Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

INTEGRATED MACHINE LEARNING AND OPTIMIZATION APPROACHES

by
Dogacan Yilmaz

This dissertation focuses on the integration of machine learning and optimization. Specifically, novel machine learning-based frameworks are proposed to help solve a broad range of well-known operations research problems to reduce the solution times. The first study presents a bidirectional Long Short-Term Memory framework to learn optimal solutions to sequential decision-making problems. Computational results show that the framework significantly reduces the solution time of benchmark capacitated lot-sizing problems without much loss in feasibility and optimality. Also, models trained using shorter planning horizons can successfully predict the optimal solution of the instances with longer planning horizons. For the hardest data set, the predictions at the 25% level reduce the solution time of 70 CPU hours to less than 2 CPU minutes with an optimality gap of 0.8% and without infeasibility. In the second study, an extendable prediction-optimization framework is presented for multi-stage decision-making problems to address the key issues of sequential dependence, infeasibility, and generalization. Specifically, an attention-based encoder-decoder neural network architecture is integrated with an infeasibility-elimination and generalization framework to learn high-quality feasible solutions. The proposed framework is demonstrated to tackle the two well-known dynamic NP-Hard optimization problems: multi-item capacitated lot-sizing and multi-dimensional knapsack. The results show that models trained on shorter and smaller-dimension instances can be successfully used to predict longer and larger-dimension problems with the presented item-wise expansion algorithm. The solution time can be reduced by three orders of magnitude with an average optimality gap below 0.1%. The proposed framework can be

advantageous for solving dynamic mixed-integer programming problems that need to be solved instantly and repetitively. In the third study, a deep reinforcement learning-based framework is presented for solving scenario-based two-stage stochastic programming problems, which are computationally challenging to solve. A general two-stage deep reinforcement learning framework is proposed where two learning agents sequentially learn to solve each stage of a general two-stage stochastic multi-dimensional knapsack problem. The results show that solution time can be reduced significantly with a relatively small gap. Additionally, decision-making agents can be trained with a few scenarios and solve problems with a large number of scenarios. In the fourth study, a learning-based prediction-optimization framework is proposed for solving scenario-based multi-stage stochastic programs. The issue of non-anticipativity is addressed with a novel neural network architecture that is based on a neural machine translation system. Furthermore, training the models on deterministic problems is suggested instead of solving hard and time-consuming stochastic programs. In this framework, the level of variables used for the solution is iteratively reduced to eliminate infeasibility, and a heuristic based on a linear relaxation is performed to reduce the solution time. An improved item-wise expansion strategy is introduced to generalize the algorithm to tackle instances with different sizes. The results are presented in solving stochastic multi-item capacitated lot-sizing and stochastic multi-stage multi-dimensional knapsack problems. The results show that the solution time can be reduced by a factor of 599 with an optimality gap of only 0.08%. Moreover, results demonstrate that the models can be used to predict similarly structured stochastic programming problems with a varying number of periods, items, and scenarios. The frameworks presented in this dissertation can be utilized to achieve high-quality and fast solutions to repeatedly-solved problems in various industrial and business settings, such as production and inventory management, capacity planning, scheduling, airline logistics, dynamic pricing, and emergency management.

INTEGRATED MACHINE LEARNING AND OPTIMIZATION APPROACHES

by
Dogacan Yilmaz

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Industrial Engineering

Department of Mechanical and Industrial Engineering

December 2022

APPROVAL PAGE

INTEGRATED MACHINE LEARNING AND OPTIMIZATION APPROACHES

Dogacan Yilmaz

Dr. İ. Esra Büyüktahtakın Toy, Dissertation Advisor — Date
Associate Professor of Industrial and Systems Engineering,
Virginia Tech, Blacksburg, VA

Dr. Sanchoy K. Das, Committee Chair — Date
Professor of Mechanical and Industrial Engineering, NJIT

Dr. Athanassios Bladikas, Committee Member — Date
Associate Professor of Mechanical and Industrial Engineering, NJIT

Dr. Wenbo Cai, Committee Member — Date
Associate Professor of Mechanical and Industrial Engineering, NJIT

Dr. David A. Bader, Committee Member — Date
Distinguished Professor of Data Science, NJIT

BIOGRAPHICAL SKETCH

Author: Dogacan Yilmaz

Degree: Doctor of Philosophy

Date: December 2022

Undergraduate and Graduate Education:

- Doctor of Philosophy in Industrial Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2022
- Bachelor of Science in Industrial Engineering,
  Boğaziçi University, İstanbul, Türkiye, 2019

Major: Industrial Engineering

Publications:

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). A non-anticipative learning-optimization framework for solving multi-stage stochastic programs. *In preparation.*

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). A deep reinforcement learning framework for solving two-stage stochastic programs. *In preparation.*

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). An expandable learning-optimization framework for sequentially dependent decision-making. *Submitted to European Journal of Operational Research.*

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). Learning optimal solutions via an LSTM-optimization framework. *Submitted to Operations Research Forum.*

Guner, G., Yilmaz, D., Eskin, D., and Bilgili, E. (2022). Effects of bead packing limit concentration on microhydrodynamics-based prediction of breakage kinetics in wet stirred media milling. *Powder Technology*, 403, 117433.

Guner, G., Yilmaz, D., and Bilgili, E. (2021). Kinetic and microhydrodynamic modeling of fenofibrate nanosuspension production in a wet stirred media mill. *Pharmaceutics*, 13(7), 1055.

Presentations:

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). Learning to solve multistage optimization problems with an expendable framework. *Institute for Operations Research and the Management Sciences Annual Meeting*, Indianapolis, IN.

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). An expandable learning-optimization framework for sequentially dependent decision-making. *17th Institute for Operations Research and the Management Sciences Workshop on Data Mining and Decision Analytics*, Indianapolis, IN.

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). An expandable learning-optimization framework for sequentially dependent decision-making. *Dana Knox Research Showcase*, NJIT, Newark, NJ.

Yilmaz, D., and Büyüktahtakın, İ. E. (2022). An expandable learning-optimization framework for sequentially dependent decision-making. *Institute for Operations Research and the Management Sciences Computing Society Conference*, Tampa, FL.

Yilmaz, D., and Büyüktahtakın, İ. E. (2021). An expandable learning-optimization framework for sequentially dependent decision-making. *Institute for Operations Research and the Management Sciences Annual Meeting*, Online.

Yilmaz, D., and Büyüktahtakın, İ. E. (2021). An LSTM-optimization framework to predict the optimal solution of a mixed-integer program. *Mixed-Integer Programming Workshop*, Online.

Yilmaz, D., and Büyüktahtakın, İ. E. (2020). An LSTM-optimization framework to predict the optimal solution of a mixed-integer program. *Institute for Operations Research and the Management Sciences Annual Meeting*, Online.

Yilmaz, D., and Büyüktahtakın, İ. E. (2020). An LSTM-optimization framework to predict the optimal solution of a mixed-integer program. *15th Institute for Operations Research and the Management Sciences Workshop on Data Mining and Decision Analytics*, Online.

I am dedicating this dissertation to my family who was always there for me. They mean everything to me.

ACKNOWLEDGMENT

First and foremost, I wish to express my most sincere gratitude to my advisor, Dr. İ. Esra Büyüktahtakın Toy. Her excellent advice and never-ending motivation is the main reason this dissertation exists. She was always there for me with guidance and encouragement. I feel lucky that I had Dr. Büyüktahtakın Toy as my advisor. I will never forget her support and kindness.

I would like to express my appreciation to Dr. Sanchoy K. Das for his decision to be my committee chair. I would like to expand my deepest thanks to the committee members, Dr. Athanassios Bladikas, Dr. Wenbo Cai, and Dr. David A. Bader. Their invaluable insights have enriched my research in many ways that I could not think of.

I had the pleasure of working with Dr. Sabah Bushaj, Dr. Xuecheng Yin, and Elson Cibaku during my time at the Systems Optimization and Data Analytics Lab. They were always present for me to discuss ideas and provide suggestions.

I gratefully acknowledge the support of the National Science Foundation (NSF) CAREER Award co-funded by the Chemical, Bioengineering, Environmental and Transport Systems/Engineering Environmental Sustainability program and the Division of Mathematical Sciences in Mathematical and Physical Sciences/NSF under Grant No. CBET-1554018.

My mother, Sevim and father, Hamit were there for me when I needed them. Their patience and encouragement were there for me throughout the years. I will be forever indebted to them. I was lucky enough to have amazing friends who have been on my side through the years. Finally, I owe my deepest thanks to Gülenay. I could not have completed this dissertation without the support and strength she brings every day.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF TABLES
(Continued)

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Analytics can be defined as the development and application of scientific methodologies to analyze complex systems. Analytics is used to discover, interpret, and impart patterns in data. Broadly, it consists of three distinct types of analytics. Descriptive analytics is used for gaining meaningful insight into the past by analyzing historical data. It is the simplest form of analytics. Predictive analytics is used to predict the future by looking at historical data. The final frontier of analytics is prescriptive analytics which is used for decision-making by suggesting actions that optimize a certain metric. This dissertation is positioned at the intersection of predictive and prescriptive analytics.

Operations research is the scientific discipline that develops and utilizes advanced mathematical methods to make better decisions. While optimization is ageless, as a scientific discipline, the history of modern operations research dates back to the Second World War, and it has expanded rapidly after that with new techniques. It can be considered as the mathematical tool used for prescriptive analytics. Many real-world problems in manufacturing, finance, healthcare, and logistics can be modeled with the language of operations research and can be solved with highly developed methodologies to provide insight into optimal decision-making.

Machine learning is a subfield of artificial intelligence. It is the study of learning by looking at the data. Machine learning is one of the computational engines of predictive analytics. It aims to identify patterns by analyzing the data to make predictions. Like operations research, machine learning has a broad range of applications, including but not limited to speech recognition, product recommendation, and self-driving cars.

In this dissertation, we focus on the intersection of machine learning and operations research. Specifically, we employ machine learning algorithms to solve problems that are modeled with the language of operations research. We aim to provide faster and better solutions to various types of optimization problems by integrating two unique disciplines. Our motivation is the success of various machine learning applications in operations research and the potential impact of generating high-quality solutions fast without a need to craft a special solution methodology.

1.1.1 Machine Learning

Machine learning can be divided into three subfields. Unsupervised learning focuses on self-discovering important patterns in unlabeled data. Unsupervised learning examples include k-means clustering, association rule mining, and principal component analysis. On the other hand, supervised learning algorithms aim to learn a mapping from inputs to correct outputs using a labeled dataset. Supervised learning employs a training set to learn, and a test set to independently test its success using algorithms such as decision trees, support vector machines, and neural networks. Lastly, reinforcement learning utilizes a different paradigm than unsupervised and supervised learning. The reinforcement learning agent learns by interacting with the environment it is in to maximize a cumulative reward, a paradigm similar to how humans learn.

For the past decade, a subfield of machine learning called deep learning has dominated the landscape of artificial intelligence fueled by ever-growing computational power and big data technologies. Some of the most well-known success includes surpassing human-level accuracy in image recognition (He et al., 2015a), generating realistic-looking superficial images (Karras et al., 2018), and developing accurate neural machine translation models (Wu et al., 2016). Deep learning uses artificial neural networks inspired by the workings of animal brains and biological neural

networks. One of the early works on neural networks is the invention of perceptron by Rosenblatt (1957) and the development of multilayer perceptron, which consists of two or more layers of artificial neural networks. Each neuron or node in each layer uses a nonlinear activation function, and its parameters are updated based on a technique called backpropagation. After going through several so-called winters where the interest in the research has slowed down, the rise of enthusiasm and optimism has grown since the 1990s and living the spring since 2010.

Derived from perceptrons, recurrent neural networks allow neurons to make temporal connections and, therefore, can process sequential data. Invented by Hopfield (1982), recurrent neural networks share memory between time stages that allow information to flow through the time steps in the sequence, and error is backpropagated through time. This structure of recurrent neural networks makes it the perfect candidate for learning sequential tasks such as speech recognition, language translation, and time series prediction since it can handle variable-length input sequences. One of the major problems encountered during the training of recurrent networks is the vanishing gradient problem during backpropagation, which makes recurrent neural networks inadequate for learning long-term dependencies. To combat this vulnerability, gated recurrence architectures have been proposed, and one of them is Long Short-Term Memory (LSTM) developed by Hochreiter and Schmidhuber (1997). An LSTM unit is called a memory cell and controls the flow of information with an input gate, an output gate, and a forget gate. During the last decade, LSTM has been used within different neural architectures and achieved tremendous results in various domains such as music composition (Eck and Schmidhuber, 2002), sentiment analysis (Tang et al., 2015), and drug design (Gupta et al., 2018).

Sequence-to-sequence learning was another course of study that thrived in the last decade. Some of these architectures have used the LSTM cell as their computing

unit and were initially developed for neural machine translation. They are considered to be the natural choice for sequential learning because, unlike classical multilayer perceptrons, they can handle variable input and output sizes. Also, unlike deep recurrent neural networks, the decision made at the previous time can be an input to prediction at the current time. In Chapters 2, 3, and 5 of this dissertation, we utilize sequence-to-sequence learning frameworks to predict the solution of operations research problems. Sutskever et al. (2014) present an encoder-decoder architecture for neural machine translation and use LSTMs as their computing cell. The encoder is a recurrent neural network that encodes the input sequence one at a time. The hidden state at the last time period of the encoder is a fixed dimensional representation of the input sequence. The decoder is another recurrent neural network that decodes the output sequence one at a time, starting with the fixed dimensional representation of the input sequence generated by the encoder. Cho et al. (2014) present a similar encoder-decoder architecture with the addition of feeding a fixed dimensional representation of the input sequence to the decoder at each time step. Bahdanau et al. (2014) introduce a mechanism called attention for encoder-decoder networks. It is a structure that enables the decoder to selectively focus on the important information of the encoder during the prediction time. Therefore, the whole output sequence is not just generated by the fixed dimensional representation, and long sentences can be better coped with. Luong et al. (2015) present a local attention model with a modification of the attention mechanics with three different methods to calculate the attention scores.

Deep reinforcement learning was the other branch of machine learning that received much attention in the 2010s. In the reinforcement learning paradigm, an agent interacts with the environment that it is currently in to maximize the total reward. It is a setting where learning is done by trial and error (Sutton and Barto, 2018) and aims to find a balance between current and unexplored

knowledge. The roots of reinforcement learning can be traced back to dynamic programming and Bellman equations (Bellman, 1966). The learning environment in reinforcement learning can be expressed as a Markov decision process, and traditional solution algorithms use dynamic programming-based techniques to solve the system (Büyüktahtakın, 2011). One of the key differences between dynamic programming and reinforcement learning is that while dynamic programming has the perfect knowledge of the model, including transition dynamics and reward functions, reinforcement learning learns by interacting with its environment and observing the transition and reward values over time (Sutton and Barto, 2018). The artificial intelligence and computer science communities use the term reinforcement learning to describe the field. The same field is called neuro-dynamic programming in control theory, and approximate dynamic programming is the terminology in the operation research community (Powell, 2009). Deep reinforcement learning integrates the approximation power of neural networks into reinforcement learning to eliminate the need for a manually designed state space. Therefore, it can handle large state spaces and generalize to unseen states. In recent years, deep reinforcement learning has made a significant and promising impact. In a well-known example, AlphaGo is the first computer program that defeated the human champion in a complex game of Go where there are 10^{170} configurations possible (Silver et al., 2016). Its broad range of applications includes learning robotic actions directly from images (Levine et al., 2016), optimizing chemical reactions (Zhou et al., 2017), and self-driving (Sallab et al., 2017).

1.1.2 Machine Learning for Operations Research

The advancements and promising results of machine learning in the last decade have gained attention from a broad range of disciplines, including operations research. Operations research and machine learning has been naturally connected through

optimization. The idea of using machine learning for combinatorial optimization problems is not new. Hopfield and Tank (1985) introduce the idea of solving the traveling salesperson problem using neural networks. The advancements were limited during the 1990s due to erratic views of the field and hardware limitations (e.g., see the survey of Smith (1999)). However, the most recent advancements in machine learning have caused a spark in solving combinatorial optimization with machine learning.

Vinyals et al. (2015b) propose a novel neural architecture called pointer networks to solve the traveling salesperson problem by pointing to an input element at each decoding step. Khalil et al. (2016) propose a machine learning framework for variable selection for branch-and-bound decisions during the solution of a mixed-integer program. Khalil et al. (2017b) propose a learning approach to decide at which nodes a heuristic should be run so that the overall efficiency of the solver is increased. Fischetti and Fraccaro (2019) train machine learning models to predict the optimal objective function value for the offshore wind farm layout problem. Oroojlooyjadid et al. (2019) present a learning framework by employing neural networks to predict optimal order quantity for the infamous newsvendor problem. Xavier et al. (2021) utilize several machine learning techniques to predict unnecessary constraints, good starting solutions, and likely spaces of solutions for a repeatedly solved mixed-integer program. Bertsimas and Stellato (2021) present a learning-based framework for predicting a set of integer variables and tight constraints for convex mixed-integer programs. Anderson et al. (2022) present a novel learning approach by utilizing generative neural networks to reduce the solution time of the transient gas optimization problem by 60%. Donti et al. (2021) outline a methodology to enforce constraints, which is a major challenge with deep learning-based approaches.

Abbasi et al. (2020) present a learning-based approach to solving large-scale stochastic optimization problems by predicting only the first-stage decision variables

since they are immediately actionable variables. Bengio et al. (2020) suggest a novel approach to solving two-stage stochastic programs by predicting a representative scenario that can ensure the model gives the optimal solution. Wu et al. (2021) deploy conditional variational autoencoders to solve two-stage stochastic optimization problems by scenario reduction and objective prediction. Dumouchelle et al. (2022) present a neural network framework to approximate the expected second-stage cost and generate a surrogate model that is easier to solve than the extended two-stage formulation. Larsen et al. (2022b) present a learning framework using multilayer perceptrons to predict expected second-stage decisions under imperfect information.

Bello et al. (2016) present a novel framework for solving the traveling salesperson problem with reinforcement learning. Khalil et al. (2017a) propose a reinforcement learning framework to solve optimization problems over graphs. Nazari et al. (2018) develop a reinforcement learning methodology and a novel neural architecture based on pointer networks to solve the vehicle routing problem. Kool et al. (2018) present a novel reinforcement learning approach using a neural architecture called transformers to solve the vehicle routing problem. Deudon et al. (2018) focus on using a multi-head attention mechanism together with a reinforcement learning-based approach to solve the traveling salesperson problem. Lu et al. (2019) integrate heuristics and reinforcement learning to iteratively improve the existing feasible solution. Hubbs et al. (2020) present detailed results for utilizing reinforcement learning for various types of optimization problems and an open-source library for further testing. Afshar et al. (2020) aim to solve a knapsack problem using reinforcement learning with a state aggregation strategy to reduce the state space of the problem. Li et al. (2021) present a reinforcement learning methodology to solve multiobjective optimization problems by a decomposing and parameter transferring strategy.

As previously noted, the literature on the intersection of machine learning and operations research is growing rapidly. Bengio et al. (2021) review the most recent

advances in the field of solving combinatorial optimization with machine learning, and Mazyavkina et al. (2021) review the same subject with a focus on reinforcement learning in particular.

1.1.3 Operations Research

This dissertation explores supervised learning and reinforcement learning methodologies to solve traditional operations research problems. One of those problems is the capacitated lot-sizing problem. This NP-hard problem is one of the most important and difficult problems in production planning and often arises in the production, medical, and chemical industries (Karimi et al., 2003). The objective of the problem is the minimization of the sum of production, setup, and inventory holding cost while satisfying the demand without exceeding production capacity. It can be a single-item problem or a much more complex multi-item problem (Karimi et al., 2003). The problem is well-studied in operations research, and the traditional solution algorithms include dynamic programming (Florian et al., 1980), generating valid inequalities (Barany et al., 1984), variable redefinition (Eppen and Martin, 1987), and cutting planes (Hartman et al., 2010; Büyüktahtakın et al., 2018b). Another problem that we are attempting to solve using machine learning algorithms is a multi-stage version of the knapsack problem. In this version, the stability of the solution through time is important. The problem considered is NP-Hard and can be applied in many settings, including computing capacity management in data centers, where the resource prices vary over time (Bampis et al., 2022).

Stochastic optimization is a mathematical modeling framework for problems involving uncertainty. Different from deterministic programming, some or all parameters of the problem are uncertain at the time of decision-making. The discipline can be considered at the intersection of operations research, mathematics, probability, and statistics (Birge and Louveaux, 2011). Since it enables uncertainty

to be modeled into the decision-making, stochastic programming is very popular in a broad range of real-world applications, including water resources management (Huang and Loucks, 2000), financial planning (Mulvey and Shetty, 2004), and electricity procurement (Carrión et al., 2007). One of the most common examples of stochastic programs is the two-stage stochastic programming, originated by Dantzig (1955). In such a setting, the decisions are made sequentially in two stages, and uncertainty is observed between stages. Usually, the problem's objective is to minimize the first-stage and expected second-stage costs. The uncertainty is generally modeled by a finite set of realizations called scenarios. Such problems are usually considered to be hard with three levels of difficulty (Ahmed, 2010): Firstly, evaluating the second-stage cost for a particular scenario given first-stage decisions; secondly, evaluation of the expected cost of the second stage given first-stage decisions; and lastly, minimization of the expected second-stage cost. The solution approaches include the L-shaped method (Van Slyke and Wets, 1969), regularized decomposition (Ruszczyński, 1986), and multicut algorithm (Birge and Louveaux, 1988). Multi-stage stochastic programming is an extension of two-stage stochastic programs to multi-stage settings but is considered harder to solve (Birge and Louveaux, 2011). The uncertainty is characterized by a scenario tree, and the solution methodologies include nested decomposition (Ho and Manne, 1974) and L-shaped method (Louveaux, 1980).

