^n c_i^2 x_i^*)
        Add z* to Ŷ_N
return Ŷ_N
```

The second classical method is known as *the weighted sum method* in the literature of multi-objective optimization. This method is known to be only an approximate method for non-convex bi-objective optimization problems such as BOKPs. Specifically, this method does not guarantee the generation of all nondominated points of BOKPs; instead, it produces a subset of nondominated points denoted by $\hat{\mathcal{Y}}_N$. The weighted sum method is outlined in Algorithm 2. At each iteration, it solves the following scalarization problem:

$$\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{i=1}^{n} \theta c_i^1 x_i + (1-\theta) c_i^2 x_i = \sum_{i=1}^{n} (\theta c_i^1 + (1-\theta) c_i^2) x_i \tag{3}$$

where $\theta \in (0,1)$. We utilize the operation $\texttt{WSM}(\theta)$ to show that Problem (3) is solved in Algorithm 2 for a given θ. It is worth mentioning that selecting the value of θ can be systematically accomplished, requiring only a finite number of choices [2, 19]. However, it is not trivial how such systematic schemes can be utilized in the context of RL. So, instead, we adopt a straightforward yet effective scheme in this study for developing RL-based solution methods, see Section 4.

3. An Exact RL-based Method

In this section, we propose an exact RL-based method for solving BOKPs. To achieve this, we combine and utilize ideas from two distinct fields: Q-learning from the RL literature and a modified version of the ε-constraint method from the multi-objective optimization literature that we call it *pseudo augmented ε-constraint method*. It is important to note that we deliberately abstain from employing the weighted sum method in this section. The rationale is that developing an exact

RL-based method by building upon an approximate technique such as the weighted sum method poses non-trivial challenges. Throughout this section, our proposed method is referred to as *pseudo augmented ε-constraint Q-learning*. The proposed method is specifically designed to "learn" to generate an optimal solution of the following problem,

$$\begin{aligned} \max_{\boldsymbol{x}\in\mathcal{X}} \lambda z_1(\boldsymbol{x}) + z_2(\boldsymbol{x}) &= \sum_{i=1}^{n} (\lambda c_i^1 + c_i^2) x_i \\ \text{s.t.} \sum_{i=1}^{n} c_i^1 x_i &\leq C_1^*. \end{aligned} \tag{4}$$

for different values of C_1^* where C_1^* is a parameter that acts as an upper bound for the value of the first objective function. As an aside, in this study, we sometimes use the operation $\texttt{PseudoEps}(C_1^*)$ to show that Problem (4) is solved for a given C_1^*. The learning process will be conducted using Q-learning, a method that is guaranteed to yield an optimal solution for Problem (4) with sufficient time allocation [29]. Therefore, instead of using standard exact optimization techniques to solve Problem (4), we choose to adopt a methodology centered on learning. It is important to highlight that Problem (4) stands apart from Problem (2), which constitutes the core of Algorithm 1. This distinction leads us to call it the pseudo ε-constraint optimization problem. The use of the term "pseudo" arises from the fact that its optimal solutions are not necessarily efficient (or Pareto optimal) for a given C_1^* value. Nonetheless, in accordance with Proposition (1), when considering the complete spectrum of C_1^* values, the entire nondominated frontier can be derived (after eliminating dominated points).

Proposition 1. *For every efficient solution $\boldsymbol{x}'$, there exists a corresponding $C_1^* \in \mathbb{Z}$ such that $\boldsymbol{x}'$ is an optimal solution for Problem* (4).

Proof. We prove by contradiction. Suppose that there exists an efficient solution $\boldsymbol{x}'$ but it is not an optimal solution for Problem (4) for any value of $C_1^* \in \mathbb{Z}$. We know that $z_1(\boldsymbol{x}') \in \mathbb{Z}$ because by assumptions all parameters of any BOKP are integers. So, we can set $C_1^* = z_1(\boldsymbol{x}')$, and consider any optimal solution of Problem (4), denoted by $\boldsymbol{x}^*$. We also note that by construction, because $\lambda = 1 + \sum_{i=1}^{n} c_i^2$, when solving Problem (4), $z_2(\boldsymbol{x})$ will be optimized only if there is no way to improve the value of $z_1(\boldsymbol{x})$ further even by one unit. With this in mind, since C_1^* is a bound on the value of $z_1(\boldsymbol{x})$ in Problem (4), we know that $z_1(\boldsymbol{x}^*) \leq C_1^*$. However, it is possible to achieve this bound because $\boldsymbol{x}'$ is a feasible solution with $z_1(\boldsymbol{x}') = C_1^*$. So, we must have that $z_1(\boldsymbol{x}^*) = C_1^*$ too, i.e., $z_1(\boldsymbol{x}') = z_1(\boldsymbol{x}^*)$. Now, we know that $\boldsymbol{x}'$ is not an optimal solution of Problem (4) for any C_1^*. So, we must have that

$$\lambda z_1(\boldsymbol{x}') + z_2(\boldsymbol{x}') = \sum_{i=1}^{n} (\lambda c_i^1 + c_i^2) x_i' < \sum_{i=1}^{n} (\lambda c_i^1 + c_i^2) x_i^* = \lambda z_1(\boldsymbol{x}^*) + z_2(\boldsymbol{x}^*).$$

Therefore, since $z_1(\boldsymbol{x}') = z_1(\boldsymbol{x}^*)$, we must have that $z_2(\boldsymbol{x}') < z_2(\boldsymbol{x}^*)$. So, $\boldsymbol{x}'$ cannot be an efficient solution because it is dominated by $\boldsymbol{x}^*$ which is a contradiction. □

In summary, our proposed method systematically traverses the complete spectrum of C_1^* values, utilizing Q-learning to generate optimal solutions. After generating optimal solutions for all C_1^* values, inferior solutions are filtered out, leaving the entire nondominated frontier for reporting. It

is noteworthy that our decision to employ Problem (4) within our proposed technique stems from a key rationale. In the context of BOKPs, Problem (4) is always *feasible*. This is because it is always possible to create a feasible solution by setting all decision variables equal to zero. This intrinsic feasibility makes the development an RL approach for Problem (4) more convenient as such a method naturally focuses on maximizing total rewards rather than directly addressing issues related to infeasibility.

3.1. A Markov Decision Process Formulation

In order to develop an RL approach for learning to generate an optimal solution of Problem (4), it is important to explain how Problem (4) can be formulated as a Markov Decision Process (MDP) for any arbitrary value of C_1^*. Note that since Problem (4) is deterministic, the MDP formulation will also be deterministic. In other words, the proposed MDP is basically a dynamic program. Our proposed MDP formulation follows an episodic structure, spanning n steps. During each time step t, a decision is made regarding the value of the decision variable x_t in Problem (4). Consequently, only two potential actions are considered: setting x_t to either zero or one. At each time step t, the agent receives a reward denoted by R, contingent upon the action taken. Specifically, when the action is $x_t = 0$, the reward is $R = 0$, and when the action is $x_t = 1$, the reward is $R = \lambda c_t^1 + c_t^2$. During time step t, the state of the system is represented as $S = (t, \mathcal{W}_t, \mathcal{C}_t)$, wherein $\mathcal{W}_t$ signifies the remaining portion of W at time t, computed as $\mathcal{W}_t = W - \sum_{i=1}^{t-1} w_i x_i$. Similarly, $\mathcal{C}_t$ indicates the remaining part of C_1^* at time step t, calculated as $\mathcal{C}_t = C_1^* - \sum_{i=1}^{t-1} c_i^1 x_i$. Note that the collective set of all states, denoted by $\mathbb{S}$, can be defined as

$$\mathbb{S} := \{(t, \mathcal{W}, \mathcal{C}) \in \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} : 1 \leq t \leq n \text{ and } \max(0, W - \sum_{i=1}^{t-1} w_i) \leq \mathcal{W} \leq W \text{ and } 0 \leq \mathcal{C} \leq \sum_{i=1}^{n} c_i^1\}.$$

It is also important to consider that in certain states, taking action 1 might not be feasible. This restriction applies when either $w_t > \mathcal{W}_t$ or $c_t^1 > \mathcal{C}_t$. Under these circumstances, at state S, only action 0 can be selected. Given this consideration, for every state $S \in \mathbb{S}$, the Bellman optimality equation can be expressed as follows:

$$v^*(S) = \begin{cases} v^*(S^0) & \text{Taking action 1 at state } S \text{ is infeasible.} \\ \max\{R(S) + v^*(S^1), v^*(S^0)\} & \text{Otherwise.} \end{cases} \tag{5}$$

In this context, $v^*(S)$ represents the optimal return (or total rewards) when initiating from state S, and $R(S)$ corresponds to the reward attained at state S upon selecting action 1. It is important to recognize that, by design, a maximum of two other states can be observed from state S: namely, S^0 and S^1. Here, S^0 signifies the state observed following the choice of action 0 at state S, while S^1 denotes the state observed after selecting action 1 at state S. In cases where S serves as the terminal state (i.e., $t = n$), no further states can be derived thereafter. In such instances, we simply establish $v^*(S^0) = v^*(S^1) = 0$ to denote the absence of new states following the terminal state S.

