Optimization Letters manuscript No.
(will be inserted by the editor)

A Deep Reinforcement Learning Framework for Solving Two-stage Stochastic Programs

Dogacan Yilmaz · İ. Esra Büyüktahtakın

Received: date / Accepted: date

Abstract In this study, we present a deep reinforcement learning framework for solving scenario-based two-stage stochastic programming problems. Stochastic programs have numerous real-time applications, such as scheduling, disaster management, and route planning, yet they are computationally challenging to solve and require specially designed solution strategies such as hand-crafted heuristics. To the extent of our knowledge, this is the first study that decomposes two-stage stochastic programs with a multi-agent structure in a deep reinforcement learning algorithmic framework to solve them faster. Specifically, we propose a general two-stage deep reinforcement learning framework that can generate high-quality solutions within a fraction of a second, in which two different learning agents sequentially learn to solve each stage of the problem. The first-stage agent is trained with the feedback of the second-stage agent using a new policy gradient formulation since the decisions are interconnected through the stages. We demonstrate our framework through a general multi-dimensional stochastic knapsack problem. The results show that solution time can be reduced up to five orders of magnitude with sufficiently good optimality gaps of around 7%. Also, a decision-making agent can be trained with a few scenarios and can solve problems with many scenarios and achieve a significant reduction in solution times. Considering the vast state and action space of the problem of interest, the results show a promising direction for generating fast solutions for stochastic online optimization problems without expert knowledge.

Dogacan Yilmaz
Department of Mechanical and Industrial Engineering, New Jersey Institute of Technology, NJ, USA
E-mail: dy234@njit.edu

İ. Esra Büyüktahtakın
Grado Department of Industrial and Systems Engineering, Virginia Tech, VA, USA
E-mail: esratoy@vt.edu

1 Introduction

The objective of this study is to develop a deep reinforcement learning methodology for solving scenario-based two-stage stochastic programs. Stochastic programming (SP), including the two-stage scenario-based stochastic programs, is a powerful way of modeling decision-making problems under uncertainty [78]. In two-stage SPs, the decisions in the system are made in two stages, and uncertainty is observed between these two stages [31]. The objective function is to minimize the first-stage and the expected second-stage costs. Two-stage SP approaches are prevalent in many real-world applications, including but not limited to disaster management [10], supply chain management [42], biofuel production [27], nurse staffing [55], water resources management [48], airline scheduling [95], and risk-averse optimization [17].

The stochastic knapsack problem is one of the most fundamental and challenging examples of the NP-Hard SPs. The multi-dimensional knapsack problem is considered to represent a very general structure among binary and integer problems. Thus, the specially-designed solutions algorithms for the two-stage stochastic knapsack can be implemented to tackle general two-stage stochastic programs. The multi-dimensional knapsack is also commonly used to demonstrate the computational performance of stochastic programming solution algorithms, as in Angulo et al. [7], Büyüktahtakın [21], and Büyüktahtakın [22]. In this study, we focus on the following two-stage multi-dimensional stochastic knapsack formulation:

$$\max_{x\in\{0,1\}^{n_1}} \quad c^\intercal x + \mathbb{E}_\xi[Q(x,\xi)] \tag{1a}$$

$$\text{s.t.} \quad Ax \leq b, \tag{1b}$$

where $x \in \{0,1\}^{n_1}$ represent the first-stage decisions, and $Q(x,\xi)$ is the optimal value for the second-stage problem. Let $\xi := (q,h,T,W)$ be a random vector with support Ξ and known probability distribution. Define the first-stage matrices c, b, and A to have sizes $n_1\times 1$, $m_1\times 1$, and $m_1\times n_1$, respectively, where the number of variables and the number of constraints in the first stage are represented by $n_1, m_1 \in \mathbb{Z}^+$, respectively. The objective of the problem is to maximize the sum of values for the selected subset of items and the expected sum of values from the second stage. In the multi-dimensional knapsack problem, the sum of sizes should be less than the capacity for each dimension. The second matrices q, h, T, and W have sizes $n_2 \times 1$, $m_2 \times 1$, $m_2 \times n_1$, and $m_2 \times n_2$, respectively. The number of variables and the number of constraints in the second stage are represented by $n_2, m_2 \in \mathbb{Z}^+$, respectively. The second-stage decisions are denoted by binary variable y:

$$\max_{y\in\{0,1\}^{n_2}} \quad q^\intercal y \tag{2a}$$

$$\text{s.t.} \quad Tx + Wy \leq h. \tag{2b}$$

Once the first-stage decisions are made, the second-stage problem would be a multi-dimensional knapsack problem. The randomness in the second stage can

be expressed with S scenarios where some or all of the second-stage matrices can be indexed by their scenarios as q^s, h^s, T^s, and W^s where $s \in \{1, \ldots, S\}$. By replicating the y-variables for each scenario s, the whole problem can be equivalently expressed as a large-scale linear binary problem:

$$\max_{x \in \{0,1\}^{n_1}, y^s \in \{0,1\}^{n_2} \forall s \in \{1,\ldots,S\}} \quad c^\intercal x + \sum_{s=1}^{S} p^s q^{s\intercal} y^s \tag{3a}$$

$$\text{s.t.} \quad Ax \leq b \tag{3b}$$

$$T^s x + W^s y^s \leq h^s \quad \forall s \in \{1, \ldots, S\}, \tag{3c}$$

where p^s represents the probability of realization for scenario s. The variations of the two-stage stochastic knapsack problems often arise in the industry with various applications, from portfolio selection [71] and telecommunications [26] to transportation planning [28]. The stochastic knapsack also has different formulations that include quadratic objectives [66], probabilistic capacity constraints [41], and risk-aversion measures [70].

Stochastic programs involving integers, such as the two-stage stochastic knapsack above, are NP-hard and require specially designed solution methodologies, such as cutting planes and decompositions with complex implementations [36, 65]. The uncertainty can be modeled in more detail with a large number of scenarios, also increasing the problem complexity. Researchers resort to developing hand-crafted exact and heuristic approaches in an attempt to reduce the computation burden. Exact solution approaches that speed up the solution include the L-shaped method [7, 59], regularized decomposition [80], dual decomposition [24], and, more recently, stochastic dual dynamic integer programming [99]. Most of the formerly developed algorithms must run from scratch for each problem without carrying information from previously solved instances, making it impractical for large-scale real-time applications. Online optimization requires solution speed for the real-time problem and solution times as low as milliseconds [13]. There is still a lack of solution approaches for online applications such as ride-sharing [37] and electric vehicle charging [93]. Online problems commonly arise in revenue management, internet advertising, and scheduling appointments in health care [53]. Some examples of real-time stochastic programs that require solutions in seconds and can benefit from faster solutions include energy demand-side management [6], inventory management [98], ambulance redeployment [74], vehicle routing [23], and material requirements planning [85]. In terms of the application of our framework, we picked a general 2-stage stochastic knapsack problem, which could be applied in numerous contexts. The 2-stage stochastic knapsack is a stochastic budget allocation problem that could be utilized in any resource allocation problem as a component. Achieving fast solutions to two-stage SPs can even be life-saving in settings like disaster and emergency management [43, 90].

Some of the recent attempts have shown the potential of deep reinforcement learning techniques to achieve near-optimal solutions to stochastic programs, such as the vehicle routing problem [75], the traveling salesperson problem

[57], and the bin packing problem [9]. However, in many safety-critical applications, including energy systems and disaster management, certain stochastic elements and realizations must be explicitly considered with scenarios to solve by the mathematical model. In this study, our goal is to develop a framework that can provide satisfactory solutions to such a complex system with the help of deep reinforcement learning. Therefore, we aim to replace the tedious process of hand-crafting heuristics with a deep reinforcement learning framework that can get very close to optimal solutions in a fraction of a second only by learning without any domain-specific information.

Reinforcement learning has long been used to tackle operations research (OR) problems and has been a major focus area in many disciplines, including OR, control theory, and game theory, with different names. Reinforcement learning is generally recognized as a subfield of machine learning where a decision-making agent learns to take actions to maximize a reward signal. The reinforcement learning environment can be described with a Markov decision process and can be modeled with a four tuple containing the set of states, set of actions, state transition function, and reward function. Dynamic programming algorithms can be utilized when there is perfect knowledge of the model. However, when the exact mathematical model is unavailable, or the dynamic programming-based solution has a large computation footprint due to the large state and action space, reinforcement learning can approximate state-value or action-value functions. Deep reinforcement learning makes use of neural networks to learn policies by trial and error. The field of deep reinforcement learning is expanded significantly in recent years and has been used for successfully solving some of the popular OR problems, including the traveling salesperson problem [11], the knapsack problem [2], the vehicle routing problem [75], and epidemic control decision making [18]. In this study, with similar motivation, we harness the power of deep reinforcement to introduce an end-to-end learning methodology to solve scenario-based two-stage stochastic problems.

Our main contribution is to present a new two-stage reinforcement learning (2SRL) framework for solving two-stage stochastic optimization problems. We propose using two different reinforcement learning agents (Agent 1 and Agent 2) for solving each stage of the problem. We train each agent sequentially. This approach prevents the complications of multi-agent reinforcement learning training, such as instability, non-stationarity, and different learning speeds [19, 20]. We propose training algorithms for Agent 1 and Agent 2 by adapting the well-known policy gradient algorithm REINFORCE [82, 92] to the case of two-stage stochastic programming. Agent 1 is trained with the feedback of Agent 2 since the decisions made in the first stage have an impact on the second-stage decisions. Further, we demonstrate the potential of our 2SRL framework for solving the two-stage stochastic knapsack problem. Our research objective is to achieve fast and satisfactory solutions, with the learned policies making solutions possible for large-scale online applications. Our 2SRL framework is general and thus can be extended to other two-stage

stochastic programs since we do not assume any specifics of the two-stage stochastic knapsack problem.