1.2 Motivation

In many practical applications, operations research problems are solved repeatedly. Usually, problems with the same structures are solved repeatedly with slightly changing parameters belonging to the same or different distributions. In this dissertation, we focus on reducing the solution times of sequential decision-making problems. Two or multi-stage problems materialize in many industrial applications.

Such cases of repeatedly-solved problems commonly arise in operations planning and management, including finance, energy demand-side management, airline scheduling, and vehicle routing. When solving similarly-structured problems frequently, decision-makers can benefit significantly from a fast solution approach that can significantly reduce the solution time. Therefore, we are motivated to address this broad category of sequential problems to reduce the solution times with learning-based methodologies when they are needed to be solved frequently. For example, the airline industry frequently observes disruptions to their planned schedule and can benefit from fast solutions to aircraft and crew scheduling problems. One option to achieve that can be a problem-specific solution algorithm, such as a heuristic designed by an analytics professional. However, this can be time and resource-consuming for the problem owner. Additionally, depending on the proposed solution, a new or adapted solution might be required when the problem description changes. That can trigger a time-consuming makeover or a costly overhaul. We are motivated by these circumstances that can disrupt or restrict business flow and aspire to eliminate or reduce the need for crafting problem-specific solution algorithms. Therefore, we look at the second option and focus on using learning-based frameworks to generate solutions in a fast manner. By doing so, we hope to generate high-quality solutions in a fast manner.

We illustrate the results of our learning-based frameworks through well-known problems: lot-sizing and knapsack. The mentioned problems are utilized highly in many industries, including traditional mining (Samavati et al., 2017), crypto mining (Monem et al., 2022), advertising (Hao et al., 2020), agriculture (Cobuloglu and Büyüktahtakın, 2015b; Boonmee and Sethanan, 2016), ecological conservation (Büyüktahtakın et al., 2011; Kıbış and Büyüktahtakın, 2017; Onal et al., 2020), and logistics (Bruno et al., 2014). Therefore, demonstrating our methodologies through such practical problems is a motivating force for us due to its potential

impact. For example, the knapsack problem is at the core of all budget-constrained resource allocation problems, with applications arising from agriculture and energy (Cobuloglu and Büyüktahtakın, 2014, 2015a, 2017; Kantas et al., 2015), capital asset management (Büyüktahtakın et al., 2014b; Büyüktahtakın and Hartman, 2016; Liu et al., 2021), healthcare (Bushaj et al., 2022b; Coşgun and Büyüktahtakın, 2018; Kıbış and Büyüktahtakın, 2019; Yin et al., 2023), and ecological conservation (Büyüktahtakın et al., 2011, 2014a; Büyüktahtakin et al., 2014; Büyüktahtakın et al., 2015; Büyüktahtakın and Haight, 2018). Moreover, lot-sizing and its variations are fundamental for numerous applications in carbon tax regulations (He et al., 2015b; Lamba et al., 2019), semiconductor manufacturing (Quadt and Kuhn, 2005; Xiao et al., 2015), energy systems (Wichmann et al., 2019), dairy production (Kopanos et al., 2010), and supply chains (Kaminsky and Simchi-Levi, 2003; Pan et al., 2009). Therefore, practical solutions to the knapsack and lot-sizing problems could provide a tremendous impact in solving combinatorial optimization problems in many industrial, business, and social settings. Throughout the dissertation, we work with the different versions of the mentioned optimization problems that include single item, multiple items, different numbers of dimensions, and varying levels of uncertainty to show our proposed learning-based solutions can be utilized for business problems.

Uncertainty is crucial in decision-making. Accounting for it can result in better capturing of real-world problems and therefore generate better solutions. However, this task can be challenging. First, the uncertainty often is not easily quantifiable. However, scenario-based two or multi-stage stochastic programs can be utilized for this task. Such modeling language is commonly utilized in airline revenue management, capacity planning, epidemic control planning, and risk-averse optimization. Second, it can be even harder to solve such problems due to their substantial size. Stochastic programming is powerful at modeling real-world problems, but stochastic programs are quite challenging and time-consuming to solve,

especially when modeled with binary variables. Therefore, they are not often suitable for practical applications that are solved commonly, or a speedy solution is needed unless a special solution methodology is developed. We are encouraged by the vast applicability of scenario-based problems and motivated to reduce their solution times.

In general, supervised learning algorithms operate like a function that maps inputs to outputs. In this case, the input is the problem parameters, and the output is the optimal solution. Even though learners can be trained to learn such pairs, the predicted solutions might not satisfy the problem requirements of mathematical optimization. Even with a slight wrong prediction of optimal decision variables, problems can easily become infeasible and unimplementable. Therefore, we are inspired by this complex challenge and motivated to develop methodologies that can ensure the feasibility of predictions. Furthermore, scenario-based multi-stage problems require a very strict property of non-anticipativity. This property ensures the decisions up to a certain point should be the same for scenarios that share uncertainty up until that point. In short, non-anticipativity is a key feature in the execution of decisions. Hence, we are motivated to certify that our predictions follow this property.

Even though some advancements have been achieved with newly developed architectures and algorithms, a research gap still exists in integrating recent progress in machine learning and traditional optimization solvers like CPLEX or Gurobi. Such integration can enhance to performance of both tools and eliminate or reduce the drawbacks of using one of them. By integrating both branches of knowledge, a powerful tool can be constructed to achieve fast and good solutions. Furthermore, introducing the heuristics to this picture can further increase the chance of an improved methodology and enhance the solution by capturing knowledge about problem characteristics. This powerful potential inspires us to pursue this collaboration.

Additionally, many learning algorithms have the potential to generalize to different types of instances. If generalization can be established, learning from easier instances to solve harder instances can be highly beneficial for three reasons. First, generating training instances can be cumbersome. Supervised learning requires optimal solutions to problems for learning. Specifically, neural networks are known for their large amounts of data requirements. Solving millions of hard instances can be impractical or intractable. Therefore, generating training data using easier instances can ease the computational requirements for the training dataset. Second, model training time can be reduced significantly if the model can learn from small-sized problems. Neural networks can be challenging to train and require computationally-demanding hyperparameter optimization. When a model is trained using a small-sized dataset, there will be simply fewer data and fewer operations. Therefore, the model training will be faster, which can make a meaningful impact during the overall training paradigm. Third, many applications can require a certain amount of flexibility. The trained model and its performance should be robust to perturbations in the distributions of the input data. Also, the models should be robust to changes in the number of stages, items, scenarios, and other problem dimensions. Otherwise, training models from scratch at each slight change would be impractical. While generalization can be challenging, the mentioned benefits can provide tremendous advantages. Therefore, such potential promise motivates us to pursue generalization in various dimensions.

As mentioned in the previous section, there has been a significant increase in the literature on using machine learning for operations research. Encouraging results have been achieved in solving various types of problems, including knapsack, vehicle routing, and traveling salesperson problems. However, there is still a gap exists in learning-based frameworks. We are motivated by the promising advancements in supervised and reinforcement learning for sequential optimization problems.

1.3 Summary of Research Objectives and Contributions

This dissertation aims to develop novel methodologies for solving combinatorial and stochastic optimization problems by exploring and developing state-of-the-art machine learning algorithms. Also, we integrate the learning paradigm into the preexisting and computationally-advanced mathematical solver to harness the power of both domains.

The research objective in Chapter 2 is to present an LSTM-based optimization approach to solve sequential decision-making problems. The decisions are highly interrelated between the problem periods. Therefore, a machine learning approach that considers sequential dependency, like LSTM, is used. Our goal is to reduce the solution time when numerous similar problems are solved repeatedly. To our knowledge, this study is the first to use LSTM to predict a binary variable for the lot-sizing problem and partially utilizes predictions when solving the problem with CPLEX to reduce the solution times. We compare our LSTM-Optimization framework with logistic regression and random forest to show that LSTMs can capture sequential dependency. We compare our framework with traditional operations research approaches, including dynamic programming (Florian et al., 1980), dynamic programming-based inequalities (Hartman et al., 2010), and (ℓ,S) inequalities (Barany et al., 1984), and show that even though those exact approaches may reduce the solution times, they are not able to find solutions very fast. We define metrics to measure the efficiency of algorithms in terms of feasibility and optimality. To increase the feasibility of the solutions, we propose to use predictions partially with CPLEX. The trained models can reduce the solution times by an order of magnitude for the instances with the same distribution. Moreover, we test the generalization properties of our framework and report that a model trained with smaller periods can be used to predict instances with a large number of periods to achieve a significant time gain. For example, the average solution time of 70 CPU hours can be reduced to only 2

minutes with a 0.8% optimality gap without an infeasible solution. Our framework can be used in settings where practical and recurring sequential decision-making problems with similar structures are solved, such as power generation scheduling, energy demand-side management, and pricing optimization. The work based on this chapter is under review in Yilmaz and Büyüktahtakın (2022b).

The objective of the study presented in the third chapter is to reduce the solution times while guaranteeing the feasibility of the solutions when solving sequential decision-making problems. The presented framework improves on the one presented in Chapter 2 by developing an encoder-decoder model with sliding attention windows to specifically expand in the time dimension. To our knowledge, this is the first study that develops an encoder-decoder model to predict multi-period optimization problems and use the predictions within a mathematical solver. Our approach employs a local attention window to capture problem dynamics over a long planning horizon. This structure enables the model to selectively focus on a few nearby periods near the current decision period. Furthermore, the proposed framework can learn from instances with a small number of periods and generate high-quality solutions for problems that have a large number of periods. Additionally, we present an item-wise expansion algorithm to expand the model's predictive capabilities in the item dimension. This algorithm enables the model to learn from problems with a few items and predict the problems with a much larger set of items. Therefore, a significant time gain in training set generation and training time is achieved without sacrificing the solution quality. We propose a novel methodology to tackle the challenge of infeasible predictions. In this approach, we iteratively reduce the prediction level used during the solution of the problem until a feasible solution is found. For this reason, we generate a relaxation of the problem using a trained neural network to significantly reduce the feasibility checking time. We present the results of our framework on two fundamental operations research problems: Multi-item

Capacitated Lot-Sizing Problem and Multi-stage Knapsack Problem. We generate benchmark instances and compare the solution quality of our prediction-optimization framework with commercial solver, heuristics, and (ℓ,S) inequalities of (Barany et al., 1984). The results show that the framework can generate all-feasible solutions and reduce the solution time by a factor of 7,236 with an optimality gap of only 0.11%. Also, the item-wise expansion algorithm allows the trained model to predict instances that have 10 times more items than they are trained with. The presented framework outperforms the utilized heuristics in terms of both solution time and quality. Our framework can be utilized to solve problems with similar structures repeatedly to achieve noteworthy reductions in solutions time with a generalization potential in both time and item dimensions. The work based on Chapter 3 is under review in Yilmaz and Büyüktahtakın (2022a).

In Chapter 4, we aim to develop a deep reinforcement learning-based framework to solve scenario-based two-stage stochastic programming problems to reduce the solution times. In recent years reinforcement learning has been used to generate impressive results, including operations research. However, there is still a lack of novel methodologies to solve various types of operations research problems, including scenario-based stochastic programs. To the best of our knowledge, this is the first study that utilizes deep reinforcement learning to solve scenario-based two-stage stochastic programs with a stage-based learning strategy. We are motivated by the broad range of applications of two-stage problems and the promising results of reinforcement learning. Our methodology involves training two different learners for each stage of the problem, in which both learners are trained based on the actor-critic paradigm. Specifically, Agent 1 is utilized to solve the first stage of the problem while Agent 2 generates the second-stage solution. This provides high levels of flexibility with different solutions characteristics of both problem stages. Also, the trained model can be utilized to solve instances with a different number of scenarios and items. In

this framework, we are inspired by cutting plane algorithms in traditional solution approaches to solve scenario subproblems. Agent 2 is trained before Agent 1 to solve second-stage subproblems given first-stage decisions. We present a detailed training algorithm for Agent 2 with a scenario sampling approach to reduce the correlations during the training. Agent 1 is trained with the feedback of Agent 2 since the decisions are interconnected through the stages. This feedback is provided with a novel gradient calculation methodology. We present the detailed training strategy for Agent 1 and show how the actor and critic networks of Agent 2 are used during Agent 1 training. We show a comparison of our result with a commercial solver, state-of-the-art stochastic programming solution methodology, and heuristics. The results show that solution time can be reduced up to five orders of magnitude with sufficiently good optimality gaps of around 7%. Considering the vast state and action space of the problem of interest, the results show a promising direction for generating fast solutions without expert knowledge.

In Chapter 5, we address a very hard category of operations research problems known as scenario-based multi-stage stochastic programs. Our aim here is to propose a framework that can provide a major solution time reduction while maintaining a good quality solution. In this chapter, we propose a non-anticipative learning-based prediction-optimization framework for solving scenario-based multi-stage stochastic programs. In such problems, the property of non-anticipativity is crucial to ensure the implementability of decisions throughout stages. We address the complication of non-anticipativity by proposing a novel model architecture based on a neural machine translation system: Non-anticipative Encoder-Decoder with Attention. To the best of our knowledge, this is the first study that makes use of encoder-decoder models to solve scenario-based multi-stage stochastic programs by integrating learning, heuristics, and commercial solver. Also, we suggest training the models on single-scenario deterministic problems instead of stochastic programs, which would

be intractable to solve at large numbers for training purposes. We propose a framework for solving multi-stage stochastic problems by building on the framework presented in Chapter 3. We integrate our novel neural network that can handle non-anticipativity and a general linear programming-based heuristic approach. The presented framework is designed to tackle any multi-stage problem involving binary variables by integrating decisions made by learning models, heuristics, and commercial solvers. We present the results on two sequential combinatorial optimization problems under uncertainty: stochastic multi-item capacitated lot-sizing and stochastic multi-stage multi-dimensional knapsack. The results show that the proposed framework outperforms heuristics, and the solution time can be reduced with a factor of 599 with a gap of 0.08%. Furthermore, we present an improved item-wise expansion algorithm that considers prediction variability to solve a broad range of instances with a varying number of periods, items, and scenarios. Our non-anticipative learning-optimization approach can be used when similarly structured stochastic programming problems are solved repeatedly in a fast setting.

1.4 Organization of the Dissertation

The remainder of the dissertation is as follows. Chapter 2 presents the details of the LSTM-based framework. Chapter 3 describes the developed prediction-optimization framework to solve two fundamental combinatorial optimization problems based on encoder-decoder neural network architecture. Chapter 4 presents a new deep reinforcement learning approach for solving scenario-based two-stage stochastic programming problems. Chapter 5 introduces a study for solving scenario-based multi-stage stochastic problems based on a novel attention-based encoder-decoder neural network. Finally, Chapter 6 outlines the contributions and main findings together with promising future research directions.

CHAPTER 2

LEARNING OPTIMAL SOLUTIONS VIA AN LSTM-OPTIMIZATION FRAMEWORK

2.1 Introduction

In the recent decade, significant progress has been achieved with the use of machine learning (ML) in various fields, such as image recognition and natural language processing. A subfield of ML, deep learning, has inspired much success over the last decade and has led to a growing interest in research and practice. ML and operations research (OR) are historically interconnected through optimization, but only recently the use of ML for OR has received more attention. In this study, we will focus on this direction. Specifically, we will leverage deep learning algorithms to predict solutions to an OR problem by taking advantage of previously solved problems. In various applications in operations planning and management, such as energy demand-side management, airline scheduling, and vehicle routing, problems with the same structures must be solved repeatedly with different parameters within a very short period of time. In such settings, a reduced solution time obtained by fast algorithms can be highly beneficial for improving the efficiency and performance of businesses.

One complex and recurring problem for industrial companies is to determine the amount and timing of production over a planning horizon under resource constraints. It is an important challenge in industry and supply chain management because a production plan directly impacts companies' output and their ability to compete in operational costs and customer service levels (Gicquel et al., 2008). Production planning is also a highly complex task because firms strive to optimize multiple conflicting objectives, such as minimizing production and inventory costs, while maximizing customer satisfaction under tight constraints on resources, such as budget, raw materials, and machine availability.

In this study, we present a general prediction framework to learn optimal solutions of combinatorial optimization problems, while focusing on tackling one core production planning problem: the single-item Capacitated Lot-Sizing Problem (CLSP). The practical importance of the CLSP is apparent from numerous examples of its application in various production and manufacturing industries, including but not limited to the textile industry, oil and gas companies, car manufacturers, and pharmaceutical industry (Karimi et al., 2003; Gicquel et al., 2008). The CLSP determines the optimal production and inventory levels that meet periodic demand under a given production capacity by minimizing the sum of production, setup, and inventory holding cost over a finite planning horizon. In the mixed-integer programming (MIP) formulation of the CLSP, the decision of whether to produce or not is represented by a binary variable. Thus, the CLSP with time-varying capacity is NP-hard, a very difficult problem to optimize (Bitran and Yanasse, 1982; Hartman et al., 2010). In this research, we focus on tackling the computational difficulty of the CLSP and provide an ML-based optimization framework to solve its MIP formulation more efficiently. Thus, we study the CLSP at a high formulation level rather than focusing on a specific real-life application.

The CLSP is a sequential decision-making problem because, in each time period, the production level is determined to meet the periodic demand, and any additional produced items not used for current demand are placed in inventory to be used for future demand. Therefore, demand, capacity, cost inputs, and production decisions constitute highly correlated temporal sequences that the classical supervised classification might not capture. Thus, the CLSP can be treated as a sequence labeling task where a recurrent neural network (RNN) is applicable.

The RNN is a specialized type of neural network that can process sequential data by enabling information flow through various time steps. The neural network with the same parameters is applied at each time step of the sequence. Input to layer

at each time step consists of data at that time step and the network activations from the previous step. As a result, RNNs allow previous inputs to affect the output rather than just the current input. Developed by Hochreiter and Schmidhuber (1997), Long Short-Term Memory (LSTM) is a specialized RNN that can store information for long time steps, which can be challenging to handle by a classical RNN (Bengio et al., 1994). Bidirectional RNN (BRNN) allows using input information of future time steps rather than processing information in sequential order (Schuster and Paliwal, 1997). The main idea is to train two separate RNNs in both time directions that connect to the same output layer. Bidirectional LSTM is an extension of BRNNs by using LSTM architecture (Graves and Schmidhuber, 2005). The LSTM architecture might be preferable to the classical RNN due to its ability to capture long-term dependencies that come at a computational cost. We train the bidirectional LSTM network on datasets with different characteristics and evaluate the quality of resulting predictions in terms of feasibility and optimality. Our computational results show that a significant reduction in solution time can be achieved without much loss in feasibility and optimality.

Our main contribution is to develop an LSTM-based framework for learning optimal solutions to CLSP quickly. We propose using bidirectional LSTM to predict the binary production decision variable. Bidirectional LSTM can process information in both time directions, which is critical to predicting solutions to various OR problems with dynamic nature and where data is available for the planning horizon. Also, instead of using all the predicted variables, we propose using them partially to reduce the number of infeasible solutions. We present the results on how the quality of the predictions changes regarding feasibility and optimality with different levels of predicted variables. Additionally, we show the results of generalization on instances with different characteristics and instances with longer time horizons and compare

them with those of traditional dynamic programming and cutting plane algorithms and well-known learning algorithms: logistic regression and random forest.

2.2 Literature Review and Contributions

2.2.1 Literature Review

In recent years, significant results have been achieved in various fields by deep learning, which is a sub-field of ML. As a result, there has been a growing literature on the interaction between ML and OR. In this study, we focus on the use of ML to improve solving OR problems, particularly focusing on the CLSP. The origin of the interest in using ML algorithms for OR can be traced back to the 1980s when Neural Computation was used for solving Combinatorial Optimization problems (see, e.g., the survey of Smith (1999) on this topic). The approaches in the literature are structured into two parts: approaches that use ML for predicting the solutions directly from inputs and approaches that predict valuable pieces of information to utilize in the solution algorithms.