3.2. Training: The Proposed RL-based Method

Within the field of RL, several methodologies are available to solve the proposed MDP. A portion of these methods involves running a sufficient number of simulation episodes to learn how to solve an MDP formulation, i.e., solving its Bellman optimality equation. In this study, we adopt

a specific approach known in RL literature as Q-learning [29]. Therefore, it is crucial to clarify the fundamental concept behind Q-learning. In our proposed MDP formulation, an alternate method to express $v^*(S^0)$ and $R(S)+v^*(S^1)$ is by employing the notation $Q^*(S,0)$ and $Q^*(S,1)$, respectively. In other words, we define

$$Q^*(S,0) = v^*(S^0), \tag{6}$$

$$Q^*(S,1) = R(S) + v^*(S^1). \tag{7}$$

In simple terms, $Q^*(S,A)$ represents the optimal return (or total rewards) that can be achieved when initiating from state S and taking action $A \in \{0,1\}$ in that state. Observe that one can use the provided Bellman optimality Equation (5) to replace $v^*(S^0)$ and $v^*(S^1)$ with their equivalent Q-functions in Equation (6)-(7). This replacement leads to the formulation of the Bellman optimality equations based on Q-functions, which can be expressed as follows:

$$Q^*(S,0) = \begin{cases} Q^*(S^0,0) & \text{Taking action 1 at state } S^0 \text{ is infeasible,} \\ \max\{Q^*(S^0,1), Q^*(S^0,0)\} & \text{Otherwise,} \end{cases} \tag{8}$$

$$Q^*(S,1) = \begin{cases} R(S) + Q^*(S^1,0) & \text{Taking action 1 at state } S^1 \text{ is infeasible,} \\ R(S) + \max\{Q^*(S^1,1), Q^*(S^1,0)\} & \text{Otherwise.} \end{cases} \tag{9}$$

The underlying idea behind Q-learning is to learn the values of $Q^*(S,1)$ and $Q^*(S,0)$ for each state $S \in \mathbb{S}$ based on Equations (8)-(9) through running a series of simulation episodes. The learning process begins with arbitrary initializations of Q-values and progresses by refining them across simulation episodes until they converge. In this context, convergence refers to the point where the left-hand-side values in Equations (8)-(9) approach (ideally, become equal to) the corresponding right-hand-side values for each state $S \in \mathbb{S}$. Once the Q-values are learned, determining an optimal solution for Problem (4) becomes straightforward. Specifically, when observing state $S \in \mathbb{S}$, if $Q^*(S,1) \geq Q^*(S,0)$, then action 1 is optimal; otherwise, action 0 is the optimal choice. With this foundation established, we next introduce our proposed method, i.e., the pseudo augmented ε-constraint Q-learning, for solving any BOKP.

Our proposed method is illustrated in Algorithm 3. For a given BOKP instance, our approach initiates by learning the optimal Q-values for the maximum value of C_1^*, which corresponds to $\sum_{i=1}^{n} c_i^1$ (Line 3). Once this is learned, the algorithm decreases the value of C_1^* by 1 (as indicated in Line 21), and the process is repeated to learn optimal Q-values for the new C_1^* value. The process will be repeated as long as the value of C_1^* remains non-negative. To learn optimal Q-values for a specific C_1^* value, the algorithm commences with initializing the Q-values (Line 2). We note that in terms of implementation, it is more efficient to conduct the initialization step on the fly, i.e., we initialize a state if the state needs to be created. This strategy is employed because not all states within $\mathbb{S}$ may be feasible for a BOKP instance, hence there is no requirement to create all states upfront. We also note that, in theory, we can initialize the Q-values randomly, so in our implementation, we set them to random small values within the range $[0.2, 0.3]$.

To learn the optimal Q-values for a specific C_1^* value, we run E number of episodes (Line 5). Note that E is a user-defined parameter, and it should be sufficiently large to ensure the convergence of the Q-values. Each episode encompasses exactly n time steps, mirroring the number of variables in the BOKP instance. At each time step t, the available actions narrow down to two choices:

assigning x_t to zero or one. At the beginning of each episode (Line 6), the initial state is defined as $(1, \mathcal{W}, \mathcal{C})$, with $\mathcal{W}$ set to W and $\mathcal{C}$ to C_1^*. During time step t, an action denoted by A is selected to represent the value of the decision variable x_t. Naturally, if $w_t > \mathcal{W}$ or $c_t^1 > \mathcal{C}$, action A must be set to zero. Otherwise, its value can be randomly determined using the current Q-values following the well-established epsilon-greedy strategy (Line 8). This strategy implies that with a probability of $1 - \epsilon$, the action with the highest Q-value is chosen, while with a probability of ϵ, a random action is selected. It is important to note that ϵ is a small positive value, distinct from the one introduced for the ε-constraint method in Section 2. In our implementation, we dynamically adjust the value of ϵ to strike a better balance between exploration and exploitation, where, specifically, we set $\epsilon = 1 - 0.99e^{\frac{episode-E}{E}}$.

Algorithm 3: The pseudo augmented ε-constraint Q-learning

```
1  Inputs: An instance of BOKP, a step-size for learning denoted by α, and the number of episodes
     denoted by E
2  Initialize Q*(S, A) randomly for all S ∈ 𝕊 and A ∈ {0, 1}
3  C*_1 ← Σ_{i=1}^{n} c_i^1
4  while C*_1 ≥ 0 do
5      foreach episode ∈ {1, ..., E} do
6          W ← W and C ← C*_1 and S ← (1, W, C)
7          foreach step t ∈ {1, ..., n} do
8              Choose an action at state S and denote it by A. Specifically, if w_t > W or c_t^1 > C then
                 choose A = 0. Otherwise, use the epsilon-greedy approach
9              R ← (λc_t^1 + c_t^2)A where R is the reward observed
10             if t = n then
11                 G ← 0
12                 Q*(S, A) ← Q*(S, A) + α[(R + G) − Q*(S, A)]
13             else
14                 W ← W − w_t A and C ← C − c_t^1 A and S' ← (t + 1, W, C)
15                 if w_{t+1} > W or c_{t+1}^1 > C then
16                     G ← Q*(S', 0)
17                 else
18                     G ← max{Q*(S', 0), Q*(S', 1)}
19                 Q*(S, A) ← Q*(S, A) + α[(R + G) − Q*(S, A)]
20                 S ← S'
21     C*_1 ← C*_1 − 1
```

It is evident that by selecting action A in state S, two outcomes follow: a reward (denoted by R) is obtained, and a new state (denoted by S') is reached. The observed reward is a direct result of the contribution of the value of x_t to the objective function of the scalarization Problem (4). In simple terms, R can be calculated as $(\lambda c_t^1 + c_t^2)A$, Line 9. It is important to note that the new state S' can be observed only if $t < n$, meaning we are not yet at the terminal state. When $t < n$, the new state is straightforwardly represented as $(t+1, \mathcal{W} - w_t A, \mathcal{C} - c_t^1 A)$, as indicated in Line 14. Subsequently, the Q-values are updated through an exponential moving average procedure (Lines 12 and 19):

$$\text{NewEstimate} = \text{OldEstimate} + \alpha(\text{Target} - \text{OldEstimate})$$

where $\alpha \in [0,1]$ is user-defined parameter representing the step-size for learning (default: $\alpha := \epsilon$). 'Target' can be defined as $R+G$ where R is the reward obtained after taking action A at state S, and G is an estimate of the maximum total rewards that can be achieved initiating from the new state, i.e., S'. We note that one can view the 'OldEstimate' as the estimation of the left-hand-side value and 'Target', i.e., $R+G$, as the estimation of the right-hand-side value in Equations (8)-(9). So, we observe that $G = 0$ if $t = n$, i.e., we are at the terminal state (Line 11). Otherwise, G is equal to $\max\{Q^*(S',0), Q^*(S',1)\}$, see Line 18. However, we know that taking action 1 is not feasible at the state S' if $w_{t+1} > \mathcal{W} - w_t A$ or $c^1_{t+1} > \mathcal{C} - c^1_t A$. So, in that case, G can be set to $Q^*(S',0)$, see Line 16, instead of $\max\{Q^*(S',0), Q^*(S',1)\}$. Finally, after updating the Q-values, the new state S can be set to S' and the next time step can start, i.e., Line 20.

3.3. Testing: Greedy Decoding

Upon completing the learning process for a given BOKP instance, we can employ a greedy decoding approach to generate its entire nondominated frontier. To begin, an initially empty list of nondominated points, denoted by $\mathcal{Y}_N$, is created. This list will ultimately encompass all nondominated points upon the termination of the greedy decoding approach. To populate this list, we run $1+\sum_{i=1}^n c^1_i$ number of episodes to generate $1+\sum_{i=1}^n c^1_i$ number of solutions. Each episode can yield a potentially efficient (or Pareto-optimal) solution, which translates to a prospective nondominated point in the criterion space. To start episode $e \in \{0,\ldots,\sum_{i=1}^n c^1_i\}$, the initial state is set to $S = (t, \mathcal{W}, \mathcal{C})$, with $t = 1$, $\mathcal{W} = W$, and $\mathcal{C} = \sum_{i=1}^n c^1_i - e$. Each episode encompasses precisely n steps. During time step $t \in \{1,\ldots,n\}$, the value of x_t needs to be determined. If $w_t > \mathcal{W}$ or $c^1_t > \mathcal{C}$, then x_t is set to 0. Otherwise, if $Q^*(S,1) > Q^*(S,0)$, x_t is set to 1; otherwise, it is set to 0. Subsequently, the state is updated by setting its parameters to $t+1$, $\mathcal{W} - w_t x_t$, and $\mathcal{C} - c^1_i x_t$, respectively. The next time step is then initiated. Once episode $e \in \{0,\ldots,\sum_{i=1}^n c^1_i\}$ terminates, a potential efficient solution, denoted by $\tilde{\boldsymbol{x}}$, is available. Consequently, its objective values, $\big(z_1(\tilde{\boldsymbol{x}}), z_2(\tilde{\boldsymbol{x}})\big)$, must be computed. Given that $\big(z_1(\tilde{\boldsymbol{x}}), z_2(\tilde{\boldsymbol{x}})\big)$ might represent a nondominated point, it is added to $\mathcal{Y}_N$ if no existing point in $\mathcal{Y}_N$ dominates it. Moreover, when adding $\big(z_1(\tilde{\boldsymbol{x}}), z_2(\tilde{\boldsymbol{x}})\big)$ to $\mathcal{Y}_N$, if any point in $\mathcal{Y}_N$ is dominated by $\big(z_1(\tilde{\boldsymbol{x}}), z_2(\tilde{\boldsymbol{x}})\big)$, that dominated point should be removed from $\mathcal{Y}_N$.