2 Literature Review

The usage of reinforcement learning for operations research problems is an active research area. In their pioneering study, Bello et al. [11] present a framework to solve the traveling salesperson problem with reinforcement learning. They use the pointer network architecture to encode the input sequence and then to decode to point to an input element, which is trained with an asynchronous advantage actor-critic algorithm. Nazari et al. [75] present a framework for solving the vehicle routing problem, which has a set of dynamic and static features. The authors propose simplifying the pointer networks by eliminating the encoder and attending over the input embeddings along with the decoder hidden state to select an input element when decoding. In a similar motivation, Hu et al. [52] present a reinforcement learning-based methodology using pointer networks to solve the three-dimensional bin packing problem and achieve a 5% improvement over heuristics.

Khalil et al. [54] explore solving combinatorial optimization problems that can be formulated as graphs with reinforcement learning. The featured framework is shown to be effective at solving minimum vertex cover, maximum cut, and traveling salesperson problems. Deudon et al. [33] attempt to solve the traveling salesperson problem with reinforcement learning, but they rely on a multi-head attention mechanism instead of recurrence. Kool et al. [57] present a reinforcement learning-based framework to learn solutions to combinatorial optimization problems. Their model includes an encoder with a multi-head attention mechanism, and they show that their framework is more effective than heuristics and comparable with specialized algorithms on the vehicle routing problem, the orienteering problem, and the prize-collecting traveling salesperson problem.

Chen and Tian [25] leverage reinforcement learning for the local search of an optimization problem. Instead of predicting the solutions directly, a solution is iteratively improved, starting from a feasible solution. Yilmaz and Büyüktahtakın [97] present an LSTM-optimization framework that uses the predicted variables partially to reduce infeasibility for solving the capacitated lot-sizing problem. Tang et al. [84] present a reinforcement learning formulation and a model for selecting cutting planes. Also, the trained agent can be used with a branch-and-cut algorithm, which is the core of commercial solvers. He et al. [49] present a two-stage framework at the intersection of reinforcement learning and operations research, where a scheduling problem is solved in two stages. First, a reinforcement learning agent reduces the solution space in its cyclical framework. Secondly, a mixed-integer process based on constructive heuristics or dynamic programming is performed.

Afshar et al. [2] suggest using a stage aggregation strategy to reduce the state space of the knapsack problem, which leads to learning faster and better

solutions. Gu et al. [47] state that some combinatorial optimization problems can be generalized to unconstrained binary quadratic programming, and they propose a framework based on pointer networks to solve them fast. Delarue et al. [32] present a framework where action selection during policy evaluation is formulated as a mixed-integer program. Bushaj and Büyüktahtakın [16] present a framework for solving the multi-dimensional knapsack problem. They form a 2-dimensional environment to reduce the action space of the agent. The agent can learn and generalize solution strategies for multi-dimensional knapsack. Yilmaz and Büyüktahtakın [96] present a learning-based framework for solving multi-period problems, including knapsack, by utilizing an attention-based encoder-decoder neural network.

Also, the solution algorithms using machine learning methodologies have gained attention for tackling two-stage stochastic programs. In a recent study, Frejinger and Larsen [38] present a framework to solve the container-railcar load planning problem, formulated as a two-stage stochastic program. They utilize a machine translation system based on supervised learning to predict a less detailed solution description instead of a fully detailed one. Larsen et al. [61] predict a less detailed solution for the same two-stage problem presented in Frejinger and Larsen [38] using multilayer perceptrons. Abbasi et al. [1] propose a framework based on supervised learning where only first-stage variables are predicted since second-stage variables are not implemented in practice. Wu et al. [94] aim to solve two-stage stochastic optimization problems that can be expressed as graphs to reduce the number of scenarios and estimate the recourse cost. Crespo-Vazquez et al. [29] attempt to solve a two-stage stochastic problem using clustering and utilizing recurrent neural networks to generate probabilities for scenarios. Bengio et al. [12] propose using machine learning to generate a representative scenario for the problem so that it can be solved with an off-the-shelf solver in a fast setting.

Traditional solution approaches for solving two-stage stochastic problems include the L-shaped method [7, 59], dual decomposition [24, 67], branch-and-bound [3], progressive hedging [40, 79], and Gomory cuts [39]. More recently, stochastic dual dynamic integer programming (SDDiP) was proposed by Zou et al. [99] to solve two-stage and multi-stage stochastic programs with integers and has shown to be applicable to a common range of problems. For a detailed discussion of two-stage stochastic programs and general solution approaches, we refer to Birge and Louveaux [14].

There is growing literature on using deep reinforcement learning methodologies to help solve various operations research problems in the last few years. Despite all the advancements, there is still a research gap in integrating deep reinforcement learning with mathematical programming to tackle stochastic programming programs. Scenario-based two-stage stochastic programs are used in numerous fields for decision-making under uncertainty. They can benefit from a reduced solution time to be used in real-time applications without the need for specially designed solution approaches, which require expert knowledge and can be time-consuming.

2.1 Key Contributions of the Study

Our objective with this study is to present a general reinforcement learning-based framework to generate high-quality solutions to two-stage SPs as quickly as a fraction of a second without the need for developing a special solution methodology. Our study is motivated by the wide applicability of two-stage scenario-based SPs and the promising results of reinforcement learning for combinatorial optimization. Our contributions are as follows:

- To our knowledge, this is the first study that utilizes deep reinforcement learning to solve two-stage scenario-based stochastic programs with a multi-agent structure. Specifically, we propose using two different agents (Agents 1 and 2) to solve each stage of the problem (stages 1 and 2).
- In our 2SRL framework, we present a strategy to generate realistic second-stage problems without the need for optimal first-stage decisions.
- We present our detailed training algorithm together with our scenario sampling approach during training to reduce the correlations and ease the training, similar to experience replay. First, Agent 2 is trained to solve second-stage subproblems given any first-stage decision. Then Agent 1 is trained with the feedback of Agent 2 since the decisions made in the first stage have an impact on both stages of the problem. For this purpose, we propose a novel policy gradient calculation for the well-known REINFORCE algorithm.
- We present the quality of our 2SRL framework by presenting the time improvement factor and optimality gap compared to a commercial solver, a state-of-the-art solution methodology (SDDiP), and two heuristic approaches.
- We investigate the opportunity of training agents using problems with a few scenarios and items to predict much larger instances with a higher number of scenarios and items. The results show that the trained agents can be used to generalize the results to a larger set of scenarios, highlighting our approach's computational impact.

The remainder of the study is as follows. Section 3 introduces the training details of the 2SRL framework along with information on pointer networks. Section 4 summarizes the details of implementation and experimentation. Section 5 presents the computational results. Section 6 concludes the study with a discussion.

3 Two-stage Reinforcement Learning (2SRL) Framework

Here, we present the details of the 2SRL framework and justify the need for two different agents for two different stages. First, the details of pointer networks are presented in Section 3.1. Then, in Section 3.2, we explain each agent's communication and information exchange during training by giving a detailed algorithm based on REINFORCE [92].

3.1 Pointer Networks

As proposed by Vinyals et al. [89], pointer networks have been essential for many different tasks, including text summarization [81], intelligent code completion [62], and airline itinerary prediction [72]. Based on attention-based encoder-decoder sequence-to-sequence learning architecture, the pointer network utilizes a pointer mechanism to select an input element at the time of decoding, which is required for various combinatorial optimization problems. Pointer networks and their modifications have been trained with reinforcement learning paradigm to successfully solve several types of operations research problems, including traveling salesperson [11, 30, 63], vehicle routing [15, 64, 75], max-cut [46], bin packing [35, 52], and knapsack [11, 45]. Further studies have utilized transformers [88] based on multi-head attention to solve optimization problems [57] and showed improvements over pointer networks for several problems. In our study, we opted to use pointer networks instead of a transformer architecture since the latter approach results in an optimality gap improvement of only 0.02% for large knapsack problems over the former architecture [58], and our motivating study [11] introduces the use of pointer networks with reinforcement learning.

The pointer networks consist of four main components as encoder, decoder, glimpse, and pointer mechanism. An encoder is a recurrent neural network that processes the input sequence. The encoder aims to generate a high-dimensional representation of input elements that ideally captures sequential relations and hidden features. Similar to the encoder, the decoder is also a recurrent neural network, but it selects a subset of input items in a sequential manner. A pointer is a mechanism that chooses an element from the input sequence by making a comparison between encoder and decoder hidden states. Glimpse is a context-based attention mechanism employed before the pointer mechanism is implemented to gain more knowledge of the input sequence.

Attention-based encoder-decoder models were originally developed for neural machine translation tasks [8, 68], where a fixed-sized vocabulary of words is used for training the models. In such a setting, the vocabulary for the neural machine translation systems must be determined before training. On the other hand, many combinatorial optimization problems present a different paradigm than neural machine translation by making predictions of which elements from the input should be selected for optimal decisions. In the knapsack problem, a subset of input items is selected by the decoder. Even though it would be possible to have a fixed-sized output for such problems, it can be impractical to use one since adding or removing one input element would require retraining the model or other special solution paradigm. Pointer networks eliminate this requirement by not having a fixed-sized prediction dictionary. Instead, they enable predicting by selecting a subset of input elements. Also, a feasibility mask ensures the selection of only feasible items.

Our neural architecture builds upon the network presented by Bello et al. [11]. However, we utilize bidirectional LSTM [44], an extension of the unidirectional LSTM [51], to better capture the input temporal characteristics.