In one of the studies, which focuses on predicting the optimal solution directly, Larsen et al. (2022b) propose a new methodology to predict solution descriptions of a stochastic load planning problem using deep learning. According to the authors, a solution can be described at different levels. The most detailed solution describes the values taken by each variable, and the least detailed solution gives the value of the objective function. Their desired level of description is somewhere in the middle. At the time of the prediction, using a deterministic optimization model is not possible because the information available is imperfect, and the computational budget is limited. They generate training data by solving a large number of deterministic problems offline and combining solutions to the desired level of description at the prediction time. They train feedforward neural networks using this generated data and predict the actual problem instances. With a similar approach, Fischetti and Fraccaro (2019) use various ML techniques, including neural networks, to estimate

the optimal value of the offshore wind farm layout optimization problem. Their goal is to determine the optimal allocation of the wind turbines in a site to maximize park power production. The authors argue that ML can be used as a fast tool to estimate the optimal value of the problem for pre-selecting between candidate wind farm sites. The optimization model can be evaluated at these promising sites instead of all candidates. Based on their findings, a fast ML+OR tool can dramatically increase the number of sites and turbine types investigated.

In a recent study, Bertsimas and Stellato (2022) use neural networks to exploit the repetitive nature of online optimization, where problems with different parameters are solved frequently. They utilize the structure of mixed-integer quadratic optimization problems using neural networks to predict the strategy, which is defined as a tuple of indexes of tight inequality constraints and values of the integer variables. At the time of prediction, they do not require a solver. They evaluate a single neural network prediction and a single linear system solution.

Oroojlooyjadid et al. (2019) utilize deep neural networks to determine the optimal order quantity in the newsvendor problem. They establish an algorithm that integrates demand forecasting with deciding optimal order quantity rather than doing both separately. The input data consists of features of demand, and the output is the optimal order quantities. Additionally, they modify the loss function of the neural network as the newsvendor objective.

In the group of studies where ML is used to generate vital information to use in solution algorithms, Khalil et al. (2016) propose an ML framework for strong branching decisions, leading to significantly smaller search trees. In Khalil et al. (2017b), authors use ML to decide if a primal heuristic should be run at which nodes during the branch-and-bound tree search so that the overall performance of the solver is optimized. The reader is referred to the survey of Lodi and Zarpellon (2017) on learning algorithms to improve branch-and-bound decisions. Xavier et al. (2021)

propose the usage of ML algorithms to improve the computational performance of MIP solvers by predicting redundant constraints, reasonable initial feasible solutions, and affine subspaces where the optimal solution is likely to lie. Kruber et al. (2017) address whether or not a reformulation should be performed and which decomposition method to choose when several are possible using ML algorithms. Bonami et al. (2018) suggest a methodology that determines the linearization decision for a mixed-integer quadratic programming problem.

The CLSP has been widely studied in the OR literature by developing exact and heuristic algorithms. Florian et al. (1980) provide a solution methodology based on dynamic programming (DP) for lot-sizing. An exact solution approach presented by Barany et al. (1984) involves generating valid (ℓ,S) inequalities and adding them to the formulation with a separation algorithm. Eppen and Martin (1987) redefine variables to generate a graph representation of the problem which has a tighter linear relaxation than the original formulation. More recently, DP-based and partial-objective inequalities have been proposed for the single-item CLSP (Hartman et al., 2010) and multi-item CLSP (Büyüktahtakın et al., 2018b), respectively. For a detailed discussion of the exact and heuristic approaches to different versions of the lot-sizing problem, we refer the readers to the excellent review of Pochet and Wolsey (2006).

Readers are referred to Goodfellow et al. (2016) for a detailed discussion on deep learning algorithms. We refer to Graves (2012) for a detailed discussion on RNN, LSTM, and sequence labeling. Readers are referred to Karimi et al. (2003) and Pochet and Wolsey (2006) for an extensive survey on the capacitated lot-sizing problem, their variants, and exact and heuristic approaches for their solution.

There has been a growing interest in using ML algorithms to help solve OR problems in recent years. Despite all the advancements in the ML-OR integration, there is still a research gap in learning optimal solutions to MIP problems,

including CLSP from previously-solved instances and evaluating the effectiveness and generalization of the learning-based optimization approach.

2.2.2 Key Contributions of the Study

To our knowledge, none of the former studies have used a deep learning algorithm, such as LSTM, to capture the sequential nature of CLSP and predict their optimal solution. Decisions are closely linked over multiple periods in a multi-stage or sequential problem. Thus, an ML approach that does not consider patterns across time may not capture the dynamic nature of the problem. The LSTM, on the other hand, is a recurrent network capable of understanding long and short-term dependencies and temporal differences in the data of optimal solutions given specific problem characteristics.

In this study, we present a new deep learning LSTM-Optimization (LSTM-Opt) architecture to learn the optimal solutions for one of the most famous combinatorial optimization problems and a classic example of a sequential decision-making problem, CLSP. Our goal here is to reduce the solution time, where numerous similar CLSP need to be solved repetitively and in a fast manner with a small optimality gap. Our specific contributions are described next.

To our knowledge, this is the first study that utilizes an LSTM approach to make predictions from the optimal solutions of CLSP instances and use those predictions to solve similar CLSP with different data. Specifically, we propose an LSTM-Opt framework, which predicts binary decision variables of the CLSP problem. The bidirectional LSTM learns optimal solutions to sequential decision-making problems where the input data is available for the planning horizon. We compare the computational performance of our algorithm with other ML approaches, such as logistic regression and random forest. We show that the LSTM networks capture the

time-wise dependency in sequential decision-making and thus are superior compared to those ML algorithms.

We evaluate the effectiveness of predictions in terms of their feasibility and optimality for the original CLSP by defining optimization-based metrics, such as the optimality gap and the percent of feasibly-predicted instances in the test set. The use of all predictions could help reduce the solution time but also may increase the infeasibility in the test set. To improve the feasibility of the solutions, we propose using the predictions partially as an input into the MIP solver, CPLEX (IBM ILOG, CPLEX, 2016). This approach provides a significant reduction in solution time while improving the optimality gap and the feasibility of solutions. To remedy the infeasibility problem, additional methods, such as the CPLEX user cuts, are utilized to solve the problem with a reasonable optimality gap with no infeasibility.

We utilize benchmark CLSP instances in the literature to demonstrate the efficiency of our LSTM-Opt approach. In addition to comparing with direct solutions of CLSP by CPLEX, we utilize a dynamic programming formulation (Florian et al., 1980), dynamic programming-based inequalities (Hartman et al., 2010), and (ℓ,S) inequalities (Barany et al., 1984) to show that the LSTM-Opt can be beneficial to reduce the solution time even when compared with these traditional exact OR methodologies proposed for solving the CLSP more efficiently. Our LSTM-Opt framework helps decrease the CPLEX solution time by multiple orders of magnitude when predicting CLSP instances. Furthermore, this prediction architecture provides more time-gain benefits as the CLSP instances get harder, i.e., for the most difficult test problems that are generated with the same distribution as training instances, the solution time is reduced by a factor of 13 without any infeasibility or an optimality gap.

We investigate if the trained LSTM model can predict instances with different underlying data distributions or instances with a larger planning horizon. The results

imply that one must be careful in picking the prediction level to solve instances with different characteristics. The computational results also show that the trained LSTM model can successfully predict longer and, thus harder instances without extra training. As an example, in those generalization experiments to predict longer planning horizons, using a prediction level of 25%, we have reduced an average solution time of 70 CPU hours to only 2 CPU minutes with a 0.8% optimality gap, which is a quite significant computational achievement.

Once an LSTM model is trained from previously solved instances, predictions to new problems can be generated in milliseconds in an online setting. Thus, our LSTM-Opt approach could, in particular, be useful for solving practical and recurring sequential decision-making problems, such as power generation scheduling, energy demand-side management, and pricing optimization, where the same problem formulations are solved repeatedly over time with updated parameters. Our LSTM-Opt framework is generalizable since it does not assume any specific information about CLSP. Thus, it can be applied to other MIPs, such as the Binary Knapsack problem, one of the most well-known MIP formulations. The Binary Knapsack problem is also a relaxation of the CLSP.

The remainder of the study is as follows. Section 2.3 presents the MIP formulation of the CLSP. Section 2.4 describes the proposed LSTM-Opt framework. Section 2.5 describes the details of implementation and experimentation. Section 2.6 presents the computational results on datasets with different characteristics and a comparison with other ML and exact approaches. Section 2.7 concludes the chapter with future research directions. Appendix A.1-A.3 provides a discussion on the LSTM training time and more results with different datasets and characteristics, respectively.

2.3 Capacitated Lot-Sizing Problem

CLSP is a fundamental problem in production planning. The CLSP determines the production and inventory levels in a multi-period planning horizon to fulfill the deterministic demand without back-ordering to minimize the sum of production, setup, and inventory holding costs. The CLSP with time-varying capacity is NP-Hard, and it has numerous variations and applications in the production and manufacturing industries (Quadt and Kuhn, 2007).

To formulate the CLSP as an MIP, the following parameters and decision variables are defined. Let T be the number of periods considered in the planning horizon. For each period $t \in \{1, 2, \ldots, T\}$ demand d_t is known in advance. For each period $t \in \{1, 2, \ldots, T\}$ associated costs are unit production cost p_t, setup cost f_t, and unit inventory holding cost h_t. Note that setup cost f_t is not per unit based. For each period $t \in \{1, 2, \ldots, T\}$ production capacity is denoted by c_t. Without loss of generality, all parameters can be assumed to be non-negative. The number of units produced and ending inventory in period t is represented by non-negative variables x_t and s_t, respectively. Binary variable y_t takes value 1 if there is production in period t, and takes value 0 otherwise. The CLSP can be formulated as:

$$\min \sum_{t=1}^{T} (p_t x_t + f_t y_t + h_t s_t) \tag{2.1a}$$

$$\text{s.t. } s_{t-1} + x_t - d_t = s_t \qquad \forall t = 1, 2, \ldots, T \tag{2.1b}$$

$$x_t \leq y_t c_t \qquad \forall t = 1, 2, \ldots, T \tag{2.1c}$$

$$x_t, s_t \geq 0 \qquad \forall t = 1, 2, \ldots, T \tag{2.1d}$$

$$y_t \in \{0, 1\} \qquad \forall t = 1, 2, \ldots, T. \tag{2.1e}$$

The objective function (2.1a) minimizes the sum of production costs, setup costs, and inventory holding costs over all periods $t \in \{1, 2, \ldots, T\}$. Constraints (2.1b) ensure the inventory flow over multiple periods. Specifically, the demand in period t must be

satisfied by inventory at the end of period $t-1$ and units produced in period t. The remaining amount is the inventory at the end of period t. Constraints (2.1c) limit the production by capacity and ensure that a fixed cost of production is incurred in the objective function if there is production in period t. Constraints (2.1d) enforce that the amounts of units produced and kept in inventory are non-negative. Finally, constraints (2.1e) ensure that y_t are binary variables. The parameter s_0 represents the initial inventory and is assumed to be zero.

2.4 LSTM-Optimization Framework

In this section, we present the LSTM-Opt framework that we develop to predict the optimal solution of the CLSP. Using the LSTM-Opt framework, we only predict the binary decision variables y_t that correspond to a production decision instead of predicting all decision variables. As depicted in Figure 2.1, the LSTM-based framework starts with data generation. The datasets with different characteristics are constructed according to the data-generation scheme described in Section 2.5.1. The resulting datasets are divided into three categories involving the training, validation, and test sets, which consist of 64%, 16%, and 20% of the data, respectively. The LSTM network parameters are optimized using a training set. This is done by minimizing a loss function that measures the performance of the model's predictions compared to actual values. The binary cross-entropy is a common choice as a smooth loss function for binary classification because it leads to faster training with a better generalization performance than the sum of squares error (Bishop, 1995). The objective function of the CLSP given by Equation (2.1a) is not minimized by the LSTM network. In the training step, we minimize the binary cross-entropy loss function given in the following Equation (2.2):

$$\mathcal{L}(y^*, \hat{y}) = -\frac{1}{T} \times \sum_{t=1}^{T} (y_t^* \times log(\hat{y}_t) + (1 - y_t^*) \times log(1 - \hat{y}_t)) \tag{2.2}$$

where y^* represents the optimal values of the binary decision variables, and $\hat{y}$ represents predicted values of the binary decision variables of a CLSP instance. The binary cross-entropy loss function in Equation (2.2) measures the discrepancy between y^* and $\hat{y}$.

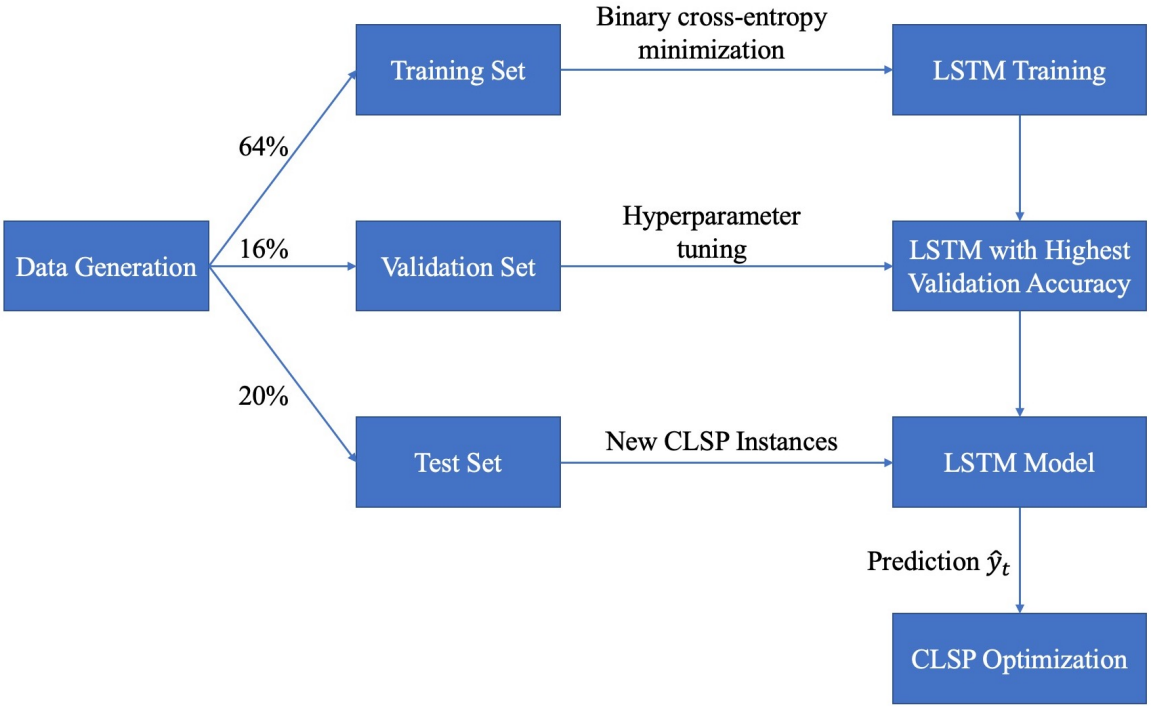


Figure 2.1 LSTM-Opt framework.

The LSTM model consists of several bidirectional LSTM layers that can process the information in both time directions and an output layer with a sigmoid activation function. Figure 2.2 shows the flow of information in the forward and backward layers in bidirectional LSTM. The input layer for LSTM consists of available features for that period: unit production cost p_t, setup cost f_t, production capacity c_t, and demand d_t for $t \in \{1, 2, \ldots, T\}$. Note that we omitted the holding cost h_t because it is taken as constant. For period t, information is carried from period $t-1$ in the forward layer and used to generate output in period t. In the backward layer, information is carried from period $t+1$ to period t, and it is used to generate output together with inputs in period t. The outputs of forward and backward layers are

combined to generate prediction $\hat{y}_t$. After each hidden layer, a dropout layer is added for regularization.

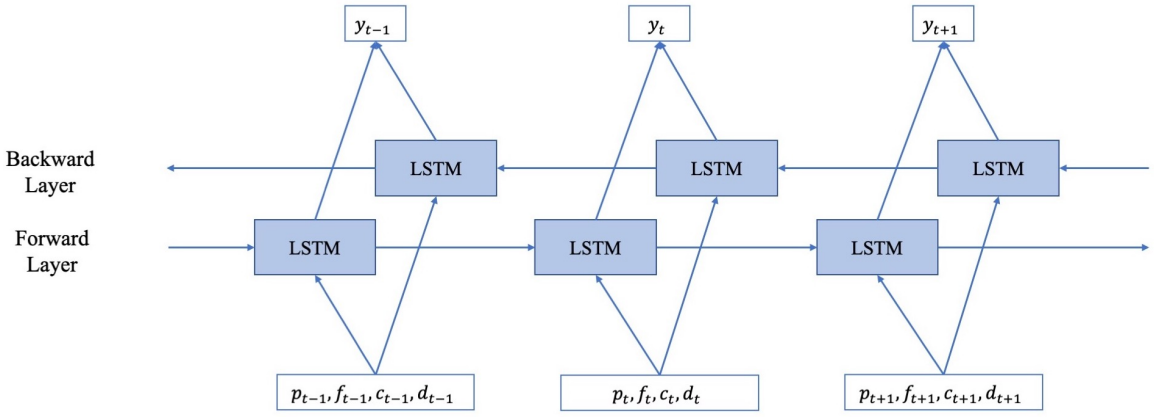


Figure 2.2 Bidirectional LSTM is adapted to represent the CLSP multi-period structure.

We compare the models with different parameters, using the instances in the validation set, in a method known as hyperparameter tuning. We then choose the model with the highest validation accuracy, which is the proportion of the correctly predicted variables. Note that the validation set is not used to minimize the binary cross-entropy in Equation (2.2); it is only used to compare LSTM networks with different hyperparameters, such as learning rate, number of layers, hidden nodes, and dropout rate. Then for each instance in the test set, a prediction is generated using the picked model. The framework described does not provide results on the feasibility of the resulting prediction and how good it is compared to the objective function value. The resulting predictions are added to problem (2.1) as constraints, and then CLSP is resolved using CPLEX. The described approach can deliver optimal solutions fast and accurately without much loss in feasibility and optimality, as demonstrated in the computational results under Section 2.6.

CLSP is an MIP because of the binary decision variables. Predicting all binary decision variables and then fixing the predicted values in the MIP formulation (2.1) makes the problem a linear program, which yields a significant reduction in the

solution time. This approach often leads to infeasibility due to its strict nature. Instead, predicting some of the binary decision variables results in more flexibility when resolving the problem instance and reduces the number of infeasible problems while still improving the solution time.

Additionally, the integral nature of the other two decision variables is preserved by solely predicting the binary variable because once the binary variables are fixed in the CLSP, it reduces to a linear program (Pochet and Wolsey, 2006). Also, predicting the binary variable carries an interpretable meaning of the production decision and its timing. Once the decision of whether to produce or not is determined and fixed in a period, the MIP solver determines the amount of production and the inventory levels.

Our integrated ML+OR tool can be beneficial for real-time applications where problems with different parameters are solved repeatedly. Lot-sizing and its variants commonly arise in the energy, pharmaceutical, electronics, food, processing, and consumer goods industries (Copil et al., 2017). After an ML model is trained, it is not necessary to update the trained model after each prediction. Therefore, once an ML model is trained, predictions can be achieved in milliseconds by an LSTM forward pass to solve many CLSP instances in a quite fast manner.

2.5 Implementation and Experimentation

This section presents the CLSP instance generation scheme and the implementation details of our LSTM-Opt framework. All the codes are written in C++ and Python to generate CLSP instances and run the LSTM-Opt framework.

2.5.1 CLSP Instance Generation

The training, validation, and testing data were generated by the scheme presented in Atamtürk and Muñoz (2004). The difficulty of problems was determined by two main factors: tightness of the capacities with respect to demand and the ratio between setup

and holding cost. Following the parameters used in Büyüktahtakın and Liu (2016), instances are generated from capacity-to-demand ratios $c \in \{3, 5, 8\}$, setup-to-holding cost ratios $f \in \{1,000, 10,000\}$ and the number of periods $T \in \{90, 120\}$. The parameters regarding demand d_t, unit production cost p_t, production capacity c_t, and setup cost f_t are generated from integer uniform distribution with the ranges $d_t \in [1, 600]$, $p_t \in [1, 5]$, $c_t \in \left[0.7c\bar{d}, 1.1c\bar{d}\right]$, $f_t \in \left[0.9f\bar{h}, 1.1f\bar{h}\right]$, where $\bar{d} = \frac{1}{T}\left(\sum_{t=1}^{T} d_t\right)$ and $\bar{h} = \frac{1}{T}\left(\sum_{t=1}^{T} h_t\right)$, respectively. Unit inventory holding cost h_t is set at one at each period.

For each of the 12 combinations of parameters c, f, and T as described above, 100,000 instances (problems) are generated, resulting in a total of 1,200,000 instances. All instances are solved using CPLEX. Infeasible problems are eliminated and replaced by feasible instances by regenerating new instances. The training, validation, and test set consists of 64,000, 16,000, and 20,000 CLSP instances, respectively, for each combination of parameters.