3.4. Numerical Results

In general, a main drawback of exact RL-based methods lies in their computational inefficiency. However, in the next section, we explain how this challenge can be effectively addressed. Meanwhile, for the purpose of illustrating the capability of the pseudo augmented ε-constraint Q-learning to derive the complete nondominated frontier of BOKPs, we perform a computational study involving 40 small instances, where the size of the state space $\mathbb{S}$ remains manageable. The set of instances is divided into four classes based on their number of decision variables. Each class is labeled by Cn where $n \in \{5, 10, 15, 20\}$ is the number of decision variables. To generate each instance, we randomly select c^1_i, c^2_i, and w_i from the discrete uniform distribution in the interval $[1, 20]$. Also, set $W = \lfloor \frac{\sum_{i=1}^n w_i}{2} \rfloor$. Table 1 summarizes the performance of the proposed exact RL-based method. Specifically, the table provides details about the minimum, average, and maximum numbers of points along with their corresponding runtimes (in minutes) for each class of instances. These results illustrate that the proposed algorithm successfully generates the entire nondominated frontier for all instances. It is worth noting that in certain cases, a significant number of episodes were necessary for the convergence of the Q-values. The runtimes serve as an indicator of the episode count required for Q-value convergence. As expected, the runtime correlates directly with both the instance size

and the quantity of nondominated points. Notably, as instance size increases, the expansion of $\mathbb{S}$ has a direct impact on the runtime.

Table 1: Performance of the proposed exact RL-based method

Class	Info	Min	Average	Max
C5	#Points	1	2	3
	Runtime (minutes)	5	7	10
C10	#Points	1	4	7
	Runtime (minutes)	36	72	125
C15	#Points	3	6	8
	Runtime (minutes)	767	1,989	3,409
C20	#Points	3	9	14
	Runtime (minutes)	6,543	14,485	22,400

4. Approximate Deep RL-based Methods

The proposed exact RL-based method in Section 3 has two main drawbacks: (1) It requires individual training for each instance, and (2) it is computationally inefficient due to the so-called "curse of dimensionality", where the number of states grows exponentially. As a result, it is not directly applicable to solving hard BOKP instances. To address these challenges, we propose two deep Q-learning-based techniques in this section. In both of these approaches, a deep learning model will be trained through reinforcement learning that its job is to estimate the optimal Q-values for various states across different instances. The main advantage of the proposed deep RL-based methods is that they do not focus on generating the optimal Q-values exactly rather they aim to visit a small fraction of states in order to train a deep learning model for predicting the optimal Q-values. Our first proposed approach is a customized version of the method proposed in Section 3. However, in the second method, we replace the pseudo augmented ε-constraint method with the weighted sum method. Building upon the empirical insights derived from our analysis of these two methodologies, as detailed in Section 5, we offer insights into prospective pathways for future research in the domain of RL-driven algorithmic advancements tailored for multi-objective optimization in Section 6.

4.1. Data Normalization

The normalization of data plays an important role in shaping the effectiveness of our proposed deep Q-learning-based methods. Following a series of trials and experiments, we advocate adopting a mechanism that numerically worked well during the course of our research. In this mechanism, each BOKP instance is assumed to be generated/sorted in such a way that $\frac{w_1}{c_1^1} \geq \frac{w_2}{c_2^1} \geq \cdots \geq \frac{w_n}{c_n^1}$. Note that in the literature of single-objective knapsack problem, the set of so-called *utility-weight ratios*, i.e., $\{\frac{c_1^1}{w_1}, \ldots, \frac{c_n^1}{w_n}\}$, denoted by $\boldsymbol{\rho}$, plays an important role in developing different variations of the well-known Dantzig greedy algorithm [6]. Such greedy algorithms can find solutions that are theoretically guaranteed to be as good as $\frac{1}{2}$-approximation. So, we are basically trying to exploit this property in our proposed mechanism.

Instances need to be normalized further once being transformed into their suitable scalarization form. Our proposed Q-learning-based methods are designed to solve different scalarization

problems. Specifically, one is designed to generate an optimal solution of the weighted sum problem, i.e., (3), for any value of θ and the other is designed to generate an optimal solution of the pseudo augmented ε-constraint problem, i.e., (4), for any value of C_1^*. Regardless of the specific scalarization problem at hand, it is crucial to normalize the corresponding parameters. For the normalization of objective function parameters, division by their maximum value suffices. Likewise, the normalization of each constraint involves dividing all constraint parameters by the maximum value found on the left-hand side of the constraint equation. For the remainder of this section, it is assumed that this normalization procedure has been executed.

4.2. The Proposed Deep Learning Model

Before providing the details of our proposed deep Q-learning-based methods, we first explain their underlying deep learning model and some details on its training process. The complexity of a deep learning model architecture depends on its specific applications. The architecture deemed appropriate for addressing BOKPs, following a sequence of trial and experimentation, is illustrated in Figure 1. In the proposed architecture, we pass the state features denoted by $\mathcal{S}$ through two hidden-layers of fully connected layers (i.e., Feed Forward) to generate the outputs. The outputs are basically the estimation of optimal Q-values for state $\mathcal{S}$, i.e., $Q^*(\mathcal{S}, 0)$ and $Q^*(\mathcal{S}, 1)$. We also use ReLU activation function for the first hidden layer and the Leaky ReLU for the second hidden layer. We note that we are intentionally using the notation $\mathcal{S}$ rather than S because unlike Section 3, we are predicting the optimal Q-values through a deep learning model in this section. So, more features are needed to describe a state in order to ensure that the deep learning model can generate reasonable estimations for the optimal Q-values even for the states that it has not observed during the training process. Next, we explain how $\mathcal{S}$ is defined for each of our proposed deep Q-learning-based method.

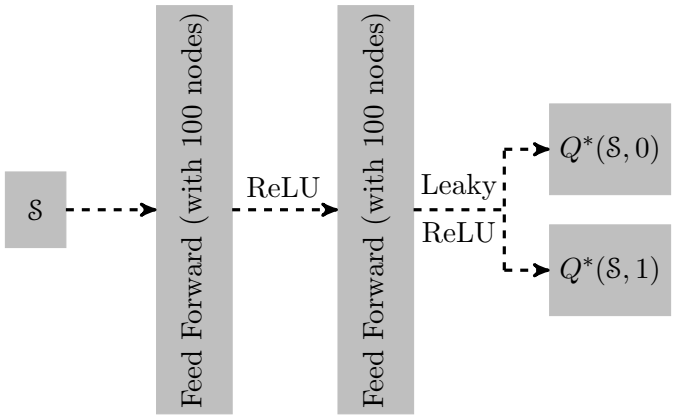


Figure 1: The proposed architecture for the deep learning model

Note that by construction at time t, our proposed methods determine the value of x_t. Keeping this in consideration, for the method tailored to solve the weighted sum scalarization problem, i.e., (3), we define $\mathcal{S}$ in the following manner at time t:

$$\mathcal{S} = (n^{[t]}, \mathcal{W}^{[t]}, c_t, w_t, \bar{\boldsymbol{\rho}}^{[t]}, \bar{\boldsymbol{c}}^{[t]}, \bar{\boldsymbol{w}}^{[t]})$$

where $n^{[t]}$ is the number of variables that their values have not yet been determined, $\mathcal{W}^{[t]}$ is the remaining knapsack budget, c_t is the coefficient of x_t in the objective function of the scalarization

problem under consideration (which is Problem (3) here), w_t is the coefficient of x_t in the knapsack constraint. Finally, a notation with "bar" has a specific meaning in this study which is designed to ensure that the cardinality of $\mathcal{S}$ is constant at all times. With this in mind, $\boldsymbol{\rho}^{[t]}$, $\boldsymbol{c}^{[t]}$, and $\boldsymbol{w}^{[t]}$ are the vectors of utility-weight ratios, the parameters of the objective function of the scalarization problem under consideration, and the knapsack weights for the variables that their values have not yet been determined at time t, respectively. Note that the cardinality of each of these vectors depends on time t and that makes the cardinality of $\mathcal{S}$ non-constant. So, instead of using them directly, we use their corresponding vectors of their *percentile* values. This implies that $\bar{\boldsymbol{\rho}}^{[t]}$ refers to

$$\texttt{Percentile}(\boldsymbol{\rho}^{[t]}),$$

where the function always returns a vector with 100 components. Component $j \in \{1, \ldots, 100\}$ is the j-th percentile among values in $\boldsymbol{\rho}^{[t]}$. Vectors $\bar{\boldsymbol{c}}^{[t]}$, and $\bar{\boldsymbol{w}}^{[t]}$ can be defined similarly, and each one has 100 components at any time too.