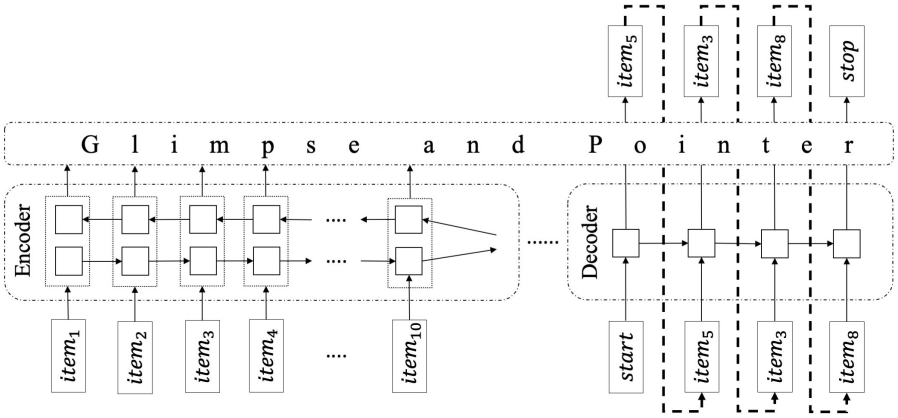


Fig. 1: An example pointer network architecture.

Figure 1 presents an overview of the pointer network architecture utilized. In this small example, the encoder processes the problem parameters for a 10-item knapsack problem with bidirectional LSTM layers. Then the decoder iteratively selects a subset containing items 5, 3, and 8 for the knapsack by utilizing the glimpse and pointer mechanism. We refer to Bello et al. [11] for further details on the pointer networks.

The input vector for each item j has size $m_1 + 1$ and is calculated as $[c_j, A_j/b]$ for Agent 1's problem from Equation (1), where c_j and A_j represent the problem parameters for item j with sizes 1 and m_1, respectively. The input vector for item k in Agent 2's problem has size $m_2 + 1$. It is calculated as $[q_k, W_k/(h - T\bar{x})]$ from Equation (2), where q_k and W_k represent the problem parameters for item k with sizes 1 and m_2, respectively, and $\bar{x}$ is the solution to the first-stage problem. Since the problem that we are trying to solve is a multi-dimensional knapsack problem, this input structure allows us to normalize each constraint with respect to the existing or remaining capacity.

In our learning paradigm, we resort to an algorithm known as actor-critic training to be described in detail later in Section 3.2. In this paradigm, the actor network learns a policy to maximize the objective function, and the critic learns the expected objective function value given a sample problem. While both networks contain encoders, the critic does not have a recurrent neural network decoder.

3.2 Training Paradigm for 2SRL Framework

Here, we present a novel 2SRL training strategy for solving two-stage scenario-based stochastic optimization problems. In our methodology, Agent 1 is the decision-making reinforcement learning agent for the first-stage problem, and Agent 2 decides on the values of decision variables for the second-stage problem. The environment can be defined as the agent's interaction point with the

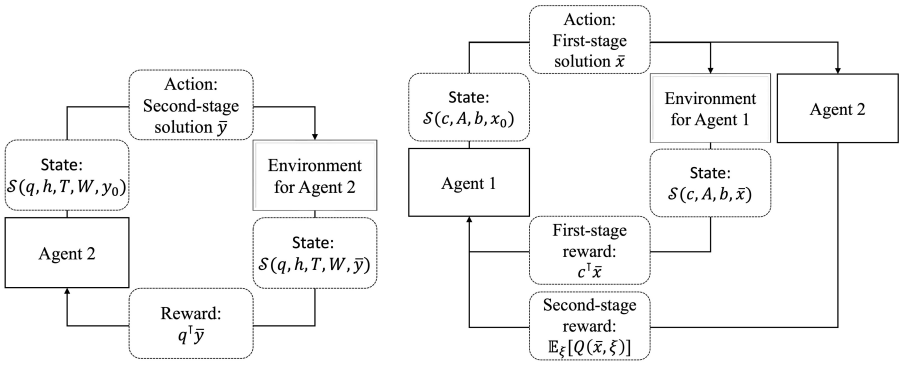
problem. Decision-maker learns based on an action-reward cycle through an environment defined by the problem characteristics. In the context of stochastic programming, those characteristics are the problem's input parameters, the decision variables, and the constraints.

In many two-stage stochastic problems, the set of decisions that need to be made in the first stage can be highly different from second-stage decisions. Such examples include the cases where each of the two stages differs in their input data's underlying distributions, the number of constraints, and variable types, such as continuous, binary, or integer. Therefore, using a single agent might not be able to handle variability in the types of decisions. Even in cases where similar decisions are taken in both stages, the structures of the input data elements can be different. Thus, we use two agents that make decisions about their respective stages instead of having a single agent. Also, by using different agents for two stages, agents work with a smaller action space compared to a problem with a single agent. In addition, this approach for training Agent 1 presents a paradigm similar to the classical solution methodologies, such as the L-shaped method [7, 87]. We are inspired by the state-of-the-art decomposition methods for solving two-stage stochastic optimization problems in the operations research literature. In general, such decomposition approaches utilize a master-subproblem methodology to approximate the second-stage objective function by solving the second-stage subproblems and using them to generate first-stage decision variables through the master problem. Decomposition approaches are considered highly efficient and specialized for solving large and challenging stochastic programs and provide two main advantages. First, they replace the computationally complex and large-scale stochastic problem with a group of subproblems that are easier to solve. Secondly, second-stage subproblems can be solved using standard solvers. We are encouraged by the success of such a decomposition framework and utilize two agents to solve each stage of the problem rather than training a single agent.

Agent 2 is trained before Agent 1. Our aim here is to provide feedback to Agent 1 on the quality of decisions during its training since the first-stage decisions must be determined based on their impact on the second-stage decisions as well as the objective function value. Additionally, stability can be challenging when two agents are trained and act in the same environment, especially when they are competing [20]. Even though both agents try to maximize a reward in the form of the objective function, the decisions made in both stages might need to be different to maximize their reward in their respective stages. Therefore, each agent might be working towards a different set of decision-making strategies, and this might result in competition between them. To eliminate that possibility, we train both agents sequentially. Furthermore, the non-stationarity of multi-agent environments is a challenge in multi-agent reinforcement learning since each agent is learning simultaneously. Each agent tries to solve a moving-target problem meaning that the optimal policy for an agent changes as the other agents' optimal policy changes [19, 76]. In our framework, we handle this prospect by learning sequentially. Also, the varying

learning speed of different agents increases the need for hyperparameter tuning and coordination between agents [20].

Hierarchical reinforcement learning is a growing field of study that shows similarities with the training paradigm of 2SRL. In hierarchical reinforcement learning, a task is divided into subtasks by a higher-level policy to decompose the long-horizon problem [77]. The subtasks can be learned through a reinforcement learning agent as well. Hierarchical reinforcement learning provides advantages by reducing the subtask complexity, increasing reusability, reducing network parameters, and handling the curse of dimensionality [5]. However, they would often require task-specific design [73], which can hinder the generalization properties of the solution. Also, more research is needed to understand when hierarchical reinforcement learning can provide recognizable benefits [77]. Moreover, the approach of hierarchical reinforcement learning has only been studied recently for solving operations research problems [69]. Therefore, we propose the 2SRL framework without a hierarchical reinforcement learning perspective to achieve generalization by exploiting widely utilized and tested training paradigms based on policy gradient methods.



(a) Agent 2 training overview. (b) Agent 1 training overview.

Fig. 2: 2SRL training overview.

Figure 2 presents the training overview for Agent 1 and Agent 2 by highlighting their respective states, actions, and rewards. The high-level training procedure for Agent 2 with the 2SRL paradigm is presented in Figure 2a. Agent 2 solves the second-stage problem given a particular realization of the uncertain data in the form of a scenario. The state space consists of the data for that second-stage scenario problem, i.e., the second-stage matrices q, h, T, and W from Equation (2) and the second-stage decisions initialized as all zeros, y_0. The initial state space is denoted as $\mathcal{S}(q, h, T, W, y_0)$. The agent generates a solution $\bar{y}$ where the action space represents all second-stage variables. Then, Agent 2 transitions into the new state $\mathcal{S}(q, h, T, W, \bar{y})$. The reward that the agent observes is a measure of the solution quality, i.e.,

the objective function value $q^\intercal \bar{y}$, where q is the objective function coefficient. Agent 2 can be described as a Markov Decision Process with a four tuple $\langle \mathcal{S}(q,h,T,W,y), y, q^\intercal \bar{y}, \mathbb{P}(\mathcal{S}(q,h,T,W,\bar{y}) \mid \mathcal{S}(q,h,T,W,y_0)) = 1 \rangle$ having set of states, actions, rewards, and transition probability function, respectively. Note that the transition from initial state $\mathcal{S}(q,h,T,W,y_0)$ to end state $\mathcal{S}(q,h,T,W,\bar{y})$ is deterministic once Agent 2 generates the solution $\bar{y}$. This is because the item selection for the whole instance takes place at once within the pointer network rather than a step-by-step cycle that takes place in conventional Markov decision processes (MDPs).