2.5.2 LSTM-Opt Implementation Details

Before the training, the data is standardized by subtracting the feature mean and dividing by the feature standard deviation as a preprocessing step, which is often practically useful for faster convergence if different inputs have typical values that differ significantly (LeCun et al., 2012). In the hyperparameter tuning step, we compared LSTM models with different parameters, such as learning rate, number of layers, hidden nodes, and dropout rate using the validation set. The values considered are $[2, 6]$ for the number of hidden layers, $[10, 150]$ for the number of units in hidden layers, $[0.1, 0.5]$ for the dropout rate, and $[0.1, 0.001]$ for the learning rate. The selected LSTM model contains three hidden LSTM layers, each with 40 hidden units in each time direction. Therefore, bidirectional LSTM for each layer has 80 hidden units. After each LSTM layer, a dropout layer with a drop rate of 0.3 is added to regularize

the network. We used Adam optimizer with an initial learning rate of 0.01, which is an adaptive learning rate optimization algorithm that has been shown to work well in practice (Kingma and Ba, 2014).

For each instance in the test set, the values of binary variables are predicted. For each period $t \in \{1, 2, \ldots, T\}$, the LSTM network generates a prediction in the range of $[0, 1]$ for each y_t. The value of $\max(\hat{y}_t, 1 - \hat{y}_t)$ for $t \in \{1, 2, \ldots, T\}$, where $\hat{y}_t$ represents the predicted value of the binary variable y_t, is calculated and ordered in decreasing order. Predicted variables are selected up to the desired level, and $\hat{y}_t$ is labeled as 0 or 1 using a cut-off value of 0.5. Those variables can be interpreted as the ones closest to either zero or one, and thus we are more confident in the LSTM model's prediction. Let $D \subseteq T$ be the set of indices of those binary decision variables predicted and selected using the $\max(\hat{y}_t, 1 - \hat{y}_t)$ function and a pre-set prediction percentage. Finally, those values are added as a constraint to the original model (2.1), as shown in the following modified problem (2.3):

$$\min \sum_{t=1}^{T} (p_t x_t + f_t y_t + h_t s_t) \tag{2.3a}$$

$$\text{s.t. } \mathit{Constraints} \quad (2.1\text{b}) - (2.1\text{e}) \tag{2.3b}$$

$$y_t = \hat{y}_t \qquad \forall t \in D. \tag{2.3c}$$

Problem (2.3) is solved again to assess the quality of the LSTM predictions.

2.6 Computational Results

This section presents results from computational experiments performed using the LSTM-Opt framework described in Section 2.4 on randomly generated CLSP instances with various characteristics, as defined in Section 2.5. All experiments are performed on a computer running Windows 10 Intel i7 with 3.6 GHz GPU and 64 GB of memory. The CLSP instances are solved with IBM ILOG CPLEX 12.7.1. All of

the results regarding the test-set solution times are presented in CPU seconds. The detailed results for training LSTM models are presented in Appendix A.1.

We solve problem (2.1) instances using the default CPLEX as a benchmark to compare the performance of our LSTM-Opt framework for solving similar and different CLSP instances. The test dataset consists of 20,000 CLSP instances for each combination of the parameters f, T, and c.

As an alternative to solving problem (2.3), where we fix the predicted values in the CLSP formulation (2.1) as constraints, we utilize two CPLEX solver methods– AddUserCuts and AddMIPStart methods of CPLEX to eliminate infeasibility as described below:

- **100(UC)**: AddUserCuts method of CPLEX, which enable CPLEX to add cuts into the user cut pool and use the cuts as needed.
- **100(MS)**: AddMIPStart method of CPLEX, which enable CPLEX to provide a starting solution to the model with a user cut pool.

2.6.1 Quality of Predictions

Here, we present a number of metrics with their formal definitions that are used to evaluate the effectiveness of our LSTM-Opt framework. Specifically, those metrics assess the proposed method to solve optimization instances with respect to the improvement in the solution time as well as the feasibility and optimality of the resulting solutions. The following metrics are used in Tables 2.1-2.6 and A.2-A.5:

- **timeCPX**: Mean solution time of a CLSP instance of the problem (2.1) in CPU seconds without any predictions using default CPLEX.
- **timeML**: Mean solution time of the LSTM-Opt framework, including the prediction generation time by the LSTM model.
- **pred(%)**: Percent of binary variables predicted by the LSTM-Opt framework.

Additionally, we provide the following metrics and their formal definitions and use a combination of them to present the results:

Definition 2.6.1 *The solution time factor improvement factor obtained by fixing the predicted variables as a constraint (or using AddUserCuts and addMIPStart) is given by:*

$$\textbf{timeimp} = \frac{timeCPX}{timeML}. \tag{2.4}$$

Definition 2.6.2 *The percent solution time gain obtained by fixing the predicted variables as a constraint (or using AddUserCuts and addMIPStart) is given by:*

$$\textbf{timegain}(\%) = \frac{(timeCPX - timeML)}{timeCPX} \times 100. \tag{2.5}$$

Definition 2.6.3 *The percent infeasibility of a test set resulted from using predicted binary variables is given by:*

$$\textbf{inf}(\%) = \frac{\hat{m}}{m} \times 100, \tag{2.6}$$

where $\hat{m}$ represents the number of CLSP instances that become infeasible by adding predictions as a constraint and m represents the total number of CLSP instances in the test set.

Definition 2.6.4 *Let (x^*, y^*, s^*) be the optimal solution for the original MIP problem (2.1) that is obtained by the CPLEX solver and $Z(x^*, y^*, s^*)$ be the corresponding optimal objective value. Let $\hat{y}$ be the partial or full prediction of binary variables, $(\tilde{x}, \tilde{y}, \tilde{s})$ be the optimal solution obtained by CPLEX using predictions $\hat{y}$ in problem (2.3), and $Z(\tilde{x}, \tilde{y}, \tilde{s})$ be the resulting objective function value. Note $\tilde{y}$ is equivalent to $\hat{y}$ when a full prediction is made. The optimality gap due to using solutions in our LSTM-Opt prediction framework is defined over feasibly-solved instances as follows:*

$$\textbf{optgap}(\%) = \frac{(Z(\tilde{x}, \tilde{y}, \tilde{s}) - Z(x^*, y^*, s^*))}{Z(x^*, y^*, s^*)} \times 100. \tag{2.7}$$

In the next section, we present computational results to demonstrate the effectiveness of our prediction-optimization method, using the training and test instances with the same distribution. The predictions fed into the CPLEX solver may not be feasible for the CLSP instance. The test set instances for which the

LSTM prediction leads to an infeasible solution are not included in the calculations of timeML, timeimp, timegain(%), and optgap(%).

2.6.2 Predicting Instances with Same Distribution

Each row of Tables 2.1, 2.3, A.2, A.3, A.4, and A.5 presents the averages of 20,000 instances, whereas Tables 2.4, 2.5, and 2.6 present the mean result for 10 instances due to long solution times for each c value and each $f-T$ pair. Table 2.1 presents results for instances with $T = 120$ with $f = 10,000$. The dataset of $c = 3$ is harder than the dataset with $c = 5$ and $c = 8$. Both 25% and 50% prediction levels achieve all-feasible predictions that do not increase the objective function value. With the 50% prediction level, the mean solution time decreases by more than 10-fold. As the prediction level increases, the time factor improvement increases as well. Predictions at the 75% level reduce the solution time by a factor of 50 with an infeasibility of the test set below 0.3% and without any optimality gap. However, as the prediction levels increases, predictions lead to more infeasible instances in the test set. At the full prediction level (pred(%)=100), more than half of the predictions result in infeasible solutions to problem (2.3); however, the issue of infeasibility is remedied by the CPLEX's user cuts approach (AddUserCuts), which eliminates the infeasible solutions in the cut pool. The user cuts (UC) approach provides a significant solution time factor improvement of 68 with an optimality gap of over 1% without any infeasibility in the test set. The detailed results of experiments with $f = 10,000$, $T = 90$ and $f = 1,000$, $T = 90, 120$ are presented in Tables A.2-A.4 in Appendix A.2.

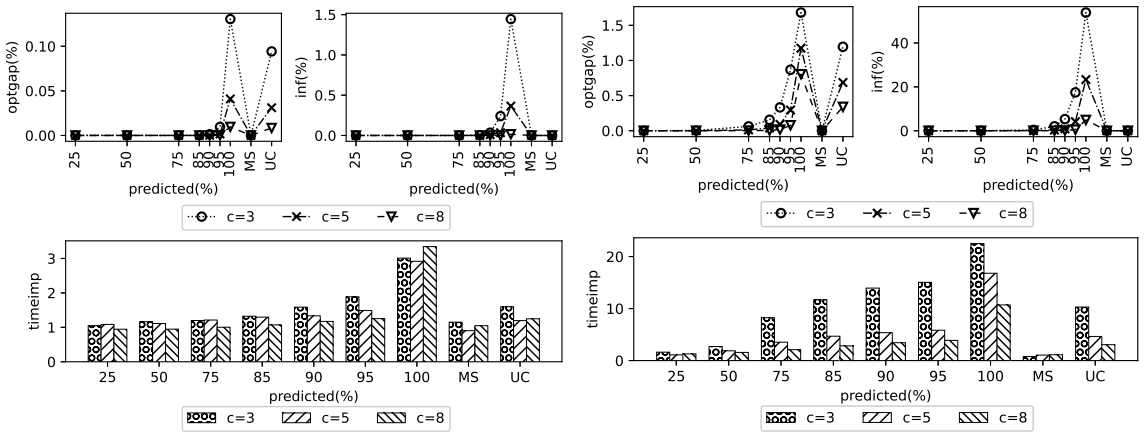
Figure 2.3a-2.3d summarize the results with the optgap(%), inf(%), and timeimp for changing c for instances with $T = 90, 120$ and $f = 1,000, 10,000$. Figure 2.3a-2.3d show that as the level of predicted variables increases, the time improvement also increases at the price of an increased optimality gap and infeasibility in the test set. The problems are harder when $c = 3$ ($f = 10,000$) compared to $c = 8$ ($f =$

Table 2.1 Summary of Experiments for $f = 10,000$ and $T = 120$

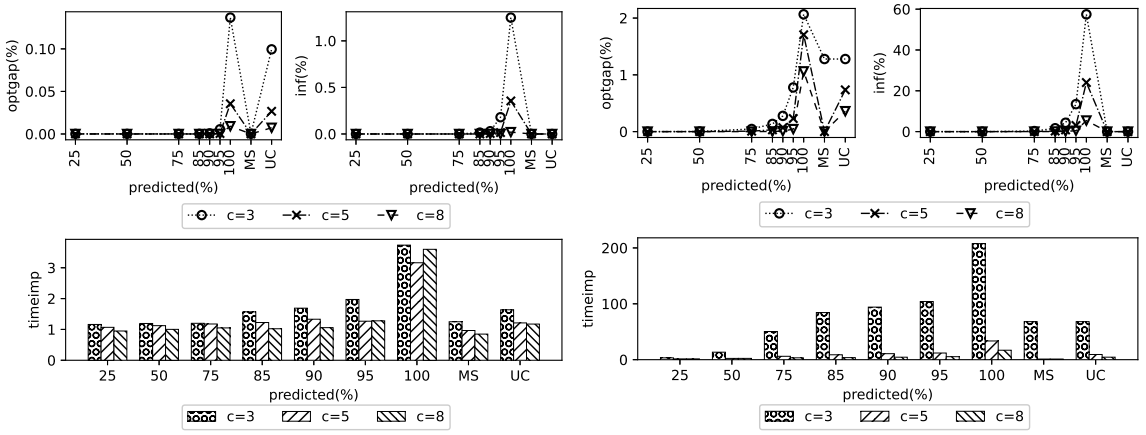
c	pred(%)	timeCPX	timeML	timeimp	timegain(%)	inf(%)	optgap(%)
3	25	22.6	6.9	3	69.3	0.0	0.0
	50		1.7	13	92.5	0.0	0.0
	75		0.4	50	98.0	0.3	0.0
	85		0.3	84	98.8	1.7	0.1
	90		0.2	94	98.9	4.3	0.3
	95		0.2	104	99.0	13.5	0.8
	100		0.1	208	99.5	57.5	2.1
	100(MS)		0.3	68	98.5	0.0	1.3
	100(UC)		0.3	68	98.5	0.0	1.3
5	25	3.0	2.1	1	28.7	0.0	0.0
	50		1.3	2	56.0	0.0	0.0
	75		0.5	6	83.5	0.0	0.0
	85		0.3	9	88.6	0.3	0.0
	90		0.3	11	90.7	0.8	0.1
	95		0.2	12	91.6	3.0	0.2
	100		0.1	33	97.0	24.0	1.7
	100(MS)		2.8	1	8.4	0.0	0.0
	100(UC)		0.3	9	89.2	0.0	0.7
8	25	1.4	1.0	1	29.4	0.0	0.0
	50		0.6	2	56.7	0.0	0.0
	75		0.4	3	69.9	0.0	0.0
	85		0.4	4	72.7	0.1	0.0
	90		0.3	4	77.5	0.1	0.0
	95		0.3	5	81.3	0.4	0.0
	100		0.1	17	94.1	5.4	1.1
	100(MS)		1.3	1	10.1	0.0	0.0
	100(UC)		0.3	4	77.6	0.0	0.4

1,000) for the same T. Predicting at lower levels provides good results for harder problems with significant time improvement without causing much optimality gap and infeasibility, e.g., a time improvement factor of 13 is achieved with the 50% prediction level without any infeasibility or optimality gap for instances with $c = 3$, $T = 120$, and $f = 10,000$ (Figure 2.3d). The time improvement factor is the highest when using the 100% prediction. For instances with $f = 1,000$, the full (100%) prediction results in less than a 1.5% infeasibility. However, it could provide high levels of infeasibility in the test set for harder problems with $f = 10,000$. On the other hand, the 50% prediction level provides over a time factor improvement of 3 and reduces the infeasibility to 0.01% and the optimality gap to zero (Figure 2.3b). When $f = 1,000$, time improvement increases significantly with the level of prediction, but the increase in the optimality gap and infeasibility is much less than the counterpart instances with $f = 10,000$, e.g., an infeasibility of 0.4% and optimality gap of zero is obtained for instances with $c = 5$, $T = 90$, and $f = 1,000$ compared to an infeasibility of 23.3% and optimality gap of 1.2% is observed for instances with $c = 5$, $T = 90$, and $f = 10,000$, using full predictions.

Figure 2.4a-2.4d show averages for changing c, f, and T, and the overall average. As the value of c increases, the optimality gap, infeasibility, and time improvement generally decrease (Figure 2.4a). Figure 2.4b shows that the value of f has a significant impact on results. The optimality gap and infeasibility are significantly lower when $f = 10,000$, with lower-level predictions. Also, the time factor improvement is significantly greater at all prediction levels when $f = 10,000$. Both the optimality gap and time improvement are slightly higher when $T = 120$ compared to the instances with $T = 90$. Figure 2.4d shows that using a prediction level of around 85% can balance all evaluation metrics by providing a solution time factor improvement of 9.



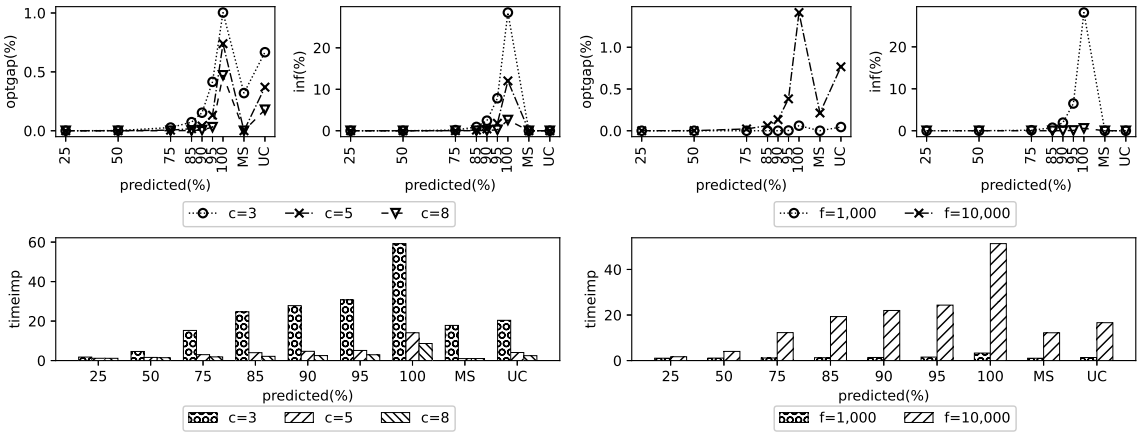
(a) $T = 90,\ f = 1,000.$ (b) $T = 90,\ f = 10,000.$



(c) $T = 120,\ f = 1,000.$ (d) $T = 120,\ f = 10,000.$

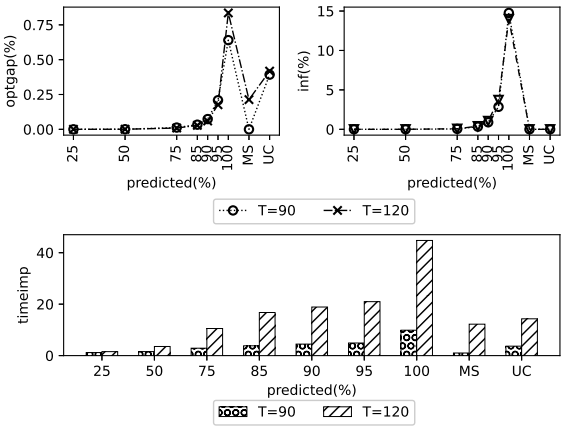
Figure 2.3 Summary of results with optgap(%), inf(%), and timeimp.

Table 2.2 shows the averages presented in Tables 2.1, A.2, A.3, and A.4 for the LSTM-Opt 85% prediction level, the 100(MS), and the 100(UC). When $f = 1,000$, both the average infeasibility and the optimality gap are zero with the 85% prediction level. The UC provides a similar average time improvement without infeasibility. When $f = 10,000$, the average time improvement is around 6 and 27 for the instances with $T = 90$ and $T = 120$, respectively, with the prediction level of 85%. The infeasibility is slightly higher than the instances with $f = 1,000$ since the higher-level predictions increase percent infeasibility for the harder test instances. The UC

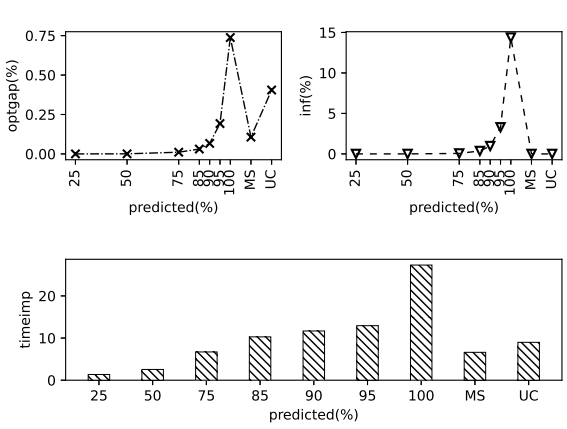


(a) Averages for c.

(b) Averages for f.



(c) Averages for T.



(d) Overall averages.

Figure 2.4 Summary of results with different data generation parameters.

remedies the infeasibility problem and improves the solution time with a factor of 6 and 28 and with an optimality gap of around 0.7% and 0.8% for the instances with $T = 90$ and $T = 120$, respectively. Also, the UC outperforms the MS in time gain for all cases. When looking at the overall averages in Table 2.2, the LSTM-Opt predictions at the 85% level reduce the CPLEX solution time by a factor of 9 on average for over 240,000 test instances with an infeasibility below 0.4% and an optimality gap of less than 0.05%. The UC provides a similar time gain without any infeasibility and a slightly higher optimality gap of 0.4% than the 85% level of prediction.

Table 2.2 Summary of Averages in Tables 2.1, A.2, A.3, and A.4

f	T	predicted(%)	timeCPX	timeML	timeimp	timegain(%)	infeasible(%)	optgap(%)
1,000	90	85	0.4	0.3	1	18.3	0.0	0.0
		100(MS)		0.3	1	3.1	0.0	0.0
		100(UC)		0.3	1	25.5	0.0	0.0
10,000	90	85	1.8	0.3	6	83.3	0.8	0.1
		100(MS)		2.1	1	-12.3	0.0	0.0
		100(UC)		0.3	6	83.2	0.0	0.7
1,000	120	85	0.4	0.3	1	21.0	0.0	0.0
		100(MS)		0.4	1	1.8	0.0	0.0
		100(UC)		0.3	1	25.4	0.0	0.0
10,000	120	85	8.8	0.3	27	96.2	0.7	0.1
		100(MS)		1.4	6	83.9	0.0	0.4
		100(UC)		0.3	28	96.4	0.0	0.8
	Avg.	85	2.8	0.3	9	54.7	0.4	0.0
		100(MS)		1.1	2	19.1	0.0	0.1
		100(UC)		0.3	9	57.6	0.0	0.4

In summary, the level of predictions used to get the best results varies notably between datasets with different characteristics. This level should be adjusted carefully considering the trade-off between time gain, infeasibility, and optimality gap. Using an appropriate level of predicted variables leads to major reductions in solution time up to an order of magnitude without increasing any infeasibility or optimality gap. It is beneficial to use lower prediction levels for harder instances and higher prediction levels for easier instances, but a prediction level of around 85% can be a reasonable level for all instances considered in this study. Also, the UC outperforms the MS in terms of providing lower optimality gaps. The UC can also be an alternative to the approach that uses predictions as constraints because it achieves zero infeasibility at the cost of a slightly higher optimality gap. As the c, f, and T increase, i.e., the instances get harder, and we observe a higher time factor improvement using the LSTM-Opt framework, highlighting the potential of our approach for solving instances with varying sizes and distributions, as discussed in the next section.