The state $\mathcal{S}$ can be established in a similar manner for the method designed to solve the pseudo augmented ε-constraint scalarization problem, i.e., (4). However, it necessitates the inclusion of three additional components due to the presence of an extra constraint in problem (4). Therefore, at time t, $\mathcal{S}$ can be described as follows:

$$\mathcal{S} = (n^{[t]}, \mathcal{W}^{[t]}, c_t, w_t, \bar{\boldsymbol{\rho}}^{[t]}, \bar{\boldsymbol{c}}^{[t]}, \bar{\boldsymbol{w}}^{[t]}, C_1^{*[t]}, c_t^1, \bar{\boldsymbol{c}}^{1,[t]})$$

where $C_1^{*[t]}$ is the remaining value from C_1^*, c_t^1 is the coefficient of x_t in $z_1(\boldsymbol{x})$, and $\boldsymbol{c}^{1,[t]}$ is the vector of parameters of $z_1(\boldsymbol{x})$ for the variables that their values have not yet been determined at time t. Note that $\bar{\boldsymbol{c}}^{1,[t]}$ is the vector of the percentile values of $\boldsymbol{c}^{1,[t]}$. It is also worth mentioning c_t and $\bar{\boldsymbol{c}}^{[t]}$ are linked to the specific scalarization problem at hand. As a result, they differ from the definitions provided earlier for the state in the context of the weighted sum scalarization problem.

To train our proposed deep learning model, we require a loss function. In this study, we employ the mean squared error as our selected loss function. As previously explained in Section 3.2, within the context of Q-learning, the Q-values are updated through an exponential moving average procedure:

$$\text{NewEstimate} = \text{OldEstimate} + \alpha(\text{Target} - \text{OldEstimate}).$$

Observe that "Target - OldEstimate" can be viewed as the *error*. Thus, the `Loss` function in our study is defined as the mean squared of these errors. Specifically, for each observation, the error is computed as $(R + G) - Q^*(\mathcal{S}, A)$. Therefore, our objective is to minimize the mean of $[(R + G) - Q^*(\mathcal{S}, A)]^2$ across batches of observations generated during simulation episodes in our proposed methods.

4.3. Training: The Proposed Deep RL-based Methods

In this section, we provide a detailed description of our proposed Deep RL-based methods. We have named them "deep augmented pseudo ε-constraint Q-learning" and "deep weighted sum Q-learning". While these methods share similarities during the training process, they diverge significantly during testing which we explain them in Section 4.4. The primary distinction between these methods in the training phase lies in a single key step, along with the way states are defined, as discussed earlier in Section 4.2. The proposed methods in training mode are outlined in Algorithm 4, which shares key steps with Algorithm 3. Specifically, the mechanism for computing G remains

Algorithm 4: The Proposed Deep Q-learning-based Methods

Inputs: A set of BOKP instances, and the number of episodes denoted by E
Make the size of all instances equal to n' where n' is the largest number of decision variables that exist in the BOKP instances
For each BOKP, generate several random instances for its corresponding scalarization problem, i.e., either (3) or (4), by randomly selecting values for either C_1^* or θ, and denote the index set of all such instances by $\{1, \ldots, M\}$
Generate the `DeepLearningModel` and initialize its parameters randomly
Generate a copy of the `DeepLearningModel` and denote it by `TargetModel`
Generate an empty experience replay list and denote it by `ReplayList`
foreach *episode* $\in \{1, \ldots, E\}$ **do**
    Partition the set $\{1, \ldots, M\}$ into batches of fixed size, and denote the index set of batches by $\{1, \ldots, B\}$
    foreach *batch* $\in \{1, \ldots, B\}$ **do**
        Generate the initial state $\mathcal{S}$ of each instance in the batch
        foreach *step* $t \in \{1, \ldots, n'\}$ **do**
            Call `DeepLearningModel`$(\mathcal{S})$ to estimate $Q^*(\mathcal{S}, 0)$ and $Q^*(\mathcal{S}, 1)$ for each instance in the batch
            Choose an action A (to determine the value x_t) following the epsilon-greedy approach for each instance in the batch
            Take the selected action A and observe the reward R, and generate the new state $\mathcal{S}'$ for each instance in the batch
            Record the experience observed, i.e., $(\mathcal{S}, A, Q^*(\mathcal{S}, A), R, \mathcal{S}')$, in the `ReplayList` for each instance of the batch
            Take a random sample set with an arbitrary size from the `ReplayList`
            Call `TargetModel`$(\mathcal{S}')$ to estimate $Q^*(\mathcal{S}', 0)$ and $Q^*(\mathcal{S}', 1)$ for each experience in the sample set
            Compute G based on $\max\{Q^*(\mathcal{S}', 0), Q^*(\mathcal{S}', 1)\}$ and feasibility conditions for each experience in the sample set
            Compute the squared estimation error $\left[(R + G) - Q^*(\mathcal{S}, A)\right]^2$ of each experience in the sample set, and denote the mean of squared errors by the Loss
            Apply the backpropagation to the `DeepLearningModel` based on the Loss
            Apply one step of the stochastic gradient decent method to the `DeepLearningModel`
            Set $\mathcal{S} \leftarrow \mathcal{S}'$ for each instance in the batch
            Once a while set `TargetModel` $\leftarrow$ `DeepLearningModel`

identical to the one presented in Algorithm 3. Therefore, we do not reiterate this mechanism in this section.

Keeping this in view, our proposed methods take a set of BOKP instances and the number of training episodes, denoted by E, as inputs (Line 1). Subsequently, in Line 2, the methods make the size of all BOKP instances the same, i.e, they all will have n' decision variables after transformation where n' is the largest number of decision variables among all provided instances (default: $n' = 200$). It is important to note that even if the initial BOKP instances possess different sizes, they can be transformed into BOKP instances with the fixed size. For instance, if we intend to convert a BOKP instance with 25 variables into an instance with 200 variables, we can achieve this by setting $w_j = 0$ and assigning suitably large negative values to c_j^1 and c_j^2 for each $j \in \{26, \ldots, 200\}$. This ensures that the RL agent receives a sufficiently large negative reward for setting the value of a decision variable to one if the variable does not exist. As explained in Section 4.1, we assume that the instances are already normalized in this section. Consequently, we adopt the default value of -2 as the "sufficiently large negative value". This choice is substantiated by the fact that the largest objective coefficient of the scalarization problem under consideration, following normalization, is one, making -2 a significant negative reward.

Next, in Line 3, the methods start to generate training instances (based on the provided BOKP instances) for their corresponding scalarization problem. This step constitutes the primary distinction between the deep weighted sum Q-learning and the deep augmented pseudo ε-constraint Q-learning during the training process. This divergence arises because they are based on two distinct scalarization problems. The former employs Problem (3), whereas the latter is grounded in Problem (4). Generating training instances can be achieved by generating diverse random values for either θ or C_1^* for each BOKP instance. After generating all training instances, their index set can be denoted by $\{1, \ldots, M\}$, and they will be used for training a deep learning model. Therefore, in Line 4, the methods create a random deep learning model, denoted by `DeepLearningModel`, based on the proposed architecture outlined in Figure 1. The subsequent objective is to train this `DeepLearningModel`. However, in the field of deep Q-learning, the state-of-the-art approach involves introducing a delayed mechanism (see for example [15]) for updating the parameters of the deep learning model, particularly when estimating Targets (i.e., $R + G$). This mechanism enhances the smooth training and convergence of the deep learning model. With this in mind, the methods establish a copy of `DeepLearningModel`, denoted by `TargetModel`, in Line 5. `TargetModel` will be utilized for estimating Target values, and its parameters will undergo updates with delays in Line 23 (default: every n' training time steps).

Subsequently, in Line 6, the methods create an empty list referred to as `ReplayList`, commonly known as the list of *experience replay* observations in the deep Q-learning literature [15]. This list possesses a substantial capacity (default: 1,000,000) and continuously maintains only the most recent observations when it is full. When it becomes necessary to update the `DeepLearningModel`, a random sample set of arbitrary size will be drawn from this list for both forward and backward propagations. With `ReplayList` in place, the main training loop starts at Line 7. Within each episode, the training instance set $\{1, \ldots, M\}$ is partitioned into batches of fixed size (Line 8, default: 50). For each batch, the training process is executed (Line 9). For every instance in the batch, the initial state is extracted and set to $\mathcal{S}$ (Line 10). Subsequently, the methods iterate through n' training time steps, corresponding to the number of decision variables in each training instance (Line 11).