The training of Agent 1 is presented in Figure 2b. There is a major difference in training between Agents 1 and 2. While Agent 2 only gets feedback from its environment where Agent 1 decision variables are fixed, Agent 1 gets feedback from both Agent 2 and its environment. This is for the fact that the objective function in the first stage involves the minimization of both the first-stage and expected second-stage costs. The state space for Agent 1 consists of first-stage parameters of the problem, i.e., the matrices c, A, and b given in Equation (1) and the first-stage decision initialized as all zeros x_0. We denote this initial state space as $\mathcal{S}(c,A,b,x_0)$. For Agent 1, the action space represents the first-stage decision variables. Agent 1 samples a complete first-stage solution $\bar{x}$ as action and transitions into a new state $\mathcal{S}(c,A,b,\bar{x})$. Agent 1 gets a reward for the first stage as $c^\intercal \bar{x}$ and the second stage as $\mathbb{E}_\xi[Q(\bar{x},\xi)]$ from the trained Agent 2 model, where ξ denotes the uncertainty. The whole process can be expressed as a Markov Decision Process with a four tuple $\langle \mathcal{S}(c,A,b,x), x, c^\intercal \bar{x} + \mathbb{E}_\xi[Q(\bar{x},\xi)], \mathbb{P}(\mathcal{S}(c,A,b,\bar{x}) \mid \mathcal{S}(c,A,b,x_0)) = 1 \rangle$ containing a set of states, actions, rewards, and transition probability function, respectively. Here, note that the initial state is $\mathcal{S}(c,A,b,x_0)$ and the transition to end state $\mathcal{S}(c,A,b,\bar{x})$ happens deterministically after the complete first-stage solution $\bar{x}$ is sampled rather than a step-by-step cycle where the transition is probabilistic.

3.2.1 Agent 2 and Simulating Subproblems

Our aim here is to train Agent 2, which can solve second-stage scenario problems given a set of second-stage problem parameters. However, this raises a complication for Agent 2. Problem (2) includes the first-stage decision variables in constraints (2b), but their optimal values are unknown during training. We propose a strategy to generate realistic second-stage problems by reducing the right-hand sides of constraints (2b) by the amount of capacity used by the first-stage decisions. To represent this, a set of first-stage coefficients $\mathcal{F}$ is calculated and saved before the training starts. The aim is to simulate first-stage decisions similar to true first-stage decisions in the second-stage problem without having to solve them throughout the training. To calculate $\mathcal{F}$, first, a small fixed number of randomly generated two-stage problems given in (3) are solved to optimality. Then, using the optimal first-stage decisions x^*, we calculate the coefficient $\mathsf{f}^s = \frac{T^s x^*}{h^s} \in [0,1]$, where f^s represents the fraction of the capacity used in the first-stage problem in Equation (3c) for

all $s \in \{1, \ldots, S\}$. Those calculated $\mathtt{f}^s$ values are saved without noting scenario superscript s to constitute $\mathcal{F}$. During training, we sample a coefficient f^i from $\mathcal{F}$ as independent and identically distributed for each training instance i and modify the right-hand side h^s in Equation (3c) to simulate the remaining capacity as $h^s(1 - \frac{T^s x}{h^s})$. With this approach, remaining capacities that are simulated would be changing at each iteration based on sample $\mathcal{F}$. The sample $\mathcal{F}$ is generated by solving a substantially low number of instances compared to the number of training instances to reduce computational complexity. Consequently, remaining capacity can overestimate or underestimate the actual remaining capacity calculated if the optimal first-stage decision has been known for all training instances. However, the critical point of our approach is that Agent 2 would not be trained using unrelated, significantly different, or extremely shifted distributions; rather, Agent 2 is trained with a wider distribution of problems compared to the distribution created by solving training instances. In addition, it would be very unpractical and computationally expensive to solve all training instances to extract the second stage problem for Agent 2, which would defeat the purpose of using a reinforcement learning approach.

Algorithm 1 REINFORCE for Agent 2

Input: Batch size B, number of epochs E, steps per epoch T, number of scenarios S, a set of first-stage coefficients $\mathcal{F}$
Output: Trained actor network θ_2^A, trained critic network θ_2^C
Procedure: Training Agent 2

Initialize actor network parameters θ_2^A
Initialize critic network parameters θ_2^C
for epochs = 1 to E **do**
  for steps = 1 to T **do**
    $KP^i \leftarrow SampleProblem()\ \forall i \in \{1, \ldots, B\}$
    $f^i \leftarrow SampleCoefficient(\mathcal{F})\ \forall i \in \{1, \ldots, B\}$
    $KP_2^{i,s} \leftarrow ExtractSecondStageProblem(KP^i, f^i)\ \forall i \in \{1, \ldots, B\}, \forall s \in \{1, \ldots, S\}$
    $KP_2^{i,s'} \leftarrow SampleScenario(KP_2^{i,s})\ \forall i \in \{1, \ldots, B\}$
    $y^i \leftarrow SampleSolution(p_{\theta_2^A}(. \mid KP_2^{i,s'}))\ \forall i \in \{1, \ldots, B\}$
    Update the actor network:
    $\tilde{z}_2^i \leftarrow \theta_2^C(KP_2^{i,s'})\ \forall i \in \{1, \ldots, B\}$
    $g_{\theta_2^A} \leftarrow \frac{1}{B} \sum_{i=1}^{B} (z(y^i \mid KP_2^i) - \tilde{z}_2^i) \nabla_{\theta_2^A} \log p_{\theta_2^A}(y^i \mid KP_2^{i,s'})$
    $\theta_2^A \leftarrow ADAM(\theta_2^A, g_{\theta_2^A})$
    Update the critic network:
    $\mathcal{L}_2^C \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|\tilde{z}_2^i - z(y^i \mid KP_2^{i,s'})\|_2^2$
    $\theta_2^C \leftarrow ADAM(\theta_2^C, \nabla_{\theta_2^C} \mathcal{L}_2^C)$
  end for
end for

Input parameters to Algorithm 1 are the batch size B, number of epochs E, steps per epoch T, number of scenarios S, and a set of first-stage coefficients $\mathcal{F}$. Algorithm 1 starts with the initialization of actor and critic networks, θ_2^A and

θ_2^C, respectively, in steps 1 and 2. The actor network θ_2^A is a pointer network that is used to make decisions for the second-stage scenario problems. Critic network θ_2^C is used to estimate the expected objective function coefficient given the second-stage problem. Training iterations are repeated for each epoch and each step with a loop in steps 3 and 4. At step 5, a batch of training data containing both first and second-stage parameters is sampled. Here, each knapsack instance i is denoted with KP^i $\forall i \in \{1, \ldots, B\}$, which is given in Equation (3) with added superscript i that denotes instance i within a batch of problems. Then at step 6, a random first-stage coefficient $f^i \in \mathcal{F}$ for all $i \in \{1, \ldots, B\}$ is sampled. This step is taken to generate realistic second-stage problems and therefore, to help with the quality of trained models. At step 7, the second-stage problem $KP_2^{i,s}$ for each scenario $s \in \{1, \ldots, S\}$ is calculated for each training instance $i \in \{1, \ldots, B\}$ using the selected first-stage coefficient f^i:

$$\max_{y^{i,s} \in \{0,1\}^{n_2}} \quad q^{i,s\intercal} y^{i,s} \tag{4a}$$

$$\text{s.t.} \quad W^{i,s} y^{i,s} \leq h^{i,s}(1 - f^i). \tag{4b}$$

In this step, we aim to extract the second-stage problem to be used in the training of Agent 2. In step 8, we sample a single scenario s' from all available scenarios $\forall s \in \{1, \ldots, S\}$ in the batch $\forall i \in \{1, \ldots, B\}$. This step is taken to reduce the correlation within the batch resulting from the same first-stage decisions, especially when not all second-stage matrices are stochastic. If some of those matrices q^s, h^s, T^s, and W^s are not dependent on scenarios, the learning efficiency would reduce due to correlated samples. By sampling a single scenario for each instance, we found out that the training efficiency is increased. The second-stage solution y^i is sampled using the actor model θ_2^A and the extracted second-stage data $KP_2^{i,s'}$ within step 9. Here, the selection of second-stage items y^i is made based on the stochastic policy $p_{\theta_2^A}$. In the next step, a baseline for the expected objective function value $\tilde{z}_2^i$ is estimated using the critic network θ_2^C, which helps reduce the policy gradient variance. In step 11, the gradients of the actor network θ_2^A are calculated using the well-known policy gradient method REINFORCE. Here operator $z(y^i \mid KP_2^{i,s'})$ calculates the reward for action y^i given second-stage problem parameters $KP_2^{i,s'}$ as $q^{i,s'\intercal} y^i$. In the next step, the parameters of the actor network are updated based on the gradients calculated in the previous step using the stochastic gradient update method Adam [56]. With step 13, the mean squared error loss for the critic network θ_2^C is calculated by squaring the difference between the objective function value estimated by the critic network $\tilde{z}_2^i$ and the objective function value using the prediction made by the actor network $z(y^i \mid KP_2^{i,s'})$. In the next step, the parameters of the critic network are updated using the loss calculated in step 13 with the Adam optimizer. These steps are repeated for all epochs and steps.

3.2.2 Training Agent 1

In this subsection, we present the detailed trained algorithm for Agent 1 to solve the first-stage problem after Agent 2 is trained to solve second-stage problems. Algorithm 2 presents the details of training Agent 1 using the policy gradient algorithm based on REINFORCE. During the training, Agent 1 gets a reward based on the quality of the decisions both from its environment and Agent 2. This is one of the most important features of our 2SRL framework for solving two-stage stochastic optimization problems. In general, the objective function of two-stage stochastic optimization problems can be expressed by Equation (1a). Here, the optimal decisions for the first stage are found considering both the first-stage and expected second-stage implications. By utilizing the feedback from the second-stage agent, we aim to ensure that the first-stage decision-maker is aware of the reward resulting from both stages of the problem. Similar to Algorithm 1; batch size B, number of epochs E, steps per epoch T, and the number of scenarios S are inputs of Algorithm 2. Additionally, trained actor $\theta_2^{A^*}$ and critic networks $\theta_2^{C^*}$ of stage 2 are taken as input. The algorithm performs a training iteration to output the trained actor θ_1^A and critic networks θ_1^C of stage 1.