2.6.3 Results on Generalization

Here, we present the results on the generalization of our approach to instances with a larger planning horizon T. It is not uncommon to have long production planning periods for industries where daily (even hourly) production planning is necessary, such as large-scale semiconductor manufacturing, and energy production Uzsoy et al. (1992); Shrouf and Miragliotta (2015). Results on different data distributions are presented in Appendix A.3. We omit the results with the MS approach in favor of UC due to its lack of performance. Generalization is a desired property because it might be beneficial to train the LSTM model in a relatively small horizon to predict instances with a larger planning horizon, saving from the training time. Specifically, the time to train the LSTM model is shorter than the training time for the instance for which the prediction is made due to the smaller number of model parameters. We also compare our framework with two other well-known ML algorithms (logistic regression and random forests) and the state-of-the-art cutting plane algorithms proposed for the CLSP.

Predicting Instances with Longer Horizons Table 2.3 presents the results for predicting datasets with longer planning horizons. The predictions for a larger horizon are generated by concatenating the smaller LSTM predictions obtained by the LSTM model, which has a shorter planning horizon. For example, in the second block of rows in Table 2.3, the LSTM model with $c = 3$, $f = 1,000$, and $T = 90$ is used to generate predictions for the dataset with the same c and f, and $T = 360$. Here, four separate prediction sets, each with 90 periods, are concatenated into a single set of predictions for generating a prediction for the test set with $T = 360$.

For those instances, predicting 85% of variables results in a time improvement of 3, with a 0.3% optimality gap and zero infeasibility in the test set of 20,000 instances. The dataset with $c = 5$, $f = 10,000$, and $T = 180$ is predicted with the LSTM model

trained using instances with the same c and f but a half-length planning horizon of $T = 90$, as shown in the third block of rows in Table 2.3. For those instances, predicting 50% of variables yields a significant time improvement of 9 and all feasible solutions in the test set at the cost of an optimality gap, which is below 0.5%. The dataset with $c = 8$, $f = 10,000$, and $T = 480$ constitutes the hardest instances presented in Table 2.3 with the mean solution time over 40 seconds and is predicted using the LSTM model trained with $T = 120$. Here, we observe significant solution time factor improvements over CPLEX using our LSTM-Opt framework. Predictions at the 75% level reduce the CPLEX solution time by a factor of 25 with no infeasibility and an optimality gap of 1%.

Table 2.4 presents the results for predicting datasets with significantly longer planning horizons; therefore, the test instances are much harder than the training instances. The predictions are generated using the model trained with $c = 8$, $f = 10,000$, and $T = 120$. The test sets for all three datasets consist of 10 instances, instead of 20,000 as previously presented, due to computational complexity and long solution times. For the first dataset with $T = 600$, problems are solved 70 times faster than the default CPLEX using predictions at the 50% level without any infeasibility and with an optimality gap below 1%. The mean CPLEX solution time for the next dataset with $T = 720$ is more than 8 CPU hours. Here, the solution time of 8 hours is reduced to under 1 minute, with the predictions used at the 25% level without any infeasibility and with an optimality gap below 1%. Predictions used at 75% reduce the solution time by more than four orders of magnitude from more than 8 CPU hours to 2.5 CPU seconds without infeasibility and with an optimality gap below 2%. The last test dataset with $c = 8$, $f = 10,000$, and $T = 960$ constitutes the hardest instances presented with a mean solution time of over 70 hours using CPLEX at its default settings. For those instances, predictions at the 25% level reduce the solution time of 70 CPU hours to only 79 CPU seconds with an optimality gap of 0.8% and

Table 2.3 Summary of Generalization Experiments to Test Datasets with Longer Planning Horizons*

LSTM Train			Test Data			pred	timeCPX	timeML	timeimp	timegain	inf	optgap
c	f	T	c	f	T	(%)				(%)	(%)	(%)
3	1,000	90	3	1,000	180	25	0.5	0.5	1	7.6	0.0	0.1
						50		0.4	1	20.5	0.0	0.1
						75		0.4	1	29.1	0.0	0.2
						85		0.4	1	33.3	0.0	0.2
						90		0.3	2	38.2	0.0	0.2
						95		0.3	2	44.2	0.1	0.2
						100		0.1	6	82.4	1.0	0.4
						100(UC)		0.3	2	36.2	0.0	0.4
3	1,000	90	3	1,000	360	25	1.2	1.0	1	19.7	0.0	0.1
						50		0.7	2	41.6	0.0	0.2
						75		0.5	2	59.6	0.0	0.2
						85		0.4	3	64.2	0.0	0.3
						90		0.4	3	66.4	0.0	0.3
						95		0.3	3	71.4	0.2	0.3
						100		0.1	12	91.9	1.3	0.5
						100(UC)		0.4	3	62.8	0.0	0.5
5	10,000	90	5	10,000	180	25	19.0	6.2	3	67.2	0.0	0.3
						50		2.2	9	88.4	0.0	0.5
						75		0.7	28	96.4	0.1	0.5
						85		0.4	46	97.8	0.3	0.6
						90		0.3	59	98.3	0.9	0.7
						95		0.2	78	98.7	2.8	0.9
						100		0.1	166	99.4	27.5	3.5
						100(UC)		0.4	49	98.0	0.0	1.6
8	10,000	120	8	10,000	480	25	42.4	11.6	4	72.7	0.0	0.6
						50		4.3	10	89.9	0.0	0.9
						75		1.7	25	96.1	0.0	0.9
						85		0.8	55	98.2	0.0	0.9
						90		0.5	86	98.8	0.1	1.0
						95		0.4	112	99.1	0.4	1.0
						100		0.1	364	99.7	5.2	2.8
						100(UC)		0.6	71	98.6	0.0	1.5

*Experiments include 20,000 test instances.

without any infeasibility. Predictions at the 50% level reduce the solution time by more than a factor of 16,000, with an optimality gap of 1.2% and zero infeasibility.

Table 2.4 Summary of Generalization Experiments to Test Datasets with Longer Planning Horizons Continued*

LSTM Train			Test Data			pred	timeCPX	timeML	timeimp	timegain	inf	optgap
c	f	T	c	f	T	(%)				(%)	(%)	(%)
8	10,000	120	8	10,000	600	25	409	31.1	13	92.4	0.0	0.6
						50		5.8	70	98.6	0.0	0.9
						75		2.5	161	99.4	0.0	1.2
						85		1.2	343	99.7	0.0	1.5
						90		0.9	476	99.8	0.0	1.7
						95		0.6	712	99.9	0.0	2.0
						100		0.2	1,990	99.9	0.0	3.0
						100(UC)		6.0	68	98.5	0.0	1.6
8	10,000	120	8	10,000	720	25	30,038	54.5	552	99.8	0.0	0.9
						50		7.2	4,168	100.0	0.0	1.2
						75		2.5	12,014	100.0	0.0	1.7
						85		1.0	28,888	100.0	0.0	2.1
						90		0.8	38,788	100.0	0.0	2.5
						95		0.6	51,267	100.0	0.0	3.0
						100		0.2	164,035	100.0	10.0	6.4
						100(UC)		9.7	3,150	100.0	1.4	2.5
8	10,000	120	8	10,000	960	25	252,186	78.6	3,208	100.0	0.0	0.8
						50		15.2	16,543	100.0	0.0	1.2
						75		3.1	80,922	100.0	0.0	1.6
						85		1.3	199,593	100.0	0.0	2.0
						90		0.8	310,612	100.0	0.0	2.3
						95		0.6	411,262	100.0	0.0	2.7
						100		0.2	1,245,361	100.0	0.0	5.6
						100(UC)		14.3	17,678	100.0	0.0	2.3

*Experiments only include ten test instances due to long solution times.

Generating 100,000 instances for training data with $c = 8$, $f = 10,000$, and $T = 120$ takes 140,170 seconds whereas the LSTM training time takes 52,724 seconds. In this specific example, it can be concluded that generating training data, training the LSTM model, and resolving with predictions for a single instance takes 16 hours

less than solving the instance with CPLEX. For such hard problems, our LSTM-Opt framework achieves a significant time reduction even in the case where just a single instance must be solved. The results discussed above highlight that our approach could be generalizable to predict larger instances with substantial benefits in reducing the solution time of those hard CLSPs with the cost of a small optimality gap.

2.6.4 Comparison with Other ML and Exact Algorithms

Table 2.5 presents the computational comparison of our LSTM-Opt framework with two other machine learning approaches that perform a binary classification task and shows that their prediction quality is not comparable to LSTM-Opt. Additionally, the comparison with two other exact approaches is presented in Table 2.6 to show that the LSTM-Opt framework could produce good-quality solutions in much less time compared to those exact approaches. The machine learning and exact approaches used to compare with the LSTM-Opt are defined as follows.

ML Approaches:

- Logistic Regression (LR): An extension of linear regression, which is more interpretable than the tree-based ensemble methods such as random forest at the cost of accuracy.
- Random Forest (RF): One of the best algorithms for classification tasks (Fernández-Delgado et al., 2014) in terms of classification accuracy at the cost of reduced interpretability.

Exact Approaches:

- CPLEX (CPX): Direct solution of the CLSP formulation (2.1a)-(2.1e) with default CPLEX.
- Dynamic programming-based inequalities (DPineq) of Hartman et al. (2010): We used the weaklu strategy to create a tighter CLSP polyhedron. The generated inequalities are added to the formulation (2.1), and the proposed algorithm is shown to outpace the dynamic programming algorithm by Hartman et al. (2010) for some cases. In the experiments, we generate cuts for the first 100 periods with $c = 3$, for the first 75 periods with $c = 5$, and for the first 50 periods with $c = 8$ for instances with $T = 360$ to combat the growing DP-based inequality generation time with increasing c.

- The (ℓ, S) inequalities (LSineq) of Barany et al. (1984): Implemented with a separation algorithm since the number of (ℓ, S) inequalities grow exponentially. The separation algorithm is iterated five times which is inclined to give the best computational achievements (Büyüktahtakın et al., 2018b).

- Dynamic programming (DP) solution approach for CLSP (Hartman et al., 2010; Florian et al., 1980): Results are omitted from the tables due to lack of performance. For example, while the mean solution times of $c = 3, 5, 8$ instances with CPLEX were 22.4, 3.3, and 1.3 seconds, respectively, the dynamic programming solution times were 610.7, 1054.9, and 1938.8 seconds for the same first 20 instances presented in Table 2.1. Additionally, the dynamic programming approach has a complexity of $\mathcal{O}(TD_T^2)$ where $D_T = \sum_{t=1}^{T} d_t$. We do not further include the dynamic programming solution to compare with the LSTM-Opt framework since CPLEX is superior for the considered instances.

Table 2.5 presents a set of instances that are tested for the comparison of LSTM-Opt, LR, and RF. Here, we have utilized a different structure to generate our test instances. For the instances with $c = 3, 5$, a solution time limit of 86,400 CPU seconds (24 CPU hours) is set for CPLEX to restrict the solution time. The metrics, including time improvement, time gain, and optimality gap, are calculated based on the best solution found by CPLEX within the solution time limit. The $IGap = 100 \times (objCPX - objLP)/objCPX$, where $objCPX$ is the objective function value of the best feasible solution to the original problem and $objLP$ is the objective function to its linear programming relaxation, is 7.5%, 16.1%, and 27.7% for the instances with $c = 3, 5$, and 8, respectively. CPLEX reports an MIP optimality gap of 0.64%, 0.06%, and 0.00% on average for test instances with $c = 3, 5$, and 8, respectively, with a one-day time limit. For the same instances with $c = 3, 5$, our preliminary results revealed that the test problems were still computationally very expensive to solve without a time limit with the 25% prediction level. Therefore, the results with the 25% prediction level are omitted from the results in this section.

In Table 2.5, for the $c = 3$ instances, predictions at the 50% level improve the solution time by more than a factor of 7,500 by reducing the limited average solution time from one day to only 12 seconds, without any infeasibility and with an optimality

gap of 0.8%. Predictions of more than 50% lead to some infeasibility in the test set, while the predictions at the 100% level lead to all infeasible predictions, and thus the corresponding results are presented in a "-" in the first-row block of results in Table 2.5. For the same instances solved at the 50% prediction level, LR and RF have caused an infeasibility of 70% and 50%, respectively, since neither considers sequential dependency like the LSTM networks. The optimality gaps of the feasible instances were significantly higher than the 0.8% of the LSTM-Opt framework at 1.9% and 2.3%, respectively, for both LR and RF. Also, the time improvements are not as big as the ones of the LSTM-Opt framework. For the instances with $c = 5$, predictions using LSTM-Opt at the 50% level decrease the limited solution time from 1-day to 20 CPU seconds and reduce the solution time by more than a factor of 4,000, including the prediction generation time without any infeasibility and with an optimality gap of 0.9%. The 85% level with LSTM-Opt reduces the solution time by five orders of magnitude without infeasibility and with an optimality gap of 2.3%. For the same prediction level, both the LR ad RF causes all infeasible predictions. The datasets with $c = 8$ constitute significantly faster to solve compared to instances with $c = 3, 5$, and the results resemble the structure with easier instances. The LR and RF cause high infeasibility in all prediction levels compared to the LSTM-Opt framework. Table 2.5 shows that a significant solution time reduction of around four to five orders of magnitude is achieved by LSTM-Opt without any infeasibility, depending on the desired optimality gap. We anticipate that the solution time gains would have further increased if the original solution time was not limited to 24 hours. Additionally, both the LR and RF cause higher infeasibility, a higher optimality gap, and a lower time improvement.

Table 2.6 presents a comparison of LSTM-Opt at the 50% prediction level with two exact approaches, namely (ℓ, S) valid inequalities (LSineq) and DP-based cutting planes (DPineq). The solution time is limited by 1 hour, including the inequality

generation time for both approaches. Here, the time improvement metric has been calculated with respect to the set solution time of 1-hour. The optimality gap metric has been calculated with respect to the best integer solution found by the CPLEX within a 1-day solution time limit. For the $c = 3$ instances, the LSTM-Opt framework achieves a 1.6% optimality gap. The objective function value is reduced below the 1-day limited CPLEX solution value with a 1-hour time limit in both formulations with DP-based and (ℓ, S) inequalities, resulting in a negative optimality gap of -0.01%. Therefore, both types of inequalities are effective at reducing the optimality gap, but they cannot solve the CLSP very fast though they speed up the solution for harder instances of $c = 3, 5$. Even though the LSTM-Opt framework has an optimality gap of 0.8%, the solution was found 322 times faster than CPLEX, showing that the LSTM-Opt can solve those problems very fast. The results with $c = 5$ show a similar pattern, and the LSTM-Opt framework has an optimality gap of 0.9% on top of the CPLEX gap. The results with $c = 8$ show that the LSTM-Opt framework can achieve a time improvement of 3 while inequality-based methods increase the solution time for easier instances.

Figure 2.5a and 2.5b present a summary of the results for the instances presented in Tables 2.5 and 2.6, for $c = 3$ and 5, respectively. Both DPineq and LSineq improve over the CPLEX gap in Figure 2.5a with $c = 3$ instances. LSTM achieves a solution with a slightly larger optimality gap much faster than those exact inequality methods and also is faster than LR and RF with a lower gap. The latter two algorithms result in high infeasibility of 70% and 50%, respectively, while LSTM-Opt achieves all-feasible predictions. The results show similarity for the $c = 5$ instances in Figure 2.5b, with the exception that both exact methods do not improve over the CPLEX gap or solution time. Even though LR and RF have a higher time improvement than LSTM-Opt, both lead to high and unpractical infeasibility rates of 60% and 40%, respectively.

Table 2.5 Computational Results for Comparing LSTM-Opt with Other Machine Learning Algorithms*

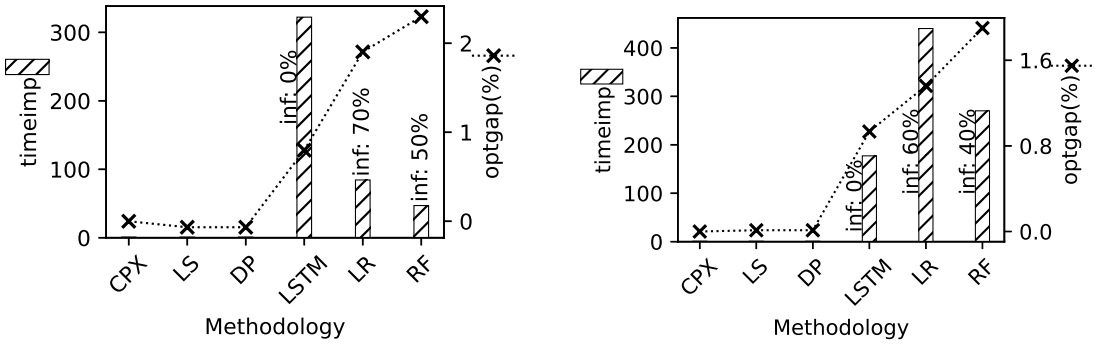
		Train	Test	pred	timeCPX	time	timeimp	inf(%)	optgap(%)	time	timeimp	inf(%)	optgap(%)	time	timeimp	inf(%)	optgap(%)
c	f	T	T	(%)		LSTM-Opt				LR				RF			
3	10,000	90	360	50	86,415	11.3	7,675	0.0	0.8	42.7	2,024	70.0	1.9	76.6	1,129	50.0	2.3
				75		1.4	63,189	10.0	2.0	0.6	146,459	90.0	17.3	-	-	100.0	-
				85		0.6	139,042	20.0	2.9	-	-	100.0	-	-	-	100.0	-
				90		0.5	159,647	30.0	3.6	-	-	100.0	-	-	-	100.0	-
				95		0.4	193,403	50.0	4.5	-	-	100.0	-	-	-	100.0	-
				100		-	-	100.0	-	-	-	100.0	-	-	-	100.0	-
5	10,000	90	360	50	86,763	20.4	4,253	0.0	0.9	8.2	10,552	60.0	1.4	13.4	6,491	40.0	1.9
				75		2.5	34,371	0.0	1.6	0.7	124,883	80.0	25.8	0.6	145,415	90.0	24.2
				85		0.8	102,727	0.0	2.3	-	-	100.0	-	-	-	100.0	-
				90		0.7	118,189	0.0	2.7	-	-	100.0	-	-	-	100.0	-
				95		0.5	190,865	10.0	3.3	-	-	100.0	-	-	-	100.0	-
				100		0.2	416,680	60.0	14.2	-	-	100.0	-	-	-	100.0	-
8	10,000	90	360	50	7.5	2.6	3	0.0	1.6	3.2	2	40.0	0.5	3.1	2	30.0	1.0
				75		1.0	8	10.0	1.9	0.7	8	80.0	14.3	0.7	8	90.0	22.5
				85		0.7	11	10.0	2.2	0.4	14	90.0	111.1	-	-	100.0	-
				90		0.6	13	20.0	2.4	-	-	100.0	-	-	-	100.0	-
				95		0.5	16	20.0	2.8	-	-	100.0	-	-	-	100.0	-
				100		0.2	38	20.0	3.8	-	-	100.0	-	-	-	100.0	-

*Experiments only include ten test instances due to long solution times.

Table 2.6 Computational Results for Comparing LSTM-Opt with Different Exact Methods*

		Train	Test	pred	inf	*time*				*timeimp*			*optgap*(%)			
c	f	T	T		(%)	CPX	Dpineq	LSineq	LSTM-Opt	Dpineq	LSineq	LSTM-Opt	CPX	Dpineq	LSineq	LSTM-Opt
3	10,000	90	360	50	0.0	3,602	3,609	3,602	11.2	1	1	322	0.03	-0.1	-0.1	0.8
5	10,000	90	360	50	0.0	3,602	3,421	3,600	20.3	1	1	177	0.05	0.0	0.0	0.9
8	10,000	90	360	50	0.0	7.5	720	146	2.5	0	0	3	0.00	0.0	0.0	1.6

*Experiments only include ten test instances and limited with 1-hour time limit due to long solution times.



(a) Instances with $c = 3$, $f = 10,000$, and $T = 360$.

(b) Instances with $c = 5$, $f = 10,000$, and $T = 360$.

Figure 2.5 Comparison of exact and ML algorithms.

2.6.5 Summary of Results

The results presented on generalization experiments show that a network trained on a smaller planning horizon can be used to successfully predict the optimal solutions of the instances with larger horizons without any additional training. The solution time can be reduced up to six orders of magnitude without increasing the optimality gap or infeasibility much, especially in harder problems. Also, LSTM-Opt can capture sequential dependencies while LR and RF cannot. Classical exact approaches cannot produce very fast solutions like the LSTM-Opt.