At each training time step $t \in \{1, \ldots, n'\}$, the methods call `DeepLearningModel`($\mathcal{S}$) to estimate

$Q^*(\mathcal{S},0)$ and $Q^*(\mathcal{S},1)$ for each instance in the batch (Line 12). Subsequently, the methods employ the epsilon-greedy approach to select an action $A \in \{0,1\}$ to determine the value of x_t for each instance in the batch. Following the action selection, the methods take the chosen action A to observe a reward R and generate the next state $\mathcal{S}'$ for each instance in the batch (Line 14). It is worth noting that the way we compute the reward in Algorithm 4 differs from Algorithm 3. This difference arises from the necessity of ensuring that the deep learning model learns that certain actions are infeasible. Additionally, it needs to learn that sometimes it is more advantageous to wait and not set the value of a decision variable to one, even when there are sufficient resources available. With this in mind, we define the reward obtainable at time t by taking action A at state $\mathcal{S}$ as follows:

$$R = \begin{cases} c_t & \text{The proposed method chose } A = 1 \text{ and taking action 1 at time } t \text{ is feasible,} \\ -\beta_1 |c_t| & \text{The proposed method chose } A = 1 \text{ but taking action 1 at time } t \text{ is not feasible,} \\ -\beta_2 c_t & \text{The proposed method chose } A = 0 \text{ and } c_t \geq 0, \\ 0 & \text{The proposed method chose } A = 0 \text{ and } c_t < 0, \end{cases} \tag{10}$$

where β_1 and β_2 are user-defined non-negative penalty values (default $\beta_1 = \beta_2 = 10$).

For each instance in the batch, $(\mathcal{S}, A, Q^*(\mathcal{S},A), R, \mathcal{S}')$ constitutes an experience observed at time step t. Specifically, it signifies that at time t, we initially occupied state $\mathcal{S}$, proceeded to select action A, and estimated its corresponding optimal Q-value. We also received an immediate reward of R and ultimately transitioned to a new state $\mathcal{S}'$. Consequently, the experience $(\mathcal{S}, A, Q^(\mathcal{S},A), R, \mathcal{S}')$ is appended to the `ReplayList` in Line 15. Following this, a random sample set of arbitrary size (default: 50) is drawn from `ReplayList` in Line 16. This sample set serves as input for computing the loss function value used to train the `DeepLearningModel`. To calculate the loss function value, it is crucial to determine G for each experience within the sample set. Therefore, in Line 17, the methods call `TargetModel`$(\mathcal{S}')$ to estimate $Q^*(\mathcal{S}',0)$ and $Q^*(\mathcal{S}',1)$ for each experience in the sample set. Subsequently, in Line 18, G is computed based on $\max\{Q^*(\mathcal{S}',0), Q^*(\mathcal{S}',1)\}$ and the feasibility conditions for each experience in the sample set, mirroring the process in Lines 10-18 of Algorithm 3. Once G is computed, the loss function, which is the mean squared error denoted by Loss, is calculated. To achieve this, in Line 19, the methods compute the squared error estimation, i.e., $\left[(R+G) - Q^*(\mathcal{S},A)\right]^2$, for each experience in the sample set, and subsequently compute the average to obtain Loss. Afterwards, in Line 20, backpropagation is applied to the `DeepLearningModel` based on Loss, and in Line 21, one step of the gradient descent method is executed to update the parameters of the `DeepLearningModel`. Before starting another training step, the methods set $\mathcal{S}$ to $\mathcal{S}'$ for each instance in the batch, as indicated in Line 22.

4.4. Testing: Beam Search

Once the training process is complete, the proposed methods can be used to generate an approximate nondominated frontier of any BOKP. The deep weighted sum Q-learning and the deep augmented pseudo ε-constraint Q-learning employ distinct search mechanisms to approximate the entire nondominated frontier of a given BOKP. However, when aiming to approximate an optimal solution for their respective scalarization problems at a specific value of θ or C_1^*, they share a common procedure. In particular, both methods utilize a technique known as the *beam search* to solve

the scalarization problem. The beam search extends the concept of greedy search or decoding by maintaining a record of the best $L \in \mathbb{Z}_+$ candidates during the process of approximating an optimal solution for a scalarization problem. Here, L represents a user-defined parameter that determines the size or length of the beam search. Keeping this in mind, when solving a scalarization problem, the methods execute a single episode with n time steps.

In the context of the greedy decoding approach, when we reach time step $t \in \{1, \ldots, n\}$, we can extract the system's state, denoted by $\mathcal{S}$, and then employ the trained deep learning model to estimate both $Q^*(\mathcal{S}, 0)$ and $Q^*(\mathcal{S}, 1)$. By selecting $A = \arg\max\{Q^*(\mathcal{S}, 0), Q^*(\mathcal{S}, 1)\}$ and identifying its feasibility (i.e., not violating the constraints of the scalarization problem under consideration), we can determine the value of x_t at time t. In contrast, the beam search approach maintains a record of the best L partial solutions at each time step t. To decide which partial solutions should be maintained at any given time t, a utility function is required for evaluating the value of each partial solution. A natural choice for this utility function involves summing the rewards obtained so far by the known part of each partial solution and its corresponding Q-values (which estimate the maximum total future rewards). Consequently, partial solutions with higher utility values need to be maintained.

Upon completing the beam search to solve a scalarization problem, we have (at most) L complete solutions in memory. It is evident that the solution with the best objective value for the scalarization problem under consideration can be viewed as an approximation of its optimal solution. However, it is important to note that since we have computed L complete solutions, we can also evaluate these solutions with respect to the original objective functions, namely, $z_1(\boldsymbol{x})$ and $z_2(\boldsymbol{x})$. These solutions might be nondominated in relation to each other. Consequently, we can maintain these solutions and utilize them to approximate the entire nondominated frontier. So, a desirable property of the beam search method is its ability to discover several (locally) Pareto-optimal solutions within the same neighborhood when solving a scalarization problem. This feature is particularly valuable in the context of deep weighted sum Q-learning. As mentioned in Section 2, the weighted-sum method theoretically falls short in generating the complete nondominated frontier for non-convex problems such as BOKPs. Specifically, it fails to find some points which are referred to as *unsupported nondominated points* in the literature of multi-objective optimization [7]. However, thanks to the beam search, it becomes possible to identify some of these unsupported nondominated points when solving a scalarization problem.

With the beam search defined, we can now provide the specifics of the proposed methods in the testing mode. The details of the deep augmented pseudo ε-constraint Q-learning in the testing mode are shown in Algorithm 5. The method starts by accepting a BOKP instance and the beam search length, denoted by L, as inputs (Line 1). Subsequently, it creates an empty list, denoted by $\widetilde{\mathcal{Y}}_N$, to maintain the set of approximate nondominated points (Line 2). The method's initial goal is to approximate the endpoints of the nondominated frontier. To estimate the first endpoint, it proceeds to solve `PseudoEps`$(+\infty)$ using the beam search (Line 3). The beam search generates a maximum of L solutions, which are then added to $\widetilde{\mathcal{Y}}_N$ (Line 4). To determine the other endpoint, the method temporarily switches the objective functions of the BOKP by relabeling them in reverse order (Line 5). Following this relabeling, it uses the beam search again to solve `PseudoEps`$(+\infty)$, see Line 6. This process generates up to L solutions that are then added to $\widetilde{\mathcal{Y}}_N$ (Line 7). After estimating the second endpoint of the nondominated frontier, the method restores the original labels of the objective functions of the BOKP (Line 8).

It is worth noting that estimating both endpoints serves a crucial purpose in determining the

Algorithm 5: The Deep Augmented Pseudo ε-constraint Q-learning – Testing Mode

```
1  Inputs: A BOKP instance, and the beam search length denoted by L
2  Create an empty list denoted by Ỹ_N to maintain the approximate nondominated points
3  Solve PseudoEps(+∞) using the beam search to compute at most L solutions
4  Add the solutions found to Ỹ_N
5  Relabel the objective functions of the BOKP in reverse manner, i.e., use z_1(x) for the second
     objective function and z_2(x) for the first objective function
6  Solve PseudoEps(+∞) using the beam search to compute at most L solutions
7  Add the solutions found to Ỹ_N
8  Restore the original labels of the BOKP's objective functions
9  C_1^* ← +∞
10 SearchDone← False
11 while SearchDone is False do
12     UpperBound ← the maximum value of z_1(x) among all solutions in Ỹ_N with z_1(x) ≤ C_1^*
13     C_1^* ← UpperBound − 1
14     LowerBound← the minimum value of z_1(x) among all solutions in Ỹ_N
15     if C_1^* < LowerBound then
16         SearchDone← True
17     else
18         Solve PseudoEps(C_1^*) using the beam search to compute at most L solutions
19         Add all the solutions found in this iteration to Ỹ_N
20 return Pareto(Ỹ_N)
```

termination condition for the proposed method. This is because the endpoints of the nondominated frontier define the minimum and maximum values of $z_1(\boldsymbol{x})$ among all nondominated solutions. Consequently, we can constrain the values of C_1^* within this minimum and maximum range. With this in mind, in Lines 9-11, we initialize C_1^* to $+\infty$ and initiate a search loop (Lines 10-11). In each iteration of the loop, the method aims to compute a new UpperBound for C_1^*. This is achieved by searching through all solutions in $\widetilde{\mathcal{Y}}_N$ with $z_1(\boldsymbol{x}) \leq C_1^*$ and identifying the maximum value of $z_1(\boldsymbol{x})$ among those solutions (Line 12). After computing UpperBound, C_1^* is updated in Line 13. Specifically, it is set to UpperBound $-\ 1$ in the hope of discovering solutions distinct from those already present in $\widetilde{\mathcal{Y}}_N$. Subsequently, the algorithm computes a LowerBound for C_1^*. This is achieved by searching through all solutions in $\widetilde{\mathcal{Y}}_N$ to identify the minimum value of $z_1(\boldsymbol{x})$ (Line 14). The search terminates when $C_1^* <$ LowerBound is met (Lines 15-16). If not, the beam search is utilized to solve `PseudoEps`(C_1^*) and the solutions found are added to $\widetilde{\mathcal{Y}}_N$ (Lines 17-19). Finally, a filtering process is conducted over $\widetilde{\mathcal{Y}}_N$ to keep and report only the solutions that are nondominated in relation to each other. This process is denoted by `Pareto`$(\widetilde{\mathcal{Y}}_N)$, see Line 20.