Algorithm 2 REINFORCE for Agent 1

Input: Batch size B, number of epochs E, steps per epoch T, number of scenarios S, trained actor network $\theta_2^{A^*}$, trained critic network $\theta_2^{C^*}$
Output: Trained actor network θ_1^A, trained critic network θ_1^C
Procedure: Training Agent 1

Initialize actor network parameters θ_1^A
Initialize critic network parameters θ_1^C
for epochs = 1 to E **do**
  for steps = 1 to T **do**
    $KP^i \leftarrow SampleProblem()\ \forall i \in \{1,\dots,B\}$
    $KP_1^i \leftarrow CalculateFirstStageProblem(KP^i)\ \forall i \in \{1,\dots,B\}$
    $x^i \leftarrow SampleSolution(p_{\theta_1^A}(.\mid KP_1^i))\ \forall i \in \{1,\dots,B\}$
    $\tilde{z}_1^i \leftarrow \theta_1^C(KP_1^i)\ \forall i \in \{1,\dots,B\}$
    $KP_2^{i,s} \leftarrow ExtractSecondStageProblem(KP^i,x^i)\ \forall i \in \{1,\dots,B\}, \forall s \in \{1,\dots,S\}$
    $y^{i,s} \leftarrow SampleSolution(p_{\theta_2^{A^*}}(.\mid KP_2^{i,s}))\ \forall i \in \{1,\dots,B\},\ \forall s \in \{1,\dots,S\}$
    Update the actor network:
    $z_2^i \leftarrow \frac{1}{S}\sum_{i=1}^{S} z(y^{i,s}\mid KP_2^{i,s})\ \forall i \in \{1,\dots,B\}$
    $\tilde{z}_2^i \leftarrow \frac{1}{S}\sum_{i=1}^{S} \theta_2^{C^*}(KP_2^{i,s})\ \forall i \in \{1,\dots,B\}$
    $g_{\theta_1^A} \leftarrow \frac{1}{B}\sum_{i=1}^{B}(z(x^i\mid KP_1^i) + z_2^i - \tilde{z}_1^i - \tilde{z}_2^i)\nabla_{\theta_1^A}\log p_{\theta_1^A}(x^i\mid KP_1^i)$
    $\theta_1^A \leftarrow ADAM(\theta_1^A, g_{\theta_1^A})$
    Update the critic network:
    $\mathcal{L}_1^C \leftarrow \frac{1}{B}\sum_{i=1}^{B}\|\tilde{z}_1^i - z(x^i\mid KP_1^i)\|_2^2$
    $\theta_1^C \leftarrow ADAM(\theta_1^C, \nabla_{\theta_1^C}\mathcal{L}_1^C)$
  end for
end for

Algorithm 2 starts with the initialization of the actor network θ_1^A and trained critic network θ_1^C. In steps 3 and 4, the training loop is continued for a predetermined number of epochs and steps. In step 5, a batch of two-stage knapsack problems is sampled randomly from the training set. We denote each knapsack instance i as KP^i $\forall i \in \{1, \ldots, B\}$, which is given in Equation (3) with added instance superscript i. In step 6, the first-stage problems are obtained for each problem in the batch. The first-stage problem KP_1^i for all $i \in \{1, \ldots, B\}$ can be expressed as:

$$\max_{x^i \in \{0,1\}^{n_1}} \quad c^{i\intercal} x^i \tag{5a}$$

$$\text{s.t.} \quad A^i x^i \leq b^i, \tag{5b}$$

where x^i is a sampled solution $\forall i \in \{1, \ldots, B\}$. Here, the two-stage problem is isolated from the second-stage problem. In step 7, a first-stage solution x^i is generated for the KP_1^i $\forall i \in \{1, \ldots, B\}$ using the first-stage actor network θ_1^A. In step 8, an estimate of the first-stage objective function value $\tilde{z}_1^i$ is made using the first-stage critic network θ_1^C. This predicted baseline is later used in step 13 to make a gradient update on the actor network θ_1^A based on the policy gradient theorem. In step 9, the second-stage problem is isolated from the two-stage problem, similar to step 7 of Algorithm 1. The second-stage scenario subproblem $KP_2^{i,s}$ is expressed given first-stage decision x^i for instances $i \in \{1, \ldots, B\}$ in the formulation below:

$$\max_{y^{i,s} \in \{0,1\}^{n_2}} \quad q^{i,s\intercal} y^{i,s} \tag{6a}$$

$$\text{s.t.} \quad W^{i,s} y^{i,s} \leq h^{i,s} - T^{i,s} x^i \tag{6b}$$

Here, we do not sample scenarios, unlike Algorithm 1, since no training iteration is performed for Agent 2. In step 10, second-stage decision $y^{i,s}$ for each scenario $\forall s \in \{1, \ldots, S\}$ is generated for each problem $\forall i \in \{1, \ldots, B\}$ given in the above formulation (6) using the actor network $\theta_2^{A^*}$ of Agent 2 from Algorithm 1. This is achieved by stochastic policy $p_{\theta_2^{A^*}}$, trained in Algorithm 1, given the second-stage scenario subproblem $KP_2^{i,s}$ $\forall i \in \{1, \ldots, B\}$, $\forall s \in \{1, \ldots, S\}$. Then in step 11, the expected second-stage cost z_2^i is calculated using the predicted second-stage decision $y^{i,s}$, where $z(y^{i,s} \mid KP_2^{i,s}) = q^{i,s\intercal} y^{i,s}$. To generate a realistic estimate of the baseline for the second-stage problem $\tilde{z}_2^i$, the second-stage objective function value is predicted using the second-stage critic network $\theta_2^{C^*}$ within step 12. Then, in step 13, the gradients are calculated based on a modified version of a policy gradient algorithm by integrating the second-stage expected objective function value and predicted second-stage baseline, where $z(x^i \mid KP_1^i) = c^{i\intercal} x^i$. Here, gradients include feedback on the decision quality for the second stage. In step 14, a gradient update is made to the first-stage actor θ_1^A using the Adam optimizer. In step 15, first-stage critic loss $\mathcal{L}_1^C$ is calculated as the mean squared error between the first-stage baseline $\tilde{z}_1^i$ predicted by first-stage critic θ_1^C and the actual first-stage objective function value calculated by using the variables predicted by

first-stage actor θ_1^A. In the next step, a gradient update with Adam is made to the first-stage critic network θ_1^C by using the loss calculated in the previous step. This training iteration is continued for all steps and epochs.

4 Implementation and Experimentation Details

In this section, we present the details of our implementation, experimentation, and evaluation. Our computational environment is a high-performance computing cluster running Linux 3.10.0 with Intel Xeon Gold 6226R 2.90 GHz, 96 GB of memory, and NVIDIA Tesla T4 GPU. To create baseline solution times for two-stage stochastic knapsack problems, we opted to use Gurobi 9.5 instead of CPLEX 20.1.0 since Gurobi performed faster when solving various instances in our preliminary results. All codes are written in Python 3.8.5. The deep learning models are trained using PyTorch 1.7.1.

4.1 Generating Two-stage Stochastic Knapsack Problems

To sample two-stage stochastic knapsack problems with scenarios, we employ a scheme similar to the one presented by Angulo et al. [7]. In their study, authors generate two-stage stochastic multiple binary knapsack problems to evaluate their methodology. The parameters of instances are sampled from uniform integer distributions between u and v, denoted by $U[u, v]$. The mean of matrices A, T, and W is denoted by $\overline{A}$, $\overline{T}$, and $\overline{W}$, respectively. The elements of the first-stage matrices c and A are sampled from $U[1, 20]$. The right-hand side parameter b is sampled from $U\left[0.4 \times \overline{A} \times n_1, 0.6 \times \overline{A} \times n_1\right]$. The elements of the second-stage matrices q, T, and W are sampled from $U[1, 20]$, and h is sampled from $U\left[0.4 \times (\overline{T} \times n_1 + \overline{W} \times n_2), 0.6 \times (\overline{T} \times n_1 + \overline{W} \times n_2)\right]$. For training, the number of items for the first stage and the number of items for the second stage are $n_1, n_2 \in \{10, 20\}$. Also, the number of resource constraints for the first stage and the number of resource constraints for the second stage are $m_1, m_2 \in \{5, 10\}$. Finally, we consider problems with 10 scenarios during training for ease of computation. For testing, we generate instances with an increasing number of items and scenarios. Table 1 presents the results with the number of items $n_1, n_2 \in \{10, 15, 20\}$ and $m_1, m_2 = 5$. Table 2 presents the results with the number of items $n_1, n_2 \in \{20, 30, 40\}$ and $m_1, m_2 = 10$. For all results in Tables 1 and 2, instances with the number of scenarios $s \in \{10, 50, 100, 500, 1000\}$ are solved.

In a recent study, SDDiP is suggested by Zou et al. [99] to solve scenario-based stochastic problems involving integers. The SDDiP is considered to be a state-of-the-art solution methodology and achieved significant improvements in solution times for a wide range of problems, including hydropower scheduling [50], power infrastructure planning [60], and lot-sizing [86]. Thus, we utilize the SDDiP approach as a benchmark solution method in our experiments for

comparison to the 2SRL. We use Python implementation of the SDDiP solution algorithm developed by Ding et al. [34] together with Gurobi. We limit the SDDiP solution time to 2-hour or 20 stable iterations, whichever comes first. We also utilize a heuristic to solve the knapsack problem presented. We utilize the greedy primal effective capacity heuristic (PECH) designed by Akçay et al. [4] for the multi-dimensional knapsack problems. While the heuristic is not specifically designed to solve a two-stage problem, it delivers high-quality solutions. Also, we generate a random feasible solution similar to Bello et al. [11] for comparison purposes.