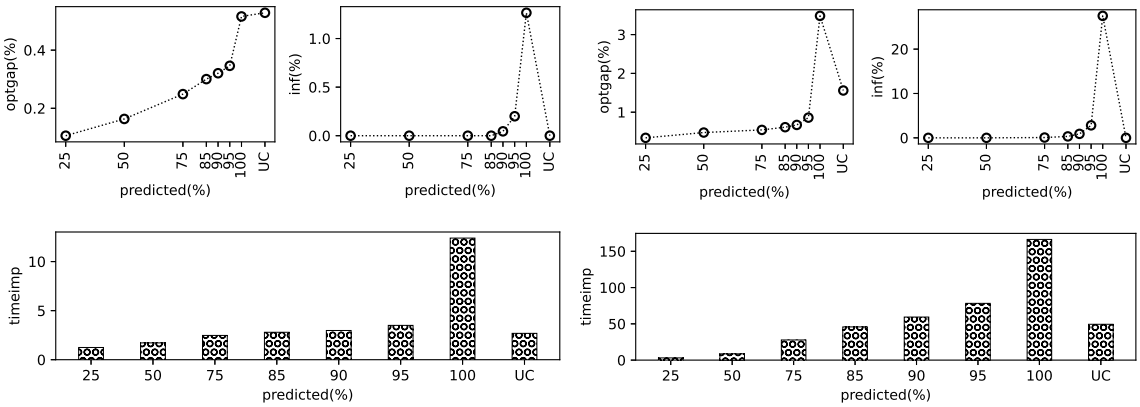
Figure 2.6a-2.6f present the results for datasets for longer planning horizons. The results for $T = 360$ in Figure 2.6a are similar to dataset where LSTM model is trained with $c = 3$, $f = 1,000$, and $T = 90$, as shown in Figure 2.3a. For the dataset with $c = 8$, $f = 10,000$, and $T = 720$ in Figure 2.6c, using predictions at the 50% level reduces the solution time from more than 8 hours to under 8 seconds without any infeasibility and with an optimality gap of 1.2%. Figure 2.6d constitutes the instances with the longest solution times. The instances with $c = 8$, $f = 10,000$, and $T = 960$ have a mean solution time over 70 hours. Predictions at the 25% and 50% levels reduce the solution time of those instances by more than a factor of 3,000

and 16,000 with an optimality gap of 0.8% and 1.2%, without any infeasibility in the test set, respectively.

Overall averages in Table 2.4 show that the solution time can be decreased with a factor of more than 9,000, with an infeasibility in the test set of only 0.5% and an optimality gap of approximately 2.1%. The UC reduces the solution time by more than a factor of 90,000 on average without infeasibility in the test set and an average optimality gap of around 3.5%. Overall, predictions at the levels between 25% and 85% provide significant time improvements with less than a 1% optimality gap and without any infeasibility in the test set. Specifically, predictions at the 85% level can balance a high solution time factor improvement with infeasibility and optimality gap at reasonable levels when predicting for longer periods using the LSTM model trained with instances of shorter and, thus easier instances.

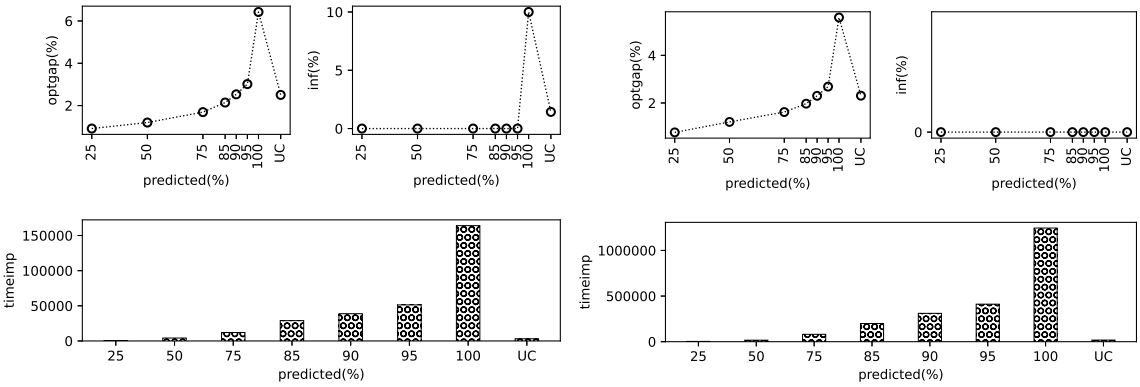
2.7 Conclusions and Future Work

In this study, we present a new LSTM-Opt framework to predict the optimal solution of the CLSP, a fundamental production planning problem in various industry settings. Our ML approach could be beneficial in reducing the solution time for many practical problems that are solved repeatedly with different parameters. We utilize bidirectional LSTMs to process information in both time directions. The metrics, defined as time factor improvement, infeasibility, and optimality gap, are presented to assess the quality of the predictions. The results for the CLSP instances with the same characteristics show that a time factor improvement of more than an order of magnitude can be achieved without much loss in feasibility or the optimality gap if the level of predictions used to solve the problem is well adjusted. Also, we tested if the trained LSTM models could generalize to instances with different data distributions or longer planning horizons. The results show that one should be careful in selecting the prediction level for predicting instances with different data distributions. The LSTM models trained on shorter planning horizons achieve great success in predicting



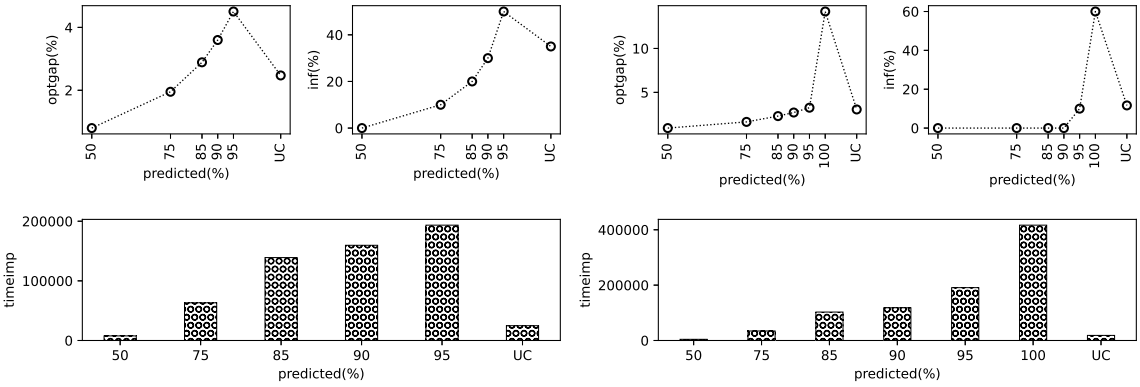
(a) LSTM trained with $c = 3$, $f = 1,000$, $T = 90$ predicts $c = 3$, $f = 1,000$, $T = 360$.

(b) LSTM trained with $c = 5$, $f = 10,000$, $T = 90$ predicts $c = 5$, $f = 10,000$, $T = 180$.



(c) LSTM trained with $c = 8$, $f = 10,000$, $T = 120$ predicts $c = 8$, $f = 10,000$, $T = 720$.

(d) LSTM trained with $c = 8$, $f = 10,000$, $T = 120$ predicts $c = 8$, $f = 10,000$, $T = 960$.



(e) LSTM trained with $c = 3$, $f = 10,000$, $T = 90$ predicts $c = 3$, $f = 10,000$, $T = 360$.

(f) LSTM trained with $c = 5$, $f = 10,000$, $T = 90$ predicts $c = 5$, $f = 10,000$, $T = 360$.

Figure 2.6 Summary of generalization experiments.

instances with longer planning horizons with any prediction level and reducing the solution time up to six orders of magnitude with a small optimality gap. Specifically, we observe the highest computational benefit from our LSTM-Opt approach when predicting the hardest set of instances. Also, the LSTM-Opt framework outperforms classical ML algorithms in terms of the quality of the solution and exact approaches with respect to the solution time improvement.

Our LSTM-Opt framework can be especially useful for reducing the solution time of dynamic combinatorial optimization problems that are solved in a repetitive setting. In this study, we have used the CLSP as a specific case to show that deep learning approaches have great potential for learning optimal solutions to MIP problems. Future research could further investigate the generalizability of our approach to instances with a larger planning horizon and different distributions in more detail. Another possible research direction is to develop methods to eliminate infeasible predictions. Additionally, the developed LSTM-Opt framework can be extended to solve more complex versions of the CLSP, such as the multi-item or multi-level CLSP, as well as other sequential decision-making problems.

CHAPTER 3
AN EXPANDABLE LEARNING-OPTIMIZATION FRAMEWORK FOR SEQUENTIALLY DEPENDENT DECISION-MAKING

3.1 Introduction

The goal of this study is to contribute to bridging the gap between two traditionally distinct research areas, Operations research (OR) and Machine learning (ML), to solve NP-hard sequential decision-making problems. OR is a discipline that aims to find the best decisions for complex problems through mathematical modeling and optimization, while ML focuses on learning from the data without explicitly programming it. In this study, we tackle a very hard category of OR problems known as combinatorial optimization problems by innovatively combining a machine translation learning framework with an optimization-learning framework.

Our objective is to substantially reduce the solution times of such combinatorial problems while providing high-quality feasible solutions, which could be very useful in practical applications. In most industrial settings, such as finance, health, energy, and manufacturing systems, OR problems with the same structures are solved repeatedly with different parameters. For such settings, a reduced solution time can provide an immense advantage to decision-makers. In this study, we present an expandable framework based on a sequence-to-sequence neural machine translation system to solve sequentially dependent optimization problems with all feasible predictions, which are either optimal or very close to optimal.

In Chapter 2, we present one of the pioneering studies that utilize an ML-based prediction methodology to reduce the solution times of repeatedly-solved combinatorial problems in a multi-period setting. Specifically, in Chapter 2, we harness bidirectional Long Short-Term Memory (LSTM) networks to predict binary decision variables that denote the production decision in the capacitated lot-sizing problem. Models are trained using the solutions of the problems that are solved to

optimality using CPLEX. Then, predictions are generated using the trained model for new instances and added to the problem as constraints to be resolved using CPLEX. The results show that depending on the hardness of the problem, solution time can be decreased by up to six orders of magnitude without a significant increase in the infeasibility or optimality gap, especially when problems with longer planning horizons are considered. In Chapter 2, predictions for problems with longer planning horizons are generated by concatenating the predictions generated by models trained using problems with shorter planning horizons, e.g., predictions for 270-period problems are generated as three independent prediction sets first from 1 to 90, second from 91 to 180, and third from 181 to 270 using the model that trained 90 periods with. However, this approach disregards the dependence between consecutive sets, e.g., prediction sets 1 to 90 and 91 to 180 are generated independently, which may impact the quality of the predictions. Also, it does not provide a methodology to accommodate problems with periods that are not exact multiples of the training periods. In this study, we integrate the attention-based encoder-decoder network presented in Luong et al. (2015) into a prediction-optimization framework to overcome those sequential dependence problems since the output sequence is allowed to be of variable size.

Furthermore, in Chapter 2, we show that the percentage of predicted binary variables that are added to the problem can have a major impact on the quality of the solutions. In other words, using high-level predictions may lead to high infeasibility since the predicted values are fixed in the problem. In Chapter 2, we state that fixing all predictions of binary variables may cause infeasibility in one of the two problems in some cases, which is highly undesirable. We propose an iterative algorithm to eliminate infeasible predictions to overcome this issue. Another important research question has been the generalizability of such ML approaches to predict optimal solutions. For example, could a smaller dimensional model with a few items be used to predict a higher-dimensional problem with a larger set of items? In this chapter,

we develop new algorithms to tackle those main research challenges in utilizing ML approaches to predict optimal solutions to NP-Hard problems.

In practice, combinatorial optimization problems are solved frequently with changing input data. In this study, we focus on solving two core NP-Hard problems where decisions are made over a time horizon: Multi-item Capacitated Lot-Sizing Problem (MCLSP) and Multi-stage Multi-dimensional Knapsack Problem (MSMK). In MCLSP, sequentially-dependent decisions are made to determine the production and inventory levels for the planning horizon, considering the changing costs, demand, and capacity. The lot-sizing is one of the most important and difficult problems in production planning. Its variants have a wide range of applications in numerous fields, including production, medical and chemical industries (Karimi et al., 2003). Another example of dynamic combinatorial optimization problems is when the stability over the solution can possess significant importance since it might incur setup costs frequently, i.e., turning on and off an electricity plant can be costly. In such settings, maintaining the stability of the current solution for successive periods is desired. Examples of such problems include MSMK, the core resource allocation problem with stability constraints (Bampis et al., 2022), the multi-stage facility location problem (Eisenstat et al., 2014), and multi-stage prize-collecting traveling salesperson (Bampis et al., 2020).

Our main contribution is to develop an extendable prediction-optimization (PredOpt) framework for sequential decision-making problems to address the key issues of sequential dependence, infeasibility, and generalization in ML prediction for OR. We have innovatively adapted a local attention-based encoder-decoder network, a deep learning tool originally developed for neural machine translation, to learn the optimal solutions for sequentially dependent optimization problems. The sequential nature of the considered combinatorial optimization problems is captured with recurrent neural networks and a sliding-attention window. The models can be trained

on short-period problems to learn long-period problems. Furthermore, we present a specific prediction algorithm that enables the trained models with smaller items to be generalized to predict instances with much larger items. Additionally, we develop an iterative algorithm for quickly checking the feasibility of predicted solutions and determining the optimal level of predictions. The resulting predictions practically eliminate the possibility of infeasible solutions resulting from predictions. We demonstrate our general framework to tackle MCLSP and MSMK without assuming any specific details of the problem and, therefore, without tailoring it to a specific problem. We show the computational efficiency of the PredOpt framework and the quality of the predictions in terms of the solution time reduction and optimality gap metrics.

3.2 Literature Review and Contributions

The use of ML for OR has recently gained much interest with successful ML applications in different problems. The closest study to ours is by Frejinger and Larsen (2019). They use a similar approach in which an attention-based encoder-decoder neural network architecture of Bahdanau et al. (2014) was used to predict solutions fast to a combinatorial optimization problem under imperfect information. They demonstrate creating input and output vocabularies to represent the optimization problem as a pair of input and output languages. Our study differs significantly from Frejinger and Larsen (2019) in several key points. First, our focus is on predicting the binary variables solely to use in the Mixed-integer Linear Programming (MILP) solver to reduce the solution time of the problem. We achieve this objective with a methodology to determine the highest level of prediction that will not cause infeasible predictions. On the other hand, Frejinger and Larsen (2019) have predicted only tactical decisions since fully detailed solutions are not needed at the time of prediction. Therefore, the type and the usage of decision variable predictions made are different.

Although Frejinger and Larsen (2019) present results on generalizability to different instances, we specifically focus on training using shorter and smaller-dimension instances to predict longer and higher-dimension, therefore harder problems. We also propose a specially designed item-wise expansion algorithm for generalization and present detailed computational results to demonstrate the generalization capability of our PredOpt framework. Additionally, Frejinger and Larsen (2019) utilize the neural machine translation model presented by Bahdanau et al. (2014), whereas we employ the architecture presented in Luong et al. (2015) with local attention to capturing the sequential dependencies. In another study, Larsen et al. (2022b) focus on solving the same optimization problem as Frejinger and Larsen (2019) with a similar motivation. Rather than a language translation system, they use multilayer perceptrons to predict the tactical descriptions of operational solutions in a less detailed aggregation compared to Frejinger and Larsen (2019).

In a recent study, Bertsimas and Stellato (2021) use optimal classification trees to learn insights into the solution strategies of optimization problems. They built their methodology by defining the strategy, consisting of the values of decision variables and active constraints. In other words, the strategy is the complete information that can be used to generate the solution by creating a smaller problem. They predict the three most-likely strategies. Even though their approach to defining the strategy and predicting it to use in the solution shows similarities with our study, several key differences exist in the overall frameworks presented. First, they use machine learning to identify the most probable several strategies to directly use in the solution and use all predictions of discrete variables with only tight constraints. In contrast, our framework predicts a single strategy to determine the prediction level required to eliminate the infeasibility of the predictions. Then we consider all constraints, not just the predicted tight constraints, with partial use of predictions of discrete variables when retrieving the solution. Additionally, in our study, the tightness of the

constraints is defined differently, dissimilar ML algorithms are used, and the focus on generalization is distinct compared to Bertsimas and Stellato (2021). The results show that fast solutions can be achieved with low infeasibility or suboptimality. Bertsimas and Stellato (2022) build upon that framework to solve parametric mixed-integer quadratic optimization problems without requiring a solver. They obtain the solution with a KKT-based linear system solution. They also define strategy pruning to reduce the number of strategies.

Anderson et al. (2022) present a generative neural network design to reduce solution times of repetitively solved optimization problems. The presented framework contains two neural networks: A generator to predict the values of binary decision variables and a discriminator to predict the objective function value when those variables are fixed. Although their motivation is similar to ours, the methodology they developed is quite different from ours. They utilize a model based on generative adversarial networks, whereas we employ an encoder-decoder network. While both studies predict binary decision variables' values, their usages are dissimilar. Anderson et al. (2022) feed predictions to a discriminator to obtain a prediction of the objective when those variables are fixed in the problem and then use them as a warm start for the optimizer. We utilize predictions of binary variables in a feasibility check loop to determine the optimal prediction level and solve the problem with the fixed variables at a determined level. They show their framework for the transient gas optimization problem, and they are able to reduce the solution time by 60.5%. We demonstrate our prediction framework for solving MCLSP and MSMK.

Zamzam and Baker (2020) utilize neural networks to learn from the optimal solutions of an AC optimal power flow problem. They also emphasize the feasibility of the generated predictions. They train the model with only feasible solutions and restrict the output of the network to satisfy the generation and voltage limits. While this approach secures the feasibility of the predicted variables, power flow equations

are solved to secure the feasibility of the overall problem. The solution time can be reduced up to a factor of 16 with a small infeasibility and optimality gap. In Pan et al. (2019), a neural network-based framework is presented to handle the infeasibility of the predictions when solving the optimal power flow problem. They ensure feasibility by adjusting the limits of the constraints during training. Another methodology to eliminate infeasibility is presented in Donti et al. (2021). They achieve this by enforcing the constraints during training for the AC optimal power flow problem. They predict variables partially and then solve for the remaining variables to ensure that the constraints are satisfied. Guha et al. (2019) present a methodology to learn solutions to the AC optimal power flow problem. In addition to predicting optimal solutions, they predict the active constraints so that they can be used together in a warm start with predictions.

Masti and Bemporad (2019) present a framework to learn the binary variables for multi-parametric mixed-integer quadratic programming problems that are usually solved with branch and bound methods. Their learner is classical artificial neural networks due to their small computational footprint. In order to reduce the infeasibility of the predictions, they add a compartment punishing infeasibility to the loss function used during training. They also propose a new branch and bound design to eliminate infeasible predictions in solution time. They initially start by fixing all binary variables to their predictions and recursively unfix variables until a feasible solution is found. Their solution approach can be used to find a global optimum with fewer computations or improve the quality of the solutions with a fixed number of iterations. Even though their study shares the same ideas as us as predicting binary variables and iteratively eliminating predictions, the methodologies presented in both studies are significantly different.

In Bengio et al. (2020), the authors present a framework to find near-optimal solutions to two-stage stochastic integer programs. Their framework tries to find a

representative scenario to aggregate all scenarios so that the solution of the two-stage problem with the representative scenario is the same as the solution of the original two-stage problem. The authors note that the first stage's feasibility is ensured since the learning algorithm does not directly predict the solution. They demonstrate their framework through a two-stage stochastic capacitated facility location problem by using linear regression and artificial neural networks as their learners.

Vinyals et al. (2015b) introduce pointer networks to solve combinatorial optimization problems such as the traveling salesperson and planar convex hull problem. The encoder-decoder architecture uses an attention mechanism as a pointer to select an input element as output at each decoding time step. The proposed architecture is suitable for combinatorial optimization problems where the size of the output depends on the size of the input sequence. Even though their seminal work shares the same motivation as ours, our problem of interest is to have a time-wise extendable framework with a constant-sized output rather than a mapping to the input elements to handle variable-sized output when decoding.

In a recent study, Bello et al. (2016) present a methodology to use reinforcement learning as an alternative to the pointer networks presented in Vinyals et al. (2015b), where supervised learning is not desired. The proposed framework can achieve solutions close to optimal when solving the traveling salesperson problem. Nazari et al. (2018) generalize the framework presented in Bello et al. (2016) to handle more complex problems in that system dynamics change over time. The authors show that the learned policy can achieve near-optimal solutions for the capacitated vehicle routing problem.

Kool et al. (2018) present a new model based on attention and a methodology for training. They follow a transformer architecture with multi-head attention instead of the encoder-decoder with recurrent neural networks. Their attention-based model

focuses on learning heuristics for solving the traveling salesperson and vehicle routing problem and achieving a performance level close to highly-specialized algorithms.

3.2.1 Key Contributions of the Study

While there has been a significant advancement in the use of ML for OR in recent years, as discussed above, an extensive research gap exists in the literature. In particular, frameworks that utilize time-dependent learning models and algorithms to enforce the feasibility and integration of advanced computation capabilities of OR solvers with promising results of ML can be further investigated to solve hard mathematical programs. Those research limitations inspire our study. Our motivation is to develop new algorithms that innovatively adapt neural translation deep learning architectures to predict optimal solutions to NP-hard optimization problems. Our goal is to reduce the solution times while ensuring feasible and near-optimal solutions where problems with similar structures are repeatedly solved. Our key contributions are explained next.