The testing process for the deep weighted sum Q-learning is more straightforward and is outlined in Algorithm 6. The method starts by accepting a BOKP instance and the beam search length, denoted by L, as inputs (Line 1). It then creates an empty list, denoted by $\widetilde{\mathcal{Y}}_N$, to manage the set of approximate nondominated points (Line 2). Subsequently, the method initiates a loop that systematically increases the value of θ from 0 to 1 (Line 3). The increment size for θ is a user-defined parameter, which is set to 0.01 in this study. For each value of θ, the method calls the beam search to solve `WSM`(θ), and all the solutions found are added to $\widetilde{\mathcal{Y}}_N$ (Lines 4-5). Finally, in

Algorithm 6: The Deep Weighted Sum Q-learning – Testing Mode

Inputs: A BOKP instance, and the beam search length denoted by L
Create an empty list denoted by $\widetilde{\mathcal{Y}}_N$ to maintain the approximate nondominated points
foreach $\theta \in \{0, 0.01, 0.02, \ldots, 1\}$ **do**
    Solve $\texttt{WSM}(\theta)$ using the beam search to compute at most L solutions
    Add all the solutions found in this iteration to $\widetilde{\mathcal{Y}}_N$
return $\texttt{Pareto}(\widetilde{\mathcal{Y}}_N)$

Line 6, the method filters and reports the results using $\texttt{Pareto}(\widetilde{\mathcal{Y}}_N)$.

5. Computational Study

In this section, we conduct a computational study to show the efficacy of the proposed deep RL-based methods. All computational experiments are executed on a Dell PowerEdge R360 equipped with dual Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB of RAM, and the RedHat Enterprise Linux 7.0 operating system. We employ Python and the PyTorch package to implement all the proposed methods in this study. Our instances and Python source codes are accessible through GitHub (`https://github.com/Multi-Objective-Optimization-Laboratory/DeepRL-BiobjectiveKP`). For the training of each of our proposed methods, we employ approximately $10,000,000$ training steps, i.e., $E \times B \times n' \simeq 10,000,000$, as outlined in Algorithm 4. Attaining this can be accomplished through the assignment of $E = 76$ and $M = 35,200$. It is important to note that in Algorithm 4, M corresponds to the count of scalarization training instances. Each instance encompasses 25 to 200 variables in this study. To elaborate, 200 scalarization instances are generated for every value of n within the range $\{25, 26, \ldots, 200\}$. As a result, the total value of M can be calculated as $176 \times 200 = 35,200$.

As mentioned in the Introduction, this study centers on hard instances of BOKPs. This choice is rooted in the potential for these complex cases to illuminate the advantages of utilizing deep RL-based methods for solving deterministic combinatorial optimization problems compared to existing highly effective exact and heuristic techniques. To address this, we adopt a technique proposed by Pisinger [22] to create a specific category of challenging BOKP instances referred to as "inverse strongly correlated instances". To elaborate, we randomly generate vectors $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$ from a discrete uniform distribution within the range $[1, R]$, where $R = 100$ in this study. The coefficients for the knapsack constraint are determined as $w_i = c_i^1 + \frac{R}{10}$ for each $i = 1, \ldots, n$. Finally, the value W is drawn from a discrete uniform distribution within the interval $[0.25 \sum_{i=1}^{p} w_i, 0.5 \sum_{i=1}^{p} w_i]$. We generate two sets of testing instances, each containing 6 classes. In each class, there are 10 instances of the same size, labeled as C_n, where n represents the number of decision variables. For the first set, we consider values of n from the set $\{25, 30, 35, 40, 45, 50\}$; for the second set, we use values of n from $\{75, 100, 125, 150, 175, 200\}$.

The rationale behind having two sets of testing instances stems from the division of our computational analysis into two distinct sections. The initial segment revolves around the utilization of the first set of instances. Its primary focus lies in demonstrating the efficacy of the proposed methods using instances where computing the exact nondominated frontier can be achieved within a 24-hour time limit. For generating the exact nondominated frontier of each instance, Algorithm 1 is employed in conjunction with Gurobi 10.0.1 to solve the corresponding single-objective opti-

mization problems. The subsequent section pertains to the utilization of the second, larger set of instances. Its objective is to assess the effectiveness of our proposed methods in contrast to a state-of-the-art heuristic approach known as the Multi-Directional Local Search (MDLS) [30].

In light of the above, for a given instance, the true nondominated frontier $\mathcal{Y}_N$ may be either known or unknown. In the latter scenario, we combine the points generated by our proposed methods and MDLS to create what we term the "best-known approximate nondominated frontier", which we treat as the true $\mathcal{Y}_N$. To assess the efficacy of any approximation method (e.g., our proposed deep RL-based approaches or MDLS), the evaluation revolves around their respective reported approximate nondominated frontiers, denoted by $\widetilde{\mathcal{Y}}_N$, using various metrics. Given that there is not a singular measure to accomplish this task, the common practice is to employ multiple metrics [24]. In the following sections, we explore several metrics adopted in this study. Interested readers may refer to [4, 20] for further details.

To introduce the metrics, we require some new notation. The Euclidean distance between two points $\boldsymbol{y}$ and $\boldsymbol{y}'$ is represented as $d(\boldsymbol{y}, \boldsymbol{y}')$. Let $k(\boldsymbol{y})$ denote the nearest point in the approximate frontier to a true nondominated point $\boldsymbol{y} \in \mathcal{Y}_N$. Lastly, for each $\boldsymbol{y} \in \widetilde{\mathcal{Y}}_N$, $n(\boldsymbol{y})$ is defined as the count of true nondominated points $\boldsymbol{y}' \in \mathcal{Y}_N \setminus \widetilde{\mathcal{Y}}_N$ where $k(\boldsymbol{y}') = \boldsymbol{y}$.

- **Hypervolume gap.** An established measure for evaluating and comparing approximate nondominated frontiers is the *hypervolume indicator* (or S-metric), which assesses the volume of the dominated region within the criterion space relative to a reference point [31]. In this study, to ensure non-negativity of the hypervolume, the reference point, denoted by $\boldsymbol{y}^R := (y_1^R, y_2^R)$, is determined by setting y_i^R to the minimum value of $z_i(\boldsymbol{x})$ among all solutions reported by different methods (used in this study) for each $i \in \{1, 2\}$. With this consideration, we define the hypervolume gap as follows:

$$\frac{100 \times (\text{ Hypervolume of } \mathcal{Y}_N - \text{ Hypervolume of } \widetilde{\mathcal{Y}}_N)}{\text{Hypervolume of } \mathcal{Y}_N}.$$

  As a result, an approximate nondominated frontier with a smaller hypervolume gap is deemed more favorable.

- **Cardinality.** We define the caridnality of an approximate nondominated frontier straightforwardly as the proportion of true nondominated points it encompasses, expressed as $\frac{100 \times |\widetilde{\mathcal{Y}}_N \cap \mathcal{Y}_N|}{|\mathcal{Y}_N|}$. Accordingly, an approximate nondominated frontier with a larger caridnality is considered better.

- **Coverage.** A straightforward coverage metric, which gauges the average distance between true nondominated points not included in the approximate frontier and their nearest counterparts within the approximate frontier, can be described as follows:

$$f^a := \frac{\sum_{\boldsymbol{y} \in \mathcal{Y}_N \setminus \widetilde{\mathcal{Y}}_N} d\big(k(\boldsymbol{y}), \boldsymbol{y}\big)}{|\mathcal{Y}_N \setminus \widetilde{\mathcal{Y}}_N|},$$

  Note that f^a can be interpreted as a gauge of how dispersed the points are within the approximate nondominated frontier. Smaller values of f^a imply that the nondominated points in the approximate frontier are spread across different areas of the criterion space. Consequently, an approximate nondominated frontier with a lower f^a is more preferable.

- **Uniformity.** An indicator of uniformity should reflect how effectively the points within an approximate frontier are distributed. Points clustered closely together do not contribute positively to the quality of an approximate frontier. Thus, we formulate the uniformity indicator as follows:

$$\mu := \frac{\sum_{\boldsymbol{y} \in \widetilde{\mathcal{Y}}_N} n(\boldsymbol{y})}{|\widetilde{\mathcal{Y}}_N|}.$$

  Therefore, an approximate nondominated frontier with a lower value of μ is considered more favorable.