4.2 Model Architecture

As explained in Section 3.1, the pointer network for both agents contains four main components: encoder, decoder, glimpse, and pointer mechanism. The actor network for both Agent 1 and Agent 2 consists of 2 bidirectional LSTM layers with 256 hidden units in the encoder and 2 unidirectional LSTM layers with 512 hidden units in the decoder. We utilized a single glimpse calculation before the selection using the pointer mechanism. The critic network for Agent 1 and Agent 2 consists of 2 bidirectional LSTM layers with 64 hidden units in the encoder, 2 layered neural networks with 128 units, and a ReLU activation function. Also, we utilize a dropout technique with a rate of 0.3 to achieve a better generalization performance [83].

Similar to Bello et al. [11], we make use of a softmax temperature with a temperature hyperparameter of 1.5. Also, the logit clipping approach is taken, which is found helpful by Bello et al. [11] for performance gains. The training set consists of 10,000 two-stage stochastic optimization problems. Since we utilize a reinforcement learning-based approach and not supervised learning, the problems in the training set do not need to be solved before training. However, we have solved and recorded the solutions of 50 instances to generate a set of first-stage coefficients $\mathcal{F}$ and use it as an input to train Agent 2 in Algorithm 1. Then, the training set is created by sampling from distributions defined in the previous section. We also utilize a sampling approach during testing, which samples multiple solutions from a stochastic policy. This approach can yield a significant improvement over the greedy decoding approach at the cost of a very small increase in computational time. In this approach, we do not perform any training iteration but rather just sample solutions from a multinomial distribution with probabilities generated by the trained network.

4.3 Evaluation Methodology

Here, we describe the metrics used to measure the success of our 2SRL framework to solve the two-stage scenario-based stochastic knapsack problem. We evaluate our methodology using optimality gap and solution time reduction with respect to the Gurobi solver, SDDiP, heuristic, and random solution.

- **timeGRB**: Average solution time of test instances using Gurobi with a 2-hour solution time limit.
- **time2SRL**: Average solution time of test instances using 2SRL framework.
- **timeSDDiP**: Average solution time of test instances using SDDiP [34] with a 2-hour solution time limit or until it reaches 20 stable iterations.
- **timePECH**: Average solution time of test instances using the PECH of Akçay et al. [4].
- **timeRand**: Average solution time of test instances with a randomly generated feasible solution.

Further, we calculate the following metrics to compare the 2SRL framework with other approaches:

Definition 1 Let the average objective function value for the Gurobi solution be objGRB and for the 2SRL solution be obj2SRL. The optimality gap **optGap2SRL** between the base solution value of Gurobi and the solution of 2SRL framework (SDDiP solution for **optGapSDDiP**, PECH heuristic solution for **optGapPECH**, and random solution for **optGapRand**) is given by:

$$\textbf{optGap2SRL}(\%) = \frac{\mid obj2SRL - objGRB \mid}{objGRB} \times 100. \tag{7}$$

Note that **optGapGRB** denotes the optimality gap returned by Gurobi within the 2-hour solution time limit.

Definition 2 The solution time improvement factor **timeImp2SRL** resulting from the 2SRL framework (**timeImpSDDiP** from using the SDDiP, **timeImpPECH** from using the PECH heuristic, and **timeImpRand** from using a random solution) is defined as:

$$\textbf{timeImp} = \frac{timeGRB}{time2SRL}. \tag{8}$$

Definition 3 A one-sided Wilcoxon signed-rank test [91] is carried out to calculate the **p-value**, which is a statistical test that measures if the pairwise differences between two solution times are symmetric around 0. The null and alternative hypotheses are:

$$H_0 : median(timePECH - time2SRL) < 0 \tag{9a}$$

$$H_1 : median(timePECH - time2SRL) > 0 \tag{9b}$$

We reject the null hypothesis H_0 if the p-value is less than 0.01 and conclude that the 2SRL framework performs statistically faster than the PECH heuristic.

5 Computational Results

This section presents the computational results for the 2SRL, along with a comparison with Gurobi, SDDiP, the PECH heuristic, and a random solution approach. For all instances in the tables, solution time is given in seconds and rounded down to zero if they are less than 0.05 seconds. The number of scenarios is denoted by #sc, and the number of items in the test set is denoted by #items. Test sets contain 100 total instances, and average results over 20 instances are presented for each specific case in the first five result columns of Tables 1 and 2. Moreover, we present the average as Avg, median as Mdn, and standard deviation as Std for each table over 100 test instances by calculating the metrics independently for each instance.

Table 1 presents the results for the 2SRL trained with instances having 10 scenarios and 10 items in both the first and second stages. The first dataset given in the second column of Table 1 has the same number of items and scenarios as the training set. For this set, 2SRL provides an instant solution with a gap of 7.29%. The next set of test instances contains 50 scenarios and 20 items. In this case, the optimality gap increases slightly to 10.23%. Again, the 2SRL provides an instant solution and reduces the solution time by five orders of magnitude compared to Gurobi. The remaining three sets of instances in Table 1 have 100, 500, and 1,000 scenarios, respectively. Their optimality gaps fall between the range of 6% to 7%. While this might be adequate for some applications, PECH heuristic outperforms 2SRL in terms of optimality gap, but 2SRL dominates the heuristic in terms of solution time, as can be seen from the time improvement factors. On average, 2SRL reduces the solution time by more than a factor of 100,000 when compared to Gurobi, with an average optimality gap of 7.53%. Moreover, the PECH heuristic results in a 5.91% optimality gap but takes longer than 2SRL, achieving a time improvement of 229. Furthermore, the solution time of PECH increases significantly with the size of the problem, while the solution time of the 2SRL is a fraction of a second for all cases with a much larger number of scenarios than the trained instances. Besides, PECH has a higher average solution time and standard deviation than the 2SRL. Therefore, 2SRL can be preferred over the heuristic in an online application where a solution is needed instantly. Also, all p-values for the one-sided Wilcoxon signed-rank test are less than 0.01, confirming that the 2SRL is faster than the PECH heuristic. In addition, the 2SRL framework outperforms the random solution with a very large margin of more than 40% in terms of the optimality gap. On average, the random solution gives an optimality gap of 48.11% while having a similar solution time to 2SRL.

Table 2 presents another set of results for 2SRL. Here, the model is trained with 20-item problems, each with 10 scenarios. The first test set presented in the second column of Table 2 has the same number of scenarios and items as the training set. Here, an optimality gap of 8.47% is achieved using 2SRL in just milliseconds. The results follow similarities to the results in Table 1, but the optimality gaps are higher for the second and fourth sets of instances since they have a much larger set of items. Increasing the number of items in the

Table 1: Experimental results for 2SRL trained with 10 items and 10 scenarios

#sc	10	50	100	500	1000	Avg	Mdn	Std
#items	10	20	10	15	10	13	10	4
timeGRB	0.1	4,339.3	370.7	6,017.9	3,448.1	2,835.2	195.1	3,260.9
time2SRL	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0
timeSDDiP	3.0	130.3	35.5	451.7	356.1	195.3	100.8	206.0
timePECH	0.1	4.5	4.3	253.5	513.6	155.2	4.9	211.2
timeRand	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.1
timeImp2SRL	112	374,893	52,401	85,935	52,420	113,152	8,135	182,509
timeImpSDDiP	0	38	10	15	9	15	1	22
timeImpPECH	3	1,019	93	25	7	229	10	516
timeImpRand	90	374,395	28,649	69,634	26,642	99,882	5,508	181,609
optGapGRB(%)	0.00	0.07	0.01	0.15	0.04	0.05	0.01	0.10
optGap2SRL(%)	7.29	10.23	6.66	6.63	6.86	7.53	6.30	5.49
optGapSDDiP(%)	0.03	0.01	0.03	0.02	0.03	0.02	0.02	0.01
optGapPECH(%)	6.01	5.74	5.87	5.98	5.98	5.91	4.93	4.24
optGapRand(%)	47.76	48.62	47.33	47.69	49.15	48.11	46.68	7.10

test set brings out another layer of complication to a problem that is already challenging to predict and solve. However, those solutions are significantly better than the random solution in terms of optimality gaps and, therefore, 2SRL can provide substantial flexibility for solving instances with a varying number of variables. For all instances in Table 2, the p-values are smaller than 0.01, ensuring that 2SRL results in a faster solution than the PECH heuristic. For example, the last dataset with 1,000 scenarios is solved in almost 2 hours with Gurobi and returns an average gap of 0.41%. The SDDiP can only decrease the solution time to around 4,600 seconds. The PECH heuristic reduces the solution time to more than 2,000 seconds, with an impressive gap of 4.71%. However, the solution time with PECH grows rapidly with increasing problem size, and PECH might not be able to provide fast-enough solutions for high-speed applications requiring solutions in less than a second. In this case, 2SRL can provide a solution in 0.2 seconds, with a gap of 6.25%. This gap would outperform the random solution gap of 47.63% significantly.

Figure 3 presents the change in the optimality gap of the 2SRL framework with the increasing number of items for instances in Tables 1 and 2. Here, we present the horizontal axis as the number of items in the test set divided by the number of items in the training set since the instances in Tables 1 and 2 have a varying number of items. The figure introduces mixed results on item-wise generalization. For example, the optimality gap initially decreases with the increasing number of items for Table 1. However, for all other cases, the optimality gap increases as the ratio of test to training items increases. Therefore, the increase in the number of items might be challenging to handle, and the solution quality should be monitored.