To our knowledge, this is the first study to explore an encoder-decoder approach to predict optimal solutions to sequential decision-making problems by integrating with a mathematical solver. Specifically, we have designed the PredOpt framework based on an encoder-decoder with an attention mechanism to capture the dynamic relationship between input parameters and optimal solutions to MCLSP and MSMK problems. Our machine learning approach involves a local attention structure with a time window to better capture the association among the problems' periods since the current prediction period is more closely related to the preceding and succeeding several periods than the entire sequence. This reduces the computational cost compared to the global attention mechanism as in the classical LSTM and enables selectively focusing on a few close-by periods near the decision point, which better suits the sequentially-dependent problems.

The presented PredOpt framework with an encoder-decoder mechanism learns from shorter problems to solve much longer ones. Further, we have developed a new generalization algorithm to create predictions abstracted from small problems to be used for larger problems. Specifically, we have shown that the models trained are generalizable by training them with a few items to predict problems with a large number of items. This approach results in a significant time gain in training set generation and training time. This is also critical because once a model with a smaller dimension is trained, it can be used for solving numerous problems with larger dimensions.

We tackle the problem of infeasibility in ML predictions by proposing a new iterative methodology to find feasible predictions and a favorable prediction level that decreases the solution time. This algorithm includes predicting tight constraints, which in turn is used to create a relaxed problem to check the feasibility quickly. We then utilize a second iteration to ensure that the prediction level determined by the relaxation is updated if necessary.

We generate benchmark MCLSP and MSMK instances and compare the computational performance of the PredOpt framework with the state-of-the-art commercial solver CPLEX version 20.1.0 and heuristics in terms of the optimality gap and solution time. Our results show that the solution time can be improved up to a factor of 7,236 with an optimality gap of only 0.11% on average while ensuring that predictions used to obtain solutions are all feasible. The presented PredOpt framework can be quite beneficial for applications where problems with similar structures are solved repeatedly, which are common in various industries from manufacturing to electronics, energy and healthcare systems, and the public good (Finnah et al., 2022; Büyüktahtakın et al., 2018a; Yin et al., 2023; Bushaj et al., 2022a).

The remainder of the chapter is as follows. Section 3.3 presents the formulations for MCLSP and MSMK. Section 3.4 presents the encoder-decoder model and the PredOpt framework. Section 3.5 explains the implementation steps, experimentation environment, and the metrics used to measure the quality of the PredOpt framework. Section 3.6 presents the results obtained by using the PredOpt framework. Section 3.7 concludes the chapter with future directions.

3.3 Problems

In this section, we present the problem formulations of the two problems of specific interest in this study: The Multi-item Capacitated Lot-Sizing Problem (MCLSP) and the Multi-stage Multi-dimensional Knapsack Problem (MSMK).

3.3.1 Multi-item Capacitated Lot-Sizing Problem

MCLSP is an extension of the single-item CLSP, where multiple items compete for a shared capacity at each time period in a production planning setting. MCLSP decides on the production and inventory amount for each item at each period by minimizing the sum of costs, which includes production, setup, and inventory costs. The demand, which is known in advance, is satisfied for each item and period pair if possible, and back-ordering is not allowed. MCLSP is NP-Hard (Bitran and Yanasse, 1982), and it has variations that include setup times, pricing decisions, lost sales, shortage costs, safety stocks, and demand uncertainty that is used in production and manufacturing industries (Maes and Wassenhove, 1988).

MCLSP is formulated as a mixed-integer program (MIP), where T is the number of periods in the planning horizon, and I is the number of items considered. For each item $i \in \{1, \ldots, I\}$ and period $t \in \{1, \ldots, T\}$ pair, problem parameters are demand d_{it}, unit production cost p_{it}, setup cost f_{it}, and unit inventory holding cost h_{it}. Production capacity c_t is the total capacity available for each period $t \in \{1, 2, \ldots, T\}$. All parameters of the MCLSP are assumed to be non-negative. The decision variables

x_{it} and s_{it} denote the number of units produced and the ending inventory of item i at period t, respectively. The decision to produce is binary y_{it}, which is set to be 1 if the item i is produced at period t and 0 otherwise. The MCLSP formulation:

$$\min \sum_{i=1}^{I}\sum_{t=1}^{T}(p_{it}x_{it} + f_{it}y_{it} + h_{it}s_{it}) \tag{3.1a}$$

$$\text{s.t. } s_{i,t-1} + x_{it} - d_{it} = s_{it} \qquad \forall i = 1,\ldots,I, \quad \forall t = 1,\ldots,T \tag{3.1b}$$

$$\sum_{i=1}^{I} x_{it} \leq c_t \qquad \forall t = 1,\ldots,T \tag{3.1c}$$

$$x_{it} \leq y_{it}c_t \qquad \forall i = 1,\ldots,I, \quad \forall t = 1,\ldots,T \tag{3.1d}$$

$$x_{it}, s_{it} \geq 0 \qquad \forall i = 1,\ldots,I, \quad \forall t = 1,\ldots,T \tag{3.1e}$$

$$y_{it} \in \{0,1\} \qquad \forall i = 1,\ldots,I, \quad \forall t = 1,\ldots,T. \tag{3.1f}$$

The sum of production, setup, and holding costs is minimized in the objective function (3.1a) over each item $i \in \{1,\ldots,I\}$ and period $t \in \{1,\ldots,T\}$. The flow of inventory is established with constraints (3.1b). Specifically, the demand in period t for item i is fulfilled with the inventory at the end of period $t-1$ and units produced in period t, and the remaining units are set to be the inventory at the end of period t. Constraints (3.1c) ensure that the sum of items produced of all types is limited by capacity for each period t. Constraints (3.1d) assert related setup cost if the item i is produced in period t. Finally, constraints (3.1e) enforce non-negativity, and constraints (3.1f) ensure y_{it} are binary. The parameter s_{i0} represents the initial inventory for item t and is assumed to be zero.

Both single and multi-item versions of the lot-sizing problem have been widely studied in the literature. For example, Florian et al. (1980) provide an exact solution approach based on dynamic programming. Another exact solution approach is developed by Barany et al. (1984), where valid (ℓ,S) inequalities are added to the problem using a separation algorithm. In recent years, inequalities based on dynamic

programming and partial-objective inequalities were proposed to solve the multi-item capacitated lot-sizing problem (Hartman et al., 2010; Büyüktahtakın et al., 2018b). We refer to the excellent review by Pochet and Wolsey (2006) for exact and heuristic solution methodologies and discussion on the different versions and modifications of the lot-sizing problem. In this study, we utilize the valid (ℓ,S) inequalities presented by Barany et al. (1984) with a strategy presented by Büyüktahtakın et al. (2018b) to show that our PredOpt framework performs well even when compared to hand-crafted special solution algorithms. Also, relax-and-fix heuristics are commonly used to solve MCLSP and its variations (Helber and Sahling, 2010; Toledo et al., 2015; Absi and van den Heuvel, 2019; Pochet and Wolsey, 2006). Absi and van den Heuvel (2019) present the famous relax-and-fix heuristic for MCLSP, which we adopt to compare with our framework.

3.3.2 Multi-stage Multi-dimensional Knapsack Problem

MSMK is a dynamic version of the classical knapsack problem where the profit and constraints vary over time. In the multi-stage multi-dimensional knapsack problem, the aim is to find stable solutions over the planning horizon that maximizes profit and satisfies the capacity constraints. In Bampis et al. (2022), the authors state that even the multi-stage single-dimensional knapsack is strongly NP-Hard when T is not fixed. Potential applications involve energy production planning and data center operations where the problem parameters, such as prices, energy, raw materials, and resources, change frequently.

MSMK can be formulated as an integer program (IP) where the number of periods considered is denoted by T, and the number of items available for the knapsack is denoted by I. For each period $t \in \{1, \ldots, T\}$, binary variable x_{it} takes the value 1 if item $i \in \{1, \ldots, I\}$ is added to the knapsack and takes the value 0 otherwise. In order to maintain the stability of the solution at each time stage, a binary variable

y_{it} is defined for each item $i \in \{1, \ldots, I\}$ and period $t \in \{1, \ldots, T-1\}$. The variable y_{it} takes value 1 if items i's decision was unchanged from period t to $t+1$, i.e. $x_{it}, x_{i,t+1} = 0$, or $x_{it}, x_{i,t+1} = 1$, otherwise it takes value 0 if the decision is changed from period t to $t+1$, i.e. $x_{it} = 0, x_{i,t+1} = 1$ or $x_{it} = 1, x_{i,t+1} = 0$. The profit of each item i at period t is denoted by p_{it}, and the bonus for stability is denoted by b_{it}. The number of available knapsack constraints is denoted by J, and the capacity is set to be c_{jt} for each resource constraint $j \in \{1, \ldots, J\}$, and period $t \in \{1, \ldots, T\}$. The weight is denoted by w_{ijt} for each item $i \in \{1, \ldots, I\}$, resource constraint $j \in \{1, \ldots, J\}$, and period $t \in \{1, \ldots, T\}$. We adapt the formulation in Bampis et al. (2022) to include multiple resource constraints and formulate the MSMK as:

$$\max \sum_{i=1}^{I} \sum_{t=1}^{T} p_{it} x_{it} + \sum_{i=1}^{I} \sum_{t=1}^{T-1} b_{it} y_{it} \tag{3.2a}$$

$$\text{s.t.} \sum_{i=1}^{I} w_{ijt} x_{it} \leq c_{jt} \qquad \forall t = 1, \ldots, T, \quad \forall j = 1, \ldots, J \tag{3.2b}$$

$$y_{it} \leq -x_{i,t+1} + x_{it} + 1 \qquad \forall i = 1, \ldots, I, \quad \forall t = 1, \ldots, T-1 \tag{3.2c}$$

$$y_{it} \leq x_{i,t+1} - x_{it} + 1 \qquad \forall i = 1, \ldots, I, \quad \forall t = 1, \ldots, T-1 \tag{3.2d}$$

$$x_{it} \in \{0, 1\} \qquad \forall i = 1, \ldots, I, \quad \forall t = 1, \ldots, T \tag{3.2e}$$

$$y_{it} \in \{0, 1\} \qquad \forall i = 1, \ldots, I, \quad \forall t = 1, \ldots, T-1. \tag{3.2f}$$

The sum of profit and stability bonus is maximized in the objective function (3.2a). Constraints (3.2b) ensure that for each knapsack j and time period t, the total weight of selected items is less than the capacity c_{jt}. Constraints (3.2c) and (3.2d) determine the association between x_{it}, $x_{i,t+1}$, and y_{it} variables. Specifically, those constraints are linear relaxations of $y_{it} = 1 - |x_{i,t+1} - x_{it}|$. Constraints (3.2e) and (3.2f) enforce that x_{it} and y_{it} are binary variables.

Solution approaches for traditional multi-dimensional knapsack problems include exact algorithms and heuristic or metaheuristic algorithms, which we refer to

the review of Varnamkhasti (2012) for further details. Bertsimas and Demir (2002) present an algorithm named adaptive fixing heuristic to solve the general multi-dimensional knapsack problem. Since their heuristic approach achieves good quality solutions (Wilbaut et al., 2008), we adopt the heuristic presented by Bertsimas and Demir (2002) to generate a benchmark solution for MSMK.

3.4 Methodology

In this section, we first discuss the machine translation model used to learn the optimal solutions to optimization problems. Here, the encoder-decoder model with attention adapted from Luong et al. (2015) is presented with modifications made. Then, the developed PredOpt framework is introduced. We then present an algorithm for generalization with item-wise expansion.

3.4.1 Neural Machine Translation and Adaptation

A machine translation system is used to translate the input sequence $x_1, x_2, \ldots, x_m$ from the source language to output sequence $y_1, y_2, \ldots, y_n$ to the target language, where m and n give the size of the input and the output sequences, respectively. For optimization problems that we are tackling, the size of the input and the output sequences is the same, i.e., $m = n$. For MCLSP, the input sequence consists of d_{it}, p_{it}, f_{it}, h_{it}, c_t, and the output sequence is y_{it} and an $(I+1)T$-dimensional binary vector that defines tight constraints. For MSMK, the input sequence consists of p_{it}, b_{it}, w_{ijt}, c_{jt}, and the output sequence is x_{it} and a JT-dimensional binary vector labeling tight constraints.

The main idea of a neural machine translation system is to utilize neural networks to fit a parameterized model to maximize the probability conditioned upon input sequence and past output elements: $P(y \mid x) = \prod_{t=1}^{n} P(y_t \mid y_{i|i<t}, x)$. We refer to Stahlberg (2020) for a detailed review of neural machine translation approaches. While our neural machine translation architecture is similar to that of Luong et al.

(2015), we modify their attention mechanism to accommodate the requirements of the combinatorial optimization problem better. At each prediction time step, the hidden states of the top forward and backward LSTM layers are concatenated as presented in Bahdanau et al. (2014) and used to calculate the attention scores as presented in Luong et al. (2015). This is different than the approach presented by Luong et al. (2015), where they use the hidden states from the top hidden layer to calculate the attention scores.

The encoder is a recurrent neural network that reads the input sequence one at a time. In the frame of neural machine translation, at each time step, the encoder reads a word from the input sequence $x_1, x_2, \ldots, x_m$. For the encoder, we used bidirectional LSTM networks, which ideally capture the representation of the input sequence by processing with the forward and backward layers. The hidden states of both forward and backward LSTMs are used to calculate the attention score and the context to predict the next work in the decoder.

The decoder is also a recurrent neural network that produces the output sequence. At each time step, the decoder input is the prediction from the previous time step. The attention mechanism enables the neural machine translation model to use the information from the encoder hidden states when decoding by determining which parts of the encoder hidden states relate more to the current decoder hidden state. It is a mechanism that enables focusing more on the selected parts of the input sequence. This is done by calculating the so-called context, which is a weighted average of the hidden states of the encoder. The context is used together with the decoder hidden state to output the prediction for each time step.

The encoder-decoder network with attention is appropriate to make predictions for sequentially dependent optimization problems for various reasons. First and foremost, it can capture the time-wise dependent relationship between the decisions made throughout the problem's horizon. Even though the size of decisions made

is constant for each period of the optimization problem, treating these sequential decisions as independent for each period by using a classical machine learning model may lead to a poorer performance compared to recurrent models, as demonstrated in Chapter 2. Using a sequence-to-sequence mapping model ensures that the knowledge is transferred through the problem's planning horizon. Secondly, recurrence-based attention models have performed highly well in various tasks, including neural machine translation (Bahdanau et al., 2014), image captioning (Mnih et al., 2014), and speech recognition (Chorowski et al., 2015). In Chapter 2, we use classical LSTMs in predicting the optimal solutions. However, a performance increase in the predictive power of the model can lead to less infeasibility, a smaller optimality gap, and increased solution time improvement. Therefore, by using a more advanced model, the solutions to harder and more complex problems can be learned. Additionally, with the use of the decoder, the decisions made in the previous period are directly considered when predicting for the next period, which is not the case in regular LSTMs. Furthermore, the attention structure enables to focus on not just the current prediction period's hidden states, but it rather enables to selectively focus on all periods' hidden states, which is a desirable feature in sequential prediction tasks.

The building blocks of the encoder-decoder model are LSTMs that are developed by Hochreiter and Schmidhuber (1997). However, the sequence-to-sequence model presented in Luong et al. (2015), as described below, better suits our problem compared to the LSTM. Firstly, we consider monotonic alignment (local) for the attention calculations rather than global attention scores. In global attention, the context is calculated by taking a weighted average of the hidden states of all periods in the whole input sequences. In the local attention mechanism, a few previous and later periods are considered for the attention score calculation for the period a prediction is made rather than the whole input sequence. In other words, if a prediction is being made for period t, the attention score, therefore, the context is calculated using the

hidden states of the encoder from period $t - D$ to $t + D$, where D represents the size of the window for the local-D attention.

There are two advantages to that choice. First, the combinatorial optimization problems we are trying to predict can be far longer than the problems used in training. Our predictions benefit from focusing on a time window of production and inventory decisions. In such cases, the initial periods have little to no effect on the final periods' decisions, considering the length and characteristics of the problems to reset the production amount with each production decision. For example, in MCLSP, once a prediction decision is made, the production amount is usually such that the demand is fully covered for a few subsequent periods. The inventory at the end of the last covered period is often reduced to zero to avoid the inventory holding cost. Then the process repeats itself by resetting the production amount. In this setting, the period that a prediction is being made has a more intertwined relationship with a few predecessor and successor periods rather than far away periods or the whole sequence. Second, local-D attention is less computationally expensive compared to global attention, and the training and prediction times can be reduced by using local-D attention compared to full-sequence attention.

Figure 3.1 shows an encoder-decoder model with attention used for a period t. The encoder takes input sequence $x_1, x_2, \ldots, x_m$ but focuses on sequence $x_{t-D}, \ldots, x_t, \ldots, x_{t+D}$ at time t with a time window of size D. If the boundaries of the input sequence exceed the problem's time horizon, the part outside of the window is ignored. The whole input sequence is processed at once with the forward and backward LSTM layers to generate the hidden states, but for the attention calculation, only a portion of this series is considered for the current period t. For a period t, the hidden state of the encoder generated by the forward layer $LSTM^e_{forward}$ is denoted by $\overrightarrow{h}^e_t$, and the hidden state of the encoder generated by the backward layer $LSTM^e_{backward}$ is denoted by $\overleftarrow{h}^e_t$. Hidden states are the output vectors of LSTM

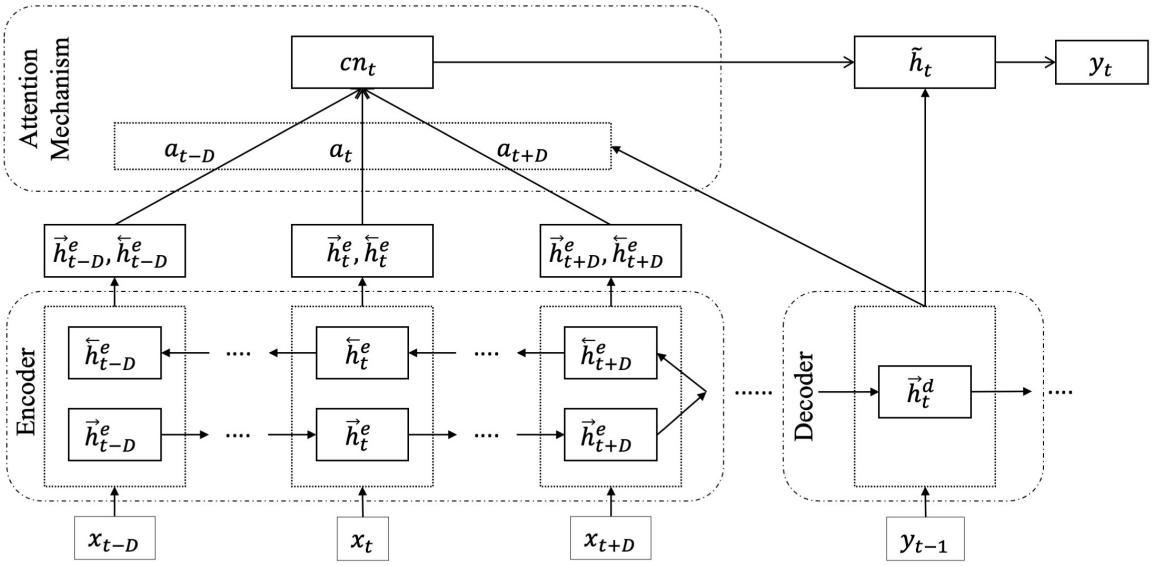


Figure 3.1 Encoder-decoder with attention.

that carry processed information related to the current period with the desired size. Those hidden states are calculated as:

$$\overrightarrow{h}^e_{t-D}, \ldots, \overrightarrow{h}^e_t, \ldots, \overrightarrow{h}^e_{t+D} = LSTM^e_{forward}(x_{t-D}, \ldots, x_t, \ldots, x_{t+D}) \tag{3.3a}$$

$$\overleftarrow{h}^e_{t-D}, \ldots, \overleftarrow{h}^e_t, \ldots, \overleftarrow{h}^e_{t+D} = LSTM^e_{backward}(x_{t-D}, \ldots, x_t, \ldots, x_{t+D}) \tag{3.3b}$$

For the current prediction period t, the decoder hidden state $\overrightarrow{h}^d_t$ is generated with the output y_{t-1} from previous period $t-1$ using the decoder network $LSTM^d$:

$$\overrightarrow{h}^d_t = LSTM^d(y_{t-1}) \tag{3.4}$$

Then a comparison is made between the current hidden state of the decoder $\overrightarrow{h}^d_t$ with all encoder hidden states in the attention window $t-D$ to $t+D$. The encoder hidden states of forward and backward LSTM layers from each period t in the attention windows are concatenated as $\overrightarrow{h}^e_t, \overleftarrow{h}^e_t$ to calculate attention scores together with the hidden state of the decoder $\overrightarrow{h}^d_t$ as given below:

$$a_i = \frac{exp(score(\overrightarrow{h}^d_t, [\overrightarrow{h}^e_i, \overleftarrow{h}^e_i]))}{\sum_{t'=t-D}^{t+D} exp(score(\overrightarrow{h}^d_t, [\overrightarrow{h}^e_{t'}, \overleftarrow{h}^e_{t'}]))} \quad \forall i = t-D, \ldots, t+D. \tag{3.5a}$$

where scores are calculated as:

$$score(\overrightarrow{h}^{d}_{t}, [\overrightarrow{h}^{e}_{i}, \overleftarrow{h}^{e}_{i}]) = \overrightarrow{h}^{d\intercal}_{t} W_{\alpha} [\overrightarrow{h}^{e}_{i}, \overleftarrow{h}^{e}_{i}], \tag{3.6a}$$

where W_{α} is a vector of learned parameters to calculate attention.