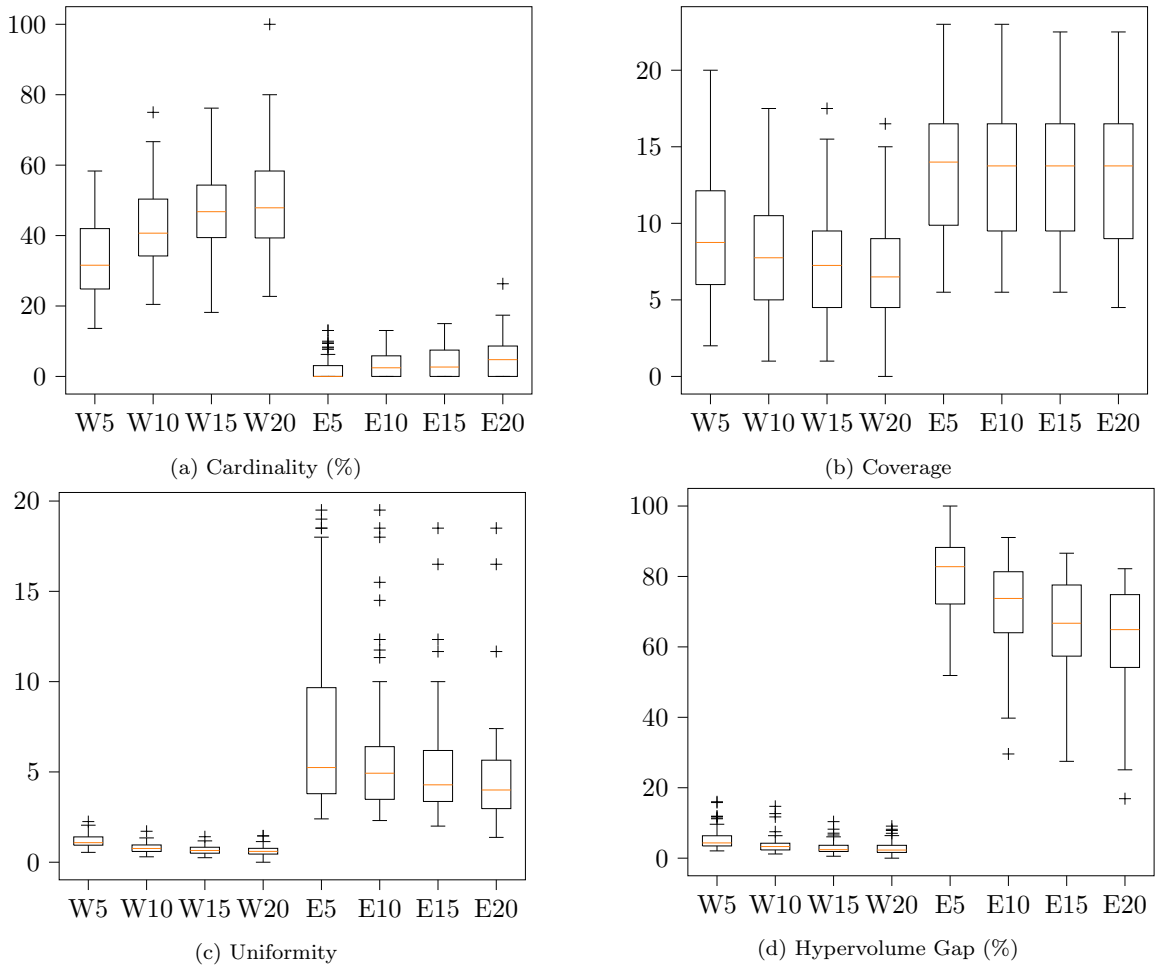
It is worth mentioning that while the training phase of the proposed deep RL-based methods demands a considerable amount of time, their efficiency during the testing phase is noteworthy. To be specific, it takes less than 2 seconds for them to solve a BOKP instance with $n = 25$ variables when the beam search size is set to one, i.e., $L = 1$. It is important to recognize that deep RL methods are inherently suited for parallelization, which can substantially enhance their speed during testing. Nevertheless, assuming a lack of parallelization, the runtime of the proposed deep RL-based methods demonstrate an almost linear correlation with the values of L and n. For instance, if we consider a BOKP instance with $n = 200$ and $L = 20$, the projected runtime can be approximated as 320 seconds using the formula $2 \times \frac{n}{25} \times L$. Given these considerations, we abstain from providing specific solution times for the proposed methods within this study.

5.1. Comparisons with an Exact Method

The goal of this section is to assess the potential effectiveness of the proposed deep RL-based methods in solving BOKPs. To achieve this goal, we evaluate the performance of the proposed methods using the first testing set. This set consists of instances where the true nondominated points are already identified. We conduct these evaluations under four different beam search sizes, i.e., L, selected from the set $\{5, 10, 15, 20\}$. It is important to note, as explained in Section 4.4, that the size of the beam search significantly influences the generation of nondominated points, including unsupported ones. Thus, a larger value of L is anticipated to yield a better approximation.

We employ the labels $W5$, $W10$, $W15$, and $W20$ to represent the deep weighted sum Q-learning method under varying beam search sizes. Similarly, the labels $E5$, $E10$, $E15$, and $E20$ denote the deep augmented pseudo ε-constraint Q-learning approach across different beam search sizes. The performance of the proposed methods across diverse metrics is depicted in Figure 2. It is evident that the performance of both methods exhibits enhancement across all metrics as the beam search size increases. Nevertheless, the deep weighted sum Q-learning method consistently outperforms the deep augmented pseudo ε-constraint Q-learning method, regardless of the specific beam search size employed.

Table 2 better highlights the superior effectiveness of the proposed deep weighted sum Q-learning method (for $L = 20$). The values presented in this table are averages computed over 10 instances. Notably, the deep weighted sum Q-learning has identified approximately 3.61 times more approximate nondominated points ($\widetilde{\mathcal{Y}}_N$) on average. This method has successfully generated around 50.5% of the true nondominated points ($\mathcal{Y}_N$) on average, which is about 10 times better than the alternate method, i.e., the deep augmented pseudo ε-constraint Q-learning. As a consequence, the average hypervolume gap of the deep weighted sum Q-learning method is about 2.9%, providing a much more accurate approximation. In contrast, the alternate method has an average hypervolume gap of 62.7%, indicating a significantly poorer performance. When considering metrics such as



(a) Cardinality (%)

(b) Coverage

(c) Uniformity

(d) Hypervolume Gap (%)

Figure 2: Performance comparisons between the proposed methods for different beam search sizes

Table 2: Performance comparisons between the proposed methods for $L = 20$

Class	$\mathcal{Y}_N$	**Deep Weighted Sum Q-Learning**					**Deep ε-Constraint Q-Learning**				
		$\widetilde{\mathcal{Y}}_N$	**Car.**	**Cov.**	**Uni.**	**Hyp.**	$\widetilde{\mathcal{Y}}_N$	**Car.**	**Cov.**	**Uni.**	**Hyp.**
C25	19.4	15.0	57.9%	3.9	0.6	3.9%	7.5	9.3%	8.3	2.7	43.1%
C30	24.4	19.1	54.8%	5.2	0.6	3.6%	7.3	6.7%	11.0	3.4	54.4%
C35	25.9	20.5	47.7%	6.3	0.7	3.4%	7.0	5.5%	11.7	3.6	63.8%
C40	30.8	24.7	51.7%	7.2	0.6	2.2%	7.4	6.6%	14.0	4.0	65.7%
C45	35.5	30.1	45.4%	9.6	0.7	2.2%	6.6	3.2%	16.7	5.5	71.6%
C50	35.5	31.8	45.7%	9.2	0.6	2.1%	4.7	1.5%	17.0	9.0	77.4%
Avg	**28.6**	**23.5**	**50.5%**	**6.9**	**0.6**	**2.9%**	**6.8**	**5.5%**	**13.1**	**4.7**	**62.7%**

uniformity and coverage, the deep weighted sum Q-learning method surpasses the alternate method by approximately 2-fold and 8-fold, respectively.

The poor performance of the proposed deep augmented pseudo ε-constraint Q-learning method can be attributed, in part, to the nature of its corresponding scalarization problem, expressed in Equation (4). This scalarization problem places a distinct emphasis on the first objective function, evident through the influence of the constant λ. This skewed focus prompts the deep learning model to exhibit a strong bias towards the first objective function. Consequently, the significance of the second objective value diminishes, rendering it akin to an approximation error. As a result, the deep augmented pseudo ε-constraint Q-learning method has the potential to yield points with significantly inaccurate second objective values. Consequently, the generated approximate nondominated frontier through this approach tends to be of low quality.

Naturally, one potential solution to address this issue is to train the model with a smaller value of λ. During the course of our research, we indeed explored this avenue and encountered two challenges. Firstly, determining an appropriate value for λ in such cases is nontrivial. More crucially, while the choice of smaller λ values leads the learning process to favor the second objective function, there is a risk of disregarding numerous nondominated points as this procedure is closely linked with determining the value of C_1^*. Hence, the effectiveness of the deep weighted sum Q-learning stems from its ability to adapt to various weights instead of relying on a fixed weight.

5.2. Comparisons with a Heuristic Method

Given that the deep weighted sum Q-learning has proven effective among our proposed methods, we employ it in this section. The goal of this section is to determine whether the proposed method, when $L = 20$, can compete with a state-of-the-art technique called MDLS [30] in solving challenging BOKPs. To accomplish this goal, we assess the performance of the proposed methods using the second testing set. This set comprises instances in which the true nondominated points are unknown. Notably, even finding a single nondominated point in this set can be exceedingly difficult when employing an exact method such as Algorithm 1. As previously mentioned, to address this challenge, we combine the sets of points identified by both methods and consider the best ones as the true set of nondominated points, i.e., $\mathcal{Y}_N$.