The results presented in Tables 1 and 2 highlight the strength and potential of the 2SRL framework to tackle two-stage stochastic SPs. Our 2SRL framework can be positioned to tackle large and challenging problems with

Table 2: Experimental results for 2SRL trained with 20 items and 10 scenarios

#sc	10	50	100	500	1000	Avg	Mdn	Std
#items	20	40	20	30	20	26	20	8
timeGRB	3.1	7,200.0	6,485.7	7,200.1	6,896.3	5,557.0	7,200.0	3,023.5
time2SRL	0.0	0.0	0.0	0.2	0.2	0.1	0.0	0.1
timeSDDiP	68.1	6,357.5	497.1	7,041.4	4,644.5	3,721.7	3,788.5	3,130.5
timePECH	0.2	17.5	17.7	957.4	2,082.3	615.0	18.7	838.8
timeRand	0.0	0.0	0.0	0.1	0.2	0.1	0.0	0.1
timeImp2SRL	951	197,542	296,104	33,263	32,183	112,009	33,601	123,689
timeImpSDDiP	0	1	14	1	2	4	1	6
timeImpPECH	15	416	369	8	3	162	8	200
timeImpRand	1,284	340,555	321,654	48,832	34,393	149,344	45,836	159,850
optGapGRB(%)	0.00	0.15	0.23	0.57	0.41	0.27	0.22	0.27
optGap2SRL(%)	8.47	16.78	8.99	10.74	6.25	10.25	9.61	4.53
optGapSDDiP(%)	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01
optGapPECH(%)	4.98	5.57	4.86	5.03	4.71	5.03	5.32	1.58
optGapRand(%)	48.05	48.93	46.93	48.55	47.63	48.02	48.46	4.91

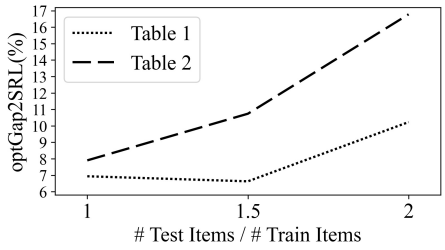


Fig. 3: The change in the optimality gap of 2SRL with the increasing ratio of test items to train items.

a high number of scenarios rather than smaller and easier problems, yielding significant benefits in the solution time reduction. For all 10 test datasets with different configurations, the 2SRL framework outperforms the PECH and all other exact approaches significantly in terms of solution times. Therefore, 2SRL creates an opportunity to be used in online applications by finding a good-quality solution in a fraction of a second. Such applications arise in airlines, ride-sharing, cloud data centers, and online advertising. When a solution to a two-stage stochastic program is needed almost instantly, 2SRL can be utilized with almost no upfront investment into development, unlike a heuristic. Agents can be trained with identified distributions and can be deployed without much challenge. However, the solution quality is lower than the specially-designed heuristic. Moreover, the time to generate a random solution is the same as the time to generate a solution with 2SRL in 9 out of 10 sets of instances with one significant digit in the optimality gap. This highlights that

our 2SRL framework can generate relatively high-quality solutions within the time budget of a naive and easy-to-implement solution approach. Considering the trade-off between the solution quality and solution time, 2SRL finds itself a place in the approaches favoring the solution time.

6 Discussion

In this study, we presented a reinforcement learning framework to solve scenario-based two-stage stochastic programs. Those problems commonly arise in various settings, but are NP-Hard. Their solution is not usually viable in fast, practical applications due to their computational burden unless a special solution strategy is developed. We intended to eliminate the process of crafting special solution approaches by automating it through learning a policy. Our 2SRL framework consists of two different agents that learn to solve each stage of the problem. We presented the details of training based on the policy gradient theorem. During training, Agent 1 gets feedback on the decision quality from Agent 2 since first-stage decisions affect both the first and second-stage objective function values. This is achieved by developing an updated gradient calculation equation for the REINFORCE algorithm. Furthermore, we introduced a strategy to isolate second-stage problems for training by sampling the first-stage coefficient. Additionally, we have presented a scenario sampling strategy for training that reduces the correlations and improves the training efficiency. The results show that the 2SRL framework can produce high-quality solutions very fast. For example, the solution time can be reduced from more than an hour to under a second with an optimality gap of 6.25%. Also, once a model is trained, it can be utilized to predict instances with the same distribution and structure but a larger number of items and scenarios, which provides a significant advantage in terms of the generalizability of the results. This flexibility opens doors for very large-scale problems to be used in online applications since high-quality solutions can be generated in a fraction of a second. While the heuristic solutions have a lower optimality gap, they are significantly slower than 2SRL and require rigorous analysis to develop. With 2SRL, the only requirement is a couple of days of training with randomly sampled problems. Considering this trade-off, 2SRL can be utilized to provide solutions to problems where generating special solutions is time or resource-consuming.

The 2SRL framework has the potential to pioneer and facilitate new research in this field. The future direction of study can involve building on 2SRL with different neural network architectures such as transformers. Reinforcement learning is a constantly-evolving area of research. Therefore, new training paradigms can increase the performance of the 2SRL framework. Further experiments can involve investigation with different network architectures, such as transformers based on self-attention. Different sampling methods, including active search and problem-specific search methods can be explored. Furthermore, new algorithms can be developed and integrated to achieve better success when predicting instances with changing number of items. Also,

future studies can focus on reducing the optimality gap even further while still maintaining the speed at which the solution is generated. Hierarchical reinforcement learning can be employed to solve multi-stage stochastic problems that can benefit from decomposing the long time horizon. In addition, frameworks that integrate reinforcement learning with other approaches can be investigated. For example, we can choose to use a commercial solver where reinforcement learning is not entirely confident in the predictions. Moreover, reinforcement learning agents can be integrated into classical operations research solution approaches, such as Lagrangean and Benders decompositions, to improve their performance.

Acknowledgements We gratefully acknowledge the support of the National Science Foundation CAREER Award co-funded by the CBET/ENG Environmental Sustainability program and the Division of Mathematical Sciences in MPS/NSF under Grant No. CBET-1554018. We also acknowledge the support of New Jersey Institute of Technology Academic & Research Computing Systems for the High Performance Computing resources. Also, we thank two reviewers and the editor for their helpful feedback, which helped improve our exposition's clarity.

Data Availability

The codes and datasets can be reached at https://github.com/dcan07/2SRL.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Babak Abbasi, Toktam Babaei, Zahra Hosseinifard, Kate Smith-Miles, and Maryam Dehghani. Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management. *Computers & Operations Research*, 119:104941, 2020.
2. Reza Refaei Afshar, Yingqian Zhang, Murat Firat, and Uzay Kaymak. A state aggregation approach for solving knapsack problem with deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 81–96. Cambridge, MA: Proceedings of Machine Learning Research, 2020.
3. Shabbir Ahmed, Mohit Tawarmalani, and Nikolaos V Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004.
4. Yalçın Akçay, Haijun Li, and Susan H Xu. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150(1):17–29, 2007.

5. Mostafa Al-Emran. Hierarchical reinforcement learning: a survey. *International Journal of Computing and Digital Systems*, 4(02), 2015.
6. Moses Amoasi Acquah, Daisuke Kodaira, and Sekyung Han. Real-time demand side management algorithm using stochastic optimization. *Energies*, 11(5):1166, 2018.
7. Gustavo Angulo, Shabbir Ahmed, and Santanu S Dey. Improving the integer l-shaped method. *INFORMS Journal on Computing*, 28(3):483–499, 2016.
8. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
9. Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy, and Chun Ye. Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*, 2019.
10. Gulay Barbarosoğlu and Yasemin Arda. A two-stage stochastic programming framework for transportation planning in disaster response. *Journal of the Operational Research Society*, 55(1):43–53, 2004.
11. Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
12. Yoshua Bengio, Emma Frejinger, Andrea Lodi, Rahul Patel, and Sriram Sankaranarayanan. A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 99–111. Cham, Switzerland: Springer, 2020.
13. Dimitris Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *INFORMS Journal on Computing*, 34(4):2229–2248, 2022.
14. John R Birge and Francois V Louveaux. *Introduction to stochastic programming*. New York, NY: Springer Science & Business Media, 2011.
15. Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Learning to solve vehicle routing problems: A survey. *arXiv preprint arXiv:2205.02453*, 2022.
16. Sabah Bushaj and İ Esra Büyüktahtakın. A K-means supported reinforcement learning algorithm to solve multi-dimensional knapsack problem. *Under Review*, 2023.
17. Sabah Bushaj, İ Esra Büyüktahtakın, and Robert G Haight. Risk-averse multi-stage stochastic optimization for surveillance and operations planning of a forest insect infestation. *European Journal of Operational Research*, 299(3):1094–1110, 2022.
18. Sabah Bushaj, Xuecheng Yin, Arjeta Beqiri, Donald Andrews, and İ Esra Büyüktahtakın. A simulation-deep reinforcement learning (SiRL) approach for epidemic control optimization. *Annals of Operations Research*,

2022.
19. Lucian Buşoniu, Robert Babuška, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38 (2):156–172, 2008.
20. Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in Multi-agent Systems and Applications-1*, 1:183, 2010.
21. İ Esra Büyüktahtakın. Stage-t scenario dominance for risk-averse multi-stage stochastic mixed-integer programs. *Annals of Operations Research*, 309(1):1–35, 2022.
22. İ Esra Büyüktahtakın. Scenario-dominance to multi-stage stochastic lot-sizing and knapsack problems. *Computers & Operations Research*, page 106149, 2023.
23. Pedro JS Cardoso, Gabriela Schütz, Andriy Mazayev, Emanuel Ey, and Tiago Corrêa. A solution for a real-time stochastic capacitated vehicle routing problem with time windows. *Procedia Computer Science*, 51:2227–2236, 2015.
24. Claus C Carøe and Rüdiger Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1-2):37–45, 1999.
25. Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32:6281–6292, 2019.
26. Steve Y Chius, Leonard Lu, and Louis Anthony Cox Jr. Optimal access control for broadband services: stochastic knapsack with advance information. *European Journal of Operational Research*, 89(1):127–134, 1996.
27. Halil I Cobuloglu and İ Esra Büyüktahtakın. A two-stage stochastic mixed-integer programming approach to the competition of biofuel and food production. *Computers & Industrial Engineering*, 107:251–263, 2017.
28. Amy Mainville Cohn and Cynthia Barnhart. The stochastic knapsack problem with random weights: A heuristic approach to robust transportation planning. *Proceedings of the Triennial Symposium on Transportation Analysis*, 3:17–23, 1998.
29. Jose L Crespo-Vazquez, C Carrillo, E Diaz-Dorado, Jose A Martinez-Lorenzo, and Md Noor-E-Alam. A machine learning based stochastic optimization framework for a wind and storage power plant participating in energy pool market. *Applied Energy*, 232:341–357, 2018.
30. Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, pages 465–480. PMLR, 2020.
31. George B Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955.
32. Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33:609–620,

2020.
33. Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Cham, Switzerland: Springer, 2018. ISBN 978-3-319-93031-2.
34. Lingquan Ding, Shabbir Ahmed, and Alexander Shapiro. A python package for multi-stage stochastic programming. *Optimization Online*, 2019.
35. Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1386–1394, 2019.
36. Csaba I Fábián and Zoltán Szőke. Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, 4(4):313–353, 2007.
37. Yiding Feng, Rad Niazadeh, and Amin Saberi. Two-stage stochastic matching with application to ride hailing. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pages 2862–2877. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2021.
38. Emma Frejinger and Eric Larsen. A language processing algorithm for predicting tactical solutions to an operational planning problem under uncertainty. *arXiv preprint arXiv:1910.08216*, 2019.
39. Dinakar Gade, Simge Küçükyavuz, and Suvrajeet Sen. Decomposition algorithms with parametric Gomory cuts for two-stage stochastic integer programs. *Mathematical Programming*, 144(1):39–64, 2014.
40. Dinakar Gade, Gabriel Hackebeil, Sarah M Ryan, Jean-Paul Watson, Roger J-B Wets, and David L Woodruff. Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming*, 157(1):47–67, 2016.
41. Alexei A Gaivoronski, Abdel Lisser, Rafael Lopez, and Hu Xu. Knapsack problem with probability constraints. *Journal of Global Optimization*, 49 (3):397–413, 2011.
42. Jiyao Gao and Fengqi You. Deciphering and handling uncertainty in shale gas supply chain design and optimization: Novel modeling framework and computationally efficient solution algorithm. *AIChE Journal*, 61(11):3739–3755, 2015.
43. Emilia Grass, Kathrin Fischer, and Antonia Rams. An accelerated l-shaped method for solving two-stage stochastic programs in disaster management. *Annals of Operations Research*, 284(2):557–582, 2020.
44. Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.
45. Shenshen Gu and Tao Hao. A pointer network based deep learning algorithm for 0–1 knapsack problem. In *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, pages 473–477. IEEE,

2018.
46. Shenshen Gu and Yue Yang. A deep learning algorithm for the max-cut problem based on pointer network structure with supervised learning and reinforcement learning strategies. *Mathematics*, 8(2):298, 2020.
47. Shenshen Gu, Tao Hao, and Hanmei Yao. A pointer network based deep learning algorithm for unconstrained binary quadratic programming problem. *Neurocomputing*, 390:1–11, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.06.111. URL https://www.sciencedirect.com/science/article/pii/S0925231220303398.
48. Ping Guo, Guo H Huang, Hua Zhu, and XL Wang. A two-stage programming approach for water resources management under randomness and fuzziness. *Environmental Modelling & Software*, 25(12):1573–1581, 2010.
49. Yongming He, Guohua Wu, Yingwu Chen, and Witold Pedrycz. A two-stage framework and reinforcement learning-based optimization algorithms for complex scheduling problems. *arXiv preprint arXiv:2103.05847*, 2021.
50. Martin N Hjelmeland, Jikai Zou, Arild Helseth, and Shabbir Ahmed. Nonconvex medium-term hydropower scheduling by stochastic dual dynamic integer programming. *IEEE Transactions on Sustainable Energy*, 10(1): 481–490, 2018.
51. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
52. Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.
53. Dawsen Hwang, Patrick Jaillet, and Vahideh Manshadi. Online resource allocation under partially predictable demand. *Operations Research*, 69 (3):895–915, 2021.
54. Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30:6351–6361, 2017.
55. Kibaek Kim and Sanjay Mehrotra. A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. *Operations Research*, 63(6):1431–1451, 2015.
56. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
57. Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018. https://openreview.net/forum?id=ByxBFsRqYm.
58. Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.
59. Gilbert Laporte and François V Louveaux. The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142, 1993.

60. Cristiana L Lara, John D Siirola, and Ignacio E Grossmann. Electric power infrastructure planning under uncertainty: stochastic dual dynamic integer programming (SDDiP) and parallelization scheme. *Optimization and Engineering*, 21(4):1243–1281, 2020.
61. Eric Larsen, Sébastien Lachapelle, Yoshua Bengio, Emma Frejinger, Simon Lacoste-Julien, and Andrea Lodi. Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS Journal on Computing*, 34(1):227–242, 2022.
62. Jian Li, Yue Wang, Michael R Lyu, and Irwin King. Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573*, 2017.
63. Kaiwen Li, Tao Zhang, and Rui Wang. Deep reinforcement learning for multiobjective optimization. *IEEE Transactions on Cybernetics*, 51(6): 3103–3114, 2020.
64. Bo Lin, Bissan Ghaddar, and Jatin Nathwani. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538, 2021.
65. Jeff Linderoth and Stephen Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2):207–250, 2003.
66. Abdel Lisser and Rafael Lopez. Stochastic quadratic knapsack with recourse. *Electronic Notes in Discrete Mathematics*, 36:97–104, 2010.
67. Miles Lubin, Kipp Martin, Cosmin G Petra, and Burhaneddin Sandıkçı. On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252–258, 2013.
68. Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
69. Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.
70. Yasemin Merzifonluoglu and Joseph Geunes. The risk-averse static stochastic knapsack problem. *INFORMS Journal on Computing*, 33(3): 931–948, 2021.
71. Hiroshi Morita, Hiroaki Ishii, and Toshio Nishida. Stochastic linear knapsack programming problem and its application to a portfolio selection problem. *European Journal of Operational Research*, 40(3):329–336, 1989.
72. Alejandro Mottini and Rodrigo Acuna-Agost. Deep choice model using pointer networks for airline itinerary prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1575–1583. New York, NY: Association for Computing Machinery, 2017.
73. Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

74. Joe Naoum-Sawaya and Samir Elhedhli. A stochastic optimization model for real-time ambulance redeployment. *Computers & Operations Research*, 40(8):1972–1978, 2013.
75. Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V Snyder. Reinforcement learning for solving the vehicle routing problem. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9861–9871. Red Hook, NY: Curran Associates Inc., 2018.
76. Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839, 2020.
77. Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.
78. András Prékopa. *Stochastic programming*. Dordrecht, Netherlands: Springer Science & Business Media, 2013.
79. R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.
80. Andrzej Ruszczyński. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35(3):309–333, 1986.
81. Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.
82. David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. Cambridge, MA: Proceedings of Machine Learning Research, 2014.
83. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
84. Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. Cambridge, MA: Proceedings of Machine Learning Research, 2020.
85. Simon Thevenin, Yossiri Adulyasak, and Jean-François Cordeau. Material requirements planning under demand uncertainty using stochastic optimization. *Production and Operations Management*, 30(2):475–493, 2021.
86. Simon Thevenin, Yossiri Adulyasak, and Jean-François Cordeau. Stochastic dual dynamic programming for multiechelon lot sizing with component substitution. *INFORMS Journal on Computing*, 2022.
87. Richard M Van Slyke and Roger Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal*

on Applied Mathematics, 17(4):638–663, 1969.
88. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
89. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in Neural Information Processing Systems*, 28:2692–2700, 2015.
90. Jing Wang, Haoxiong Yang, and Jianming Zhu. A two-stage stochastic programming model for emergency resources storage region division. *Systems Engineering Procedia*, 5:125–130, 2012.
91. Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
92. Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.
93. Fei Wu and Ramteen Sioshansi. A two-stage stochastic optimization model for scheduling electric vehicle charging loads to relieve distribution-system constraints. *Transportation Research Part B: Methodological*, 102:55–82, 2017.
94. Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning scenario representation for solving two-stage stochastic integer programs. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=06Wy2BtxXrz.
95. Joyce W Yen and John R Birge. A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40(1):3–14, 2006.
96. Dogacan Yilmaz and İ Esra Büyüktahtakın. An expandable learning-optimization framework for sequentially dependent decision-making. *Submitted to European Journal of Operational Research*, 2022.
97. Dogacan Yilmaz and İ Esra Büyüktahtakın. Learning optimal solutions via an LSTM-optimization framework. *Accepted for Publication in Operations Research Forum*, 2023.
98. Fengqi You and Ignacio E Grossmann. Mixed-integer nonlinear programming models and algorithms for large-scale supply chain design with stochastic inventory management. *Industrial & Engineering Chemistry Research*, 47(20):7802–7817, 2008.
99. Jikai Zou, Shabbir Ahmed, and Xu Andy Sun. Stochastic dual dynamic integer programming. *Mathematical Programming*, 175(1):461–502, 2019.