Table 3: Performance comparisons with MDLS for $L = 20$

Class	$\mathcal{Y}_N$	**Deep Weighted Sum Q-Learning**					**MDLS**				
		$\widetilde{\mathcal{Y}}_N$	**Car.**	**Cov.**	**Uni.**	**Hyp.**	$\widetilde{\mathcal{Y}}_N$	**Car.**	**Cov.**	**Uni.**	**Hyp.**
C75	45.1	45.0	98.0%	0.1	0.0	0.5%	24.2	3.2%	21.4	1.8	26.9%
C100	55.1	53.7	97.3%	0.3	0.0	0.0%	28.7	3.4%	26.1	1.9	26.6%
C125	60.5	59.7	98.7%	0.1	0.0	0.0%	31.3	1.5%	29.3	1.9	27.1%
C150	71.2	70.4	98.8%	0.1	0.0	0.0%	33.7	1.4%	34.6	2.1	27.2%
C175	80.8	80.0	99.0%	0.2	0.0	0.0%	35.3	1.1%	39.5	2.3	26.9%
C200	57.9	44.7	75.1%	6.8	0.3	5.2%	38.2	25.7%	20.9	1.1	11.9%
Avg	**61.8**	**58.9**	**94.5%**	**1.3**	**0.1**	**0.9%**	**31.9**	**6.1%**	**28.6**	**1.9**	**24.4%**

Table 3 presents a comprehensive summary of the performance comparisons between the proposed method and MDLS, effectively showcasing the potential of the proposed approach. To improve the quality of solutions produced by MDLS, we adjusted certain parameters, such as the number of iterations and the quantity of initial locally nondominated points, to values larger than

their default settings. Consequently, MDLS takes approximately 800 seconds to solve each of these instances. The values provided in Table 3 are averages derived from 10 instances. Notably, the deep weighted sum Q-learning has identified approximately 1.9 times more approximate nondominated points, i.e., $\widetilde{\mathcal{Y}}_N$ on average. This method has managed to generate approximately 94.5% of the "true" nondominated points, i.e., $\mathcal{Y}_N$ on average, marking a 3.3-fold improvement compared to MDLS. Consequently, the average hypervolume gap of the deep weighted sum Q-learning hovers around 0.9%, signifying a notably accurate approximation. In contrast, the alternate method exhibits an average hypervolume gap of 24.4%, illustrating a relatively lesser effectiveness. Regarding metrics such as uniformity and coverage, the deep weighted sum Q-learning method excels over the alternate method by approximately 22-fold and 19-fold, respectively.

6. Final Remarks

In this study, we introduced new exact and approximate solution methods for BOKPs by merging concepts from the (deep) Q-learning in RL with two classical scalarization methods from multi-objective integer programming: the ε-constraint method and the weighted sum method. Our findings demonstrated that combining a scalarization method capable of generating the exact nondominated frontier in multi-objective integer programming, e.g., the ε-constraint method, with an exact reinforcement learning method such as Q-learning allows us to construct an exact RL-based approach for BOKPs. However, it is essential to note that this exact method comes with computational inefficiencies. On the other hand, we also explored the possibility of creating an efficient approximate RL-based approach for BOKPs by uniting scalarization methods with an approximate RL method, i.e., the deep Q-learning.

Our numerical results brought to light three key observations: Firstly, to develop a high-quality approximate RL-based approach, it is preferable to choose a method from multi-objective optimization that dynamically adjusts the objective function weights during the search, such as the weighted sum method. Secondly, we found that the performance of our proposed methods experiences significant enhancements when we employ beam search instead of greedy decoding. This strategic shift enables the identification of numerous (locally) nondominated points within the vicinity of certain points. Lastly, our research revealed that RL-based methods exhibit superior performance when dealing with hard instances of BOKPs.

These insights have paved the way for several promising avenues for future research. One such direction involves extending the proposed deep weighted Q-learning approach to handle hard classes of bi-objective integer programs, particularly those where solving their single-objective optimizations is computationally intensive, e.g., vehicle routing problem. Achieving this is anticipated to require more advanced deep learning models at the heart of our proposed method, including but not limited to pointer networks and transformer models. The primary contribution of this research direction lies in the development of a suitable deep learning model. Another intriguing research path involves extending our approach to scenarios with more than two objective functions and exploring various scalarization techniques from the literature, customizing them to accommodate dynamic weights for the objective functions. This expansion can possibly enable the development of more effective RL-based methods. Lastly, it is worth exploring alternative deep RL methods, such as deep REINFORCE or deep Actor-critic approaches, as potential substitutes for the deep Q-learning. This exploration can help amplify the capabilities of the deep RL-based methods when solving hard multi-objective integer programs.

References

[1] Aickelin, U., Khorshidi, H. A., Qu, R., Charkhgard, H., 2023. Guest editorial special issue on multiobjective evolutionary optimization in machine learning. IEEE Transactions on Evolutionary Computation 27 (4), 746–748.

[2] Aneja, Y. P., Nair, K. P. K., 1979. Bicriteria transportation problem. Management Science 27, 73–78.

[3] Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research 290 (2), 405–421.

[4] Boland, N., Charkhgard, H., Savelsbergh, M., 2016. The L-shape search method for triobjective integer programming. Mathematical Programming Computation 8 (2), 217–251.

[5] Dächert, K., Klamroth, K., Lacour, R., Vanderpooten, D., 2017. Efficient computation of the search region in multi-objective optimization. European Journal of Operational Research 260 (3), 841 – 855.

[6] Dantzig, G. B., 1957. Discrete-variable extremum problems. Operations Research 5 (2), 266–288.

[7] Ehrgott, M., 2005. Multicriteria optimization, 2nd Edition. Springer, New York.

[8] Gadegaard, S. L., Nielsen, L. R., Ehrgott, M., 2019. Bi-objective branch-and-cut algorithms based on lp relaxation and bound sets. INFORMS Journal on Computing 31 (4), 790–804.

[9] Helfrich, S., Perini, T., Halffmann, P., Boland, N., Ruzika, S., 2023. Analysis of the weighted tchebycheff weight set decomposition for multiobjective discrete optimization problems. Journal of Global Optimization 86, 417–440.

[10] Hu, Y., Yao, Y., Lee, W. S., 2020. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. Knowledge-Based Systems 204, 106244.

[11] Kirlik, G., Sayın, S., 2014. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. European Journal of Operational Research 232 (3), 479 – 488.

[12] Kool, W., van Hoof, H., Gromicho, J., Welling, M., 2022. Deep policy dynamic programming for vehicle routing problems. In: International conference on integration of constraint programming, artificial intelligence, and operations research. Springer, pp. 190–213.

[13] Li, K., Zhang, T., Wang, R., 2021. Deep reinforcement learning for multiobjective optimization. IEEE Transactions on Cybernetics 51 (6), 3103–3114.

[14] Lokman, B., Köksalan, M., 2013. Finding all nondominated points of multi-objective integer programs. Journal of Global Optimization 57 (2), 347–365.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1. pp. 201–220.

[16] Nazari, M., Oroojlooy, A., Takáč, M., Snyder, L. V., 2018. Reinforcement learning for solving the vehicle routing problem. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS'18. Curran Associates Inc., Red Hook, NY, USA, p. 9861–9871.

[17] Özlen, M., Azizoğlu, M., 2009. Multi-objective integer programming: A general approach for generating all non-dominated solutions. European Journal of Operational Research 199, 25–35.

[18] Özlen, M., Burton, B. A., MacRae, C. A. G., 2013. Multi-objective integer programming: An improved recursive algorithm. Journal of Optimization Theory and Applications10.1007/s10957-013-0364-y.

[19] Özpeynirci, Ö., Köksalan, M., 2010. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. Management Science 56 (12), 2302–2315.

[20] Pal, A., Charkhgard, H., 2019. Fpbh: A feasibility pump based heuristic for multi-objective mixed integer linear programming. Computers & Operations Research 112, 104760.

[21] Peng, B., Wang, J., Zhang, Z., 2020. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In: Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11. Springer, pp. 636–650.

[22] Pisinger, D., 2005. Where are the hard knapsack problems? Computers & Operations Research 32 (9), 2271–2284.

[23] Reiners, M., Klamroth, K., Heldmann, F., Stiglmayr, M., 2022. Efficient and sparse neural networks by pruning weights in a multiobjective learning approach. Computers & Operations Research 141, 105676.

[24] Sayın, S., 2000. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. Mathematical Programming 87 (3), 543–560.

[25] Shao, Y., Lin, J. C.-W., Srivastava, G., Guo, D., Zhang, H., Yi, H., Jolfaei, A., 2021. Multiobjective neural evolutionary algorithm for combinatorial optimization problems. IEEE transactions on neural networks and learning systems.

[26] Smith-Miles, K., Christiansen, J., noz, M. A. M., 2021. Revisiting where are the hard knapsack problems? via instance space analysis. Computers & Operations Research 128, 105184.

[27] Soylu, B., Yıldız, G. B., 2016. An exact algorithm for biobjective mixed integer linear programming problems. Computers & Operations Research 72, 204–213.

[28] Stidsen, T., Andersen, K. A., Dammann, B., 2014. A branch and bound algorithm for a class of biobjective mixed integer programs. Management Science 60 (4), 1009–1032.

[29] Sutton, R. S., Barto, A. G., 2018. Reinforcement Learning: An Introduction, 2nd Edition. The MIT Press.

[30] Tricoire, F., 2012. Multi-directional local search. Computers & Operations Research 39 (12), 3089 – 3101.

[31] Zitzler, E., Brockhoff, D., Thiele, L., 2007. The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (Eds.), Evolutionary Multi-Criterion Optimization. Vol. 4403 of Lecture Notes in Computer Science. pp. 862–876.