After the scores are calculated for each period in the attention window, the context cn_t, which ideally accumulates all relevant information of the input sequence, is calculated by taking a weighted average: $cn_t = \sum_{i=t-D}^{t+D} a_i * [\overrightarrow{h}^{e}_{i}, \overleftarrow{h}^{e}_{i}]$. Then context cn_t is concatenated with the current decoder hidden state $\overrightarrow{h}^{d}_{t}$, and passed through a linear layer with tanh activation function. Finally, predictions are generated by passing the activated output to another linear layer with a sigmoid activation function. The details of the model can be found in Luong et al. (2015).

3.4.2 The PredOpt Framework

Training and Validation. Our prediction approach to learning optimal solutions starts with data generation. We randomly generate all parameters for the optimization problem that is of interest. Then, the problems generated are solved using CPLEX. The resulting data set is divided into two following the standard schema for learning (Alpaydin, 2020): training and validation sets. Both training and validation data sets include the parameters of an instance and its optimal solution for a set of instances. Using the training set, the parameters of the neural machine translation model are optimized to create an output mapping to the optimal solutions for the problem of interest. Then models with different hyperparameters are evaluated using the validation set. A third independent test set of instances with a higher number of periods is generated to measure the effectiveness of the proposed PredOpt framework.

Testing. The PredOpt framework presents a strategy to eliminate infeasible predictions in a fast manner. In Chapter 2, the predictions of the decision variables are fixed in the solution, and the results show that predictions can cause solutions that are

infeasible at a significant rate. In Chapter 2, we state that infeasibility depends on the prediction level and can be adjusted empirically. This can be a rather computationally challenging process and does not guarantee the feasibility of the predictions. The main idea in the PredOpt framework is to construct a feasibility-check loop to determine the highest level of predictions that will not lead to an infeasible solution. In the PredOpt framework, the prediction level is not constant for each instance as in Chapter 2, but it is determined with a feasibility check loop and a relaxation of the original problem, as described in the next two sub-sections.

Predicting Tight Constraints and Forming the Relaxation Problem A relaxation of the original problem is formed by identifying the tight and close-to-tight constraints to reduce the feasibility checking time of the PredOpt framework. For brevity, we will refer to those inequalities as tight even if the inequality is not held strictly. Those constraints that are identified as tight are included in the relaxation of the formulation, and those that are identified as not tight are removed from the relaxation.

During the training, the model is not only used to learn the values of the optimal binary decision variables but is also used to learn the tight constraints. Thus, before training starts, using the optimal solutions from the training data sets, we determine tight constraints and label them so that the trained model can learn which constraints are tight in the test set. For example, given that x^*_{it} and y^*_{it} are the optimal solutions to an MCLSP (3.1) training instance, constraints (3.1c) are identified to be tight with a predetermined tightness coefficient $\eta \in [0, 1]$ if $\sum_{i=1}^{I} x^*_{it} \geq \eta * c_t$ and identified to be non-tight otherwise. The constraints (3.1d) are identified to be tight if $x^*_{it} \geq \eta * y^*_{it} c_t$ and identified to be non-tight otherwise. For the MSMK presented in formulation (3.2), constraints (3.2b) are tight if $\sum_{i=1}^{I} w_{ijt} x^*_{it} \geq \eta * c_{jt}$ and non-tight otherwise. The constraints (3.2c), (3.2d), and (3.2f) are discarded from the relaxation since they

do not affect the feasibility of the knapsack constraints presented. The relaxation involves the binary decision variables for both MCLSP and MSMK.

As explained above, the full training data set includes both the optimal solutions to the problem and the tight constraints that were identified from those optimal solutions. In MCLSP, at each period t, the network predicts a total of $2 \cdot I + 1$ variables, where I of them are used to predict variables y_{it} and I number of variables is used to predict the tightness of I constraints (3.1d). Then the additional variable represents the tightness prediction for the single constraint (3.1c) at time t. Similarly, in MSMK, for each period t, a total of $I + J$ predictions are made where the first I represents the decision variables x_{it} and the remaining J predicts if the constraints (3.2b) are tight or not. While the sizes of the output dictionaries are $2 \cdot I + 1$ and $I + J$ for MCLSP and MSMK, respectively, in neural machine translation, at each period, a single element in the output dictionary is predicted. In PredOpt, unlike neural machine translation, the problem is a multi-label binary classification problem. Therefore, at each period, a subset of the output dictionary, possibly with multiple elements, is selected.

Feasibility Check and Infeasible Elimination Loop As previously discussed, an attentional encoder-decoder model is used to learn both optimal solutions and tight constraints of the problem of interest. Then during the testing phase, a strategy is applied to determine the appropriate level of prediction for each instance. Figure 3.2 presents the PredOpt framework in detail. In the left side of the figure (Feasibility Check and Infeasible Elimination), for each instance in the test set, a set of predictions are generated using the trained network. Using the set of predicted tight constraints, a relaxation of the original model is generated for a fast feasibility check. Then $predLevel\%$ of predicted variables are fixed in the relaxed formulation. This is done by calculating $maximum(\hat{y}, 1-\hat{y})$, ordering decreasingly, and taking the top $predLevel\%$

to fix in the model where $\hat{y}$ represents predictions. In this process, variables are ordered based on their closeness to either 0 or 1, and the first *predLevel*% of them are fixed in the relaxed problem. If a feasible solution is found by solving the relaxed instance with fixed variables using CPLEX, the loop is exited, and the determined level of prediction is used in the original formulation to determine other variables' values and the optimal objective function value. If a feasible solution is not found, the *predLevel*% is reduced by *reduceLevel*%, which cannot be lower than zero, and the feasibility check is repeated with new *predLevel*% until a feasible solution is found.

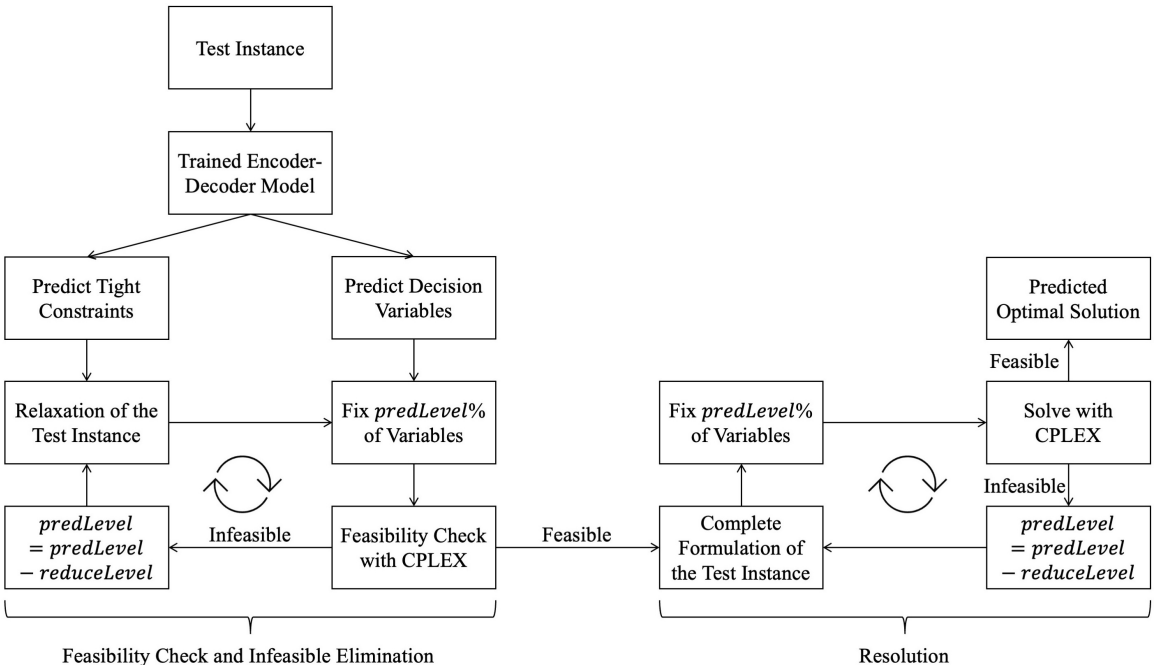


Figure 3.2 PredOpt framework.

Once a feasible solution is found, a new loop begins to solve the complete formulation of the test instance, as demonstrated on the right side of Figure 3.2 (Resolution). The determined level of prediction is fixed in the original formulation, which is attempted to be solved with CPLEX. If a feasible solution is found, the loop is exited, and the optimal solution is used to report the quality of the proposed framework. The relaxation of the problem from the first loop of Figure 3.2 does

not necessarily guarantee a prediction level that gives a feasible solution since it is only a relaxation. Thus, if a feasible solution is not found by solving the original problem with fixed values, similar to the first loop on the left side, $predLevel\%$ is reduced by $reduceLevel\%$, and the loop continues until a feasible solution is found. Computational results show that the number of iterations in the second loop is very few compared to the number of iterations made in the first loop, highlighting the high quality of the relaxation problems in the first loop.

3.4.3 Generalization with Item-wise Expansion

In this section, we discuss the generalization of the trained models to predict longer and higher-dimensional problems. The encoder-decoder can be inherently used to predict for the longer instances than they are trained with since the size of the predictions is not limited by the length of training instances. The attention structure helps with the collection of encoder information and passes it to the decoder without a limitation on the input length. Therefore, the encoder-decoder with attention is able to keep up with the increasing number of periods, and thus, the time-wise expansion of the prediction framework is straightforward. The key question we address here is if models trained with a few items can be used to predict problems with a large number of items. For example, a significant time reduction can be achieved if an 8-item model can successfully predict a 32-item problem. Therefore, we can solve instances with a small number of items to predict instances with a larger number of items. This would allow us to perform training only once to solve instances generated from the same distribution without retraining if the number of items considered changes. Also, it can reduce the training instance generation and the training time.

Here, our strategy includes making multiple forward passes using the trained model with a subset of the items instead of predicting all decision variables at once. In Algorithm 1, we present our strategy for generating predictions for instances with

a larger set of items. In step 1, we initialize the prediction counter for each item i, γ_i, as zero. The algorithm is continued until each item i has been predicted at least $\delta = 10$ times. At each iteration, we select a subset S of items such that the size of the subset S is equal to I^M, representing the number of items the model is trained with. For example, if a prediction is made using an 8-item model, then only $|S| = 8$ items are selected in step 3. Then, in step 4, we make a forward pass with the model M using the input data α_S of selected subset S of items to generate predictions $\hat{\theta}_S$. Here, the right-hand sides of the constraints are scaled down with the proportion of selected items aiming to mimic the original problem. For MCLSP, we modify the right-hand side of constraint (3.1c) as $c_t = c_t \times \frac{\sum_{i \in S} d_{it}}{\sum_{i=1}^{I} d_{it}}$, $\forall t = 1, \ldots, T$. For MSMK, we modify the right-hand side of constraint (3.2b) as $c_{jt} = c_{jt} \times \frac{\sum_{i \in S} w_{ijt}}{\sum_{i=1}^{I} w_{ijt}}$, $\forall t = 1, \ldots, T,\ \forall j = 1, \ldots, J$. Then predictions are saved in step 5 by summing current predictions $\hat{\theta}_i$ and previous prediction $\hat{\beta}_i$ for each item i in the subset S. Then, in step 6, the count of prediction γ_i is increased for each item i in the subset S. Finally, the final prediction $\hat{\beta}_i$ for each item i in the whole test set is calculated as dividing the sum of predictions $\hat{\beta}_i$ by their respective counts γ_i.

3.5 Implementation and Experimentation

In this section, we present the details of instance generation, encoder-decoder parameters, and metrics used in measuring the performance of the PredOpt framework. The MCLSP and MSMK instances are solved using Python 3.8.5 with DOcplex API and CPLEX 20.1.0. The generation of training and test instances, model training, and testing with PredOpt are completed on 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 with Python 3.8.5. The encoder-decoder model is trained with PyTorch 1.7.1 on GPU.

Algorithm 1 Item-wise Generalization Prediction Algorithm

Input: Trained model M, number of items during model training I^M, input data of the test set α, set of items in the test set $i \in \{1, \ldots, I\}$, and threshold prediction count δ

Output: Predicted value for item $\forall i \in \{1, \ldots, I\}$, $\hat{\beta}_i$, for the test set

Item-wise Generalization

1: Initialize prediction count of each item to be zero: $\gamma_i = 0$, $\forall i \in \{1, \ldots, I\}$
2: **while** $\exists\ \gamma_i \leq \delta\ i \in \{1, \ldots, I\}$ **do**
3: Sample a subset of items, S, with the number of items in the subset equal to the number of items in the trained model I^M: $S \subset \{1, \ldots, I\}$ and $|S| = I^M$
4: Make a forward pass using the input data of the selected subset S of items: $\hat{\theta}_S = M(\alpha_S)$
5: Save predictions for subset S of selected items: $\hat{\beta}_i := \hat{\beta}_i + \hat{\theta}_i$, $\forall i \in S$
6: Increase prediction counts for subset S of selected items: $\gamma_i = \gamma_i + 1$, $\forall i \in S$
7: **end while**
8: Calculate the final prediction value for each item $i \in \{1, \ldots, I\}$ by dividing the sum of saved predictions with respective counts: $\hat{\beta}_i := \hat{\beta}_i \div \gamma_i$.

3.5.1 Instance Generation

MCLSP Instances: To generate instances, we employ the scheme presented in Büyüktahtakın et al. (2018b). Two underlying parameters can simulate the problems of varying hardness levels: capacity-to-demand ratios $c \in \{10, 14\}$ and setup-to-holding cost ratio $f = 1,000$. The uniform integer distribution between a and b is denoted by $U[a, b]$. The shared capacity c_t between items is sampled from $U\left[0.8c\bar{d}, 1.2c\bar{d}\right]$ where $\bar{d}$ is the overall average demand. The unit production cost p_{it} is sampled from $U[1, 200]$, the holding cost h_{it} is sampled from $U[1, 100]$, and the demand d_{it} is sampled from $U[500, 1500]$. The periodic setup cost is sampled from $U\left[0.9f\bar{h}, 1.1f\bar{h}\right]$ where $\bar{h}$ is the overall average holding cost. A planning horizon of $T = 40$ with the number of items of $I = 8$ and $T = 30$ with $I = 12$ are used in the training of two different encoder-decoder models. First model with $I = 8$

is used to predict instances with $T \in \{40, 60, 80, 100, 150, 200\}$. Also, the first model is used to predict item-wise generalization instances with $I \in \{32, 40, 80\}$ and $T \in \{200, 150, 100\}$. Second model with $T = 30$ with $I = 12$ is used to predict instances with $T \in \{30, 50, 75, 100, 125, 150\}$. Also, the second model is used to predict item-wise generalization instances with $I \in \{36, 48, 60\}$ and $T \in \{125, 100, 80\}$. For MCLSP, a total of 18 sets of instances are used for testing, each containing 20 instances.

MSMK Instances: The instances are solved by sampling the problem parameters from the following integer uniform distributions. The profit p_{it} of each item i at period t is sampled from $U[1, 1000]$, the stability bonus b_{it} from $U[1, 1000]$, item weights w_{ijt} from $U[1, 1000]$, and the capacity c_{jt} from $U\left[0.5\sum_{i=1}^{I} w_{ijt}, 0.8\sum_{i=1}^{I} w_{ijt}\right]$. The number of periods for training is $T = 30$ and the number of items is $I \in \{8, 10\}$. For the first model with $I = 8$, testing instances with $T \in \{30, 50, 80, 100, 150, 200\}$ are used. Additionally, item-wise generalization test instances with $I \in \{16, 24, 32\}$ and $T \in \{30, 20, 15\}$ are used. For the second model with $I = 10$, testing instances with $T \in \{30, 50, 70, 80, 90, 100\}$ and the item-wise generalization test instances with $I \in \{20, 30, 40\}$ and $T \in \{30, 20, 15\}$ are used. For MSMK, 18 test sets are also generated in total with each having 20 instances.

3.5.2 Model Training

In this section, we discuss the specific architectural details and hyperparameters of the trained models. In these models, the number of layers is two, and the number of hidden units in the encoder is 128 for Table 3.1, 64 for Table 3.2, 256 for Table 3.3, and 128 for Table 3.4. The decoder for each model is twice the size of the encoder. Our choice of attention score mechanism is based on the general score calculations. Our choice of optimizer for the model training is the well-known Adam Optimizer (Kingma and Ba, 2014), with an initial learning rate of 0.01 for Tables 3.1 and 3.2

and 0.001 for Tables 3.3 and 3.4. We have utilized the dropout technique with a rate from $\{0.25, 0.30, 0.35\}$. It is a commonly used approach to prevent overfitting (Srivastava et al., 2014). We have also utilized a label smoothing approach to achieve a better generalization (Müller et al., 2019). Our models are trained with 3,500,000 instances for each problem type. Training times are 18 and 30 hours for MCLSP with 8 and 12 items, respectively. Training times are 24 and 8 hours for MSMK with 8 and 10 items, respectively.

3.5.3 Evaluation Methodology

In this section, we present the metrics used in evaluating the performance of the PredOpt framework. The metrics are intended to evaluate the success of the PredOpt framework with respect to the optimality gap and reduction in solution time for the test set. We mainly follow the success metrics defined in Chapter 2:

- **timeCPX**: Average solution time of an optimization problem in CPU seconds with CPLEX at default setting without using any predictions.
- **timePredOpt**: Average solution time of an optimization problem in CPU seconds with CPLEX at default setting using predictions determined by PredOpt framework, including the prediction generation and infeasibility elimination time.
- **timeLS**: Average solution time of an MCLSP in CPU seconds with CPLEX using valid (ℓ,S) inequalities of Barany et al. (1984).
- **timeHeur**: Average solution time in CPU seconds with the relax-and-fix heuristic of Absi and van den Heuvel (2019) for MCLSP or adaptive fixing heuristic of Bertsimas and Demir (2002) for MSMK.
- **accuracy(%)**: Percentage of binary variables correctly predicted by the PredOpt framework compared to the optimal solution determined by CPLEX.

Also, the following metrics are defined to assess the quality of the PredOpt framework:

Definition 3.5.1 *Let x^* be the optimal solution of the problem of interest and $Z(x^*)$ be the corresponding optimal objective function value. Let $\hat{x}^*$ be the solution*

determined by the PredOpt framework (or solution determined by a heuristic for calculating optGapHeur(%)) and $Z(\hat{x}^*)$ be the corresponding optimal objective function value. The optimality gap is defined as:

$$\textbf{optGapPredOpt}(\%) = \frac{|Z(\hat{x}^*) - Z(x^*)|}{Z(x^*)} \times 100. \tag{3.7}$$

Definition 3.5.2 *The solution time improvement factor achieved by the PredOpt framework (timeImpLS for CPLEX with (ℓ,S) inequalities and timeImpHeur for heuristic time) with respect to the default CPLEX is given by:*

$$\textbf{timeImpPredOpt} = \frac{timeCPX}{timePredOpt}. \tag{3.8}$$

Definition 3.5.3 *The **p-value** is calculated based on a one-sided Wilcoxon signed-rank test (Wilcoxon, 1945). It is a statistical test that measures if the differences between the two distributions are symmetric around 0. It is a non-parametric version of a paired T-test. The null and alternative hypotheses are:*

$$H_0 : median(timeHeur - timePredOpt) < 0 \tag{3.9a}$$

$$H_1 : median(timeHeur - timePredOpt) > 0 \tag{3.9b}$$

If the p-value is less than 0.01, the null hypothesis is rejected, implying that PredOpt performs statistically better than the corresponding heuristic in terms of solution time.

3.6 Computational Results

In this section, we present the computation results of the PredOpt framework for solving a variety of MCLSP and MSMK instances and compare the results to the direct solution with CPLEX. We also compare the performance of the PredOpt algorithm with the state-of-the-art heuristics: the heuristic of Absi and van den Heuvel (2019) for MCLSP and the heuristic of Bertsimas and Demir (2002) for MSMK. In the heuristic of Absi and van den Heuvel (2019), at each iteration, the number of fixed periods is taken as $\frac{T}{20}$, and the number of periods with binary decision variables is set to be $\frac{T}{10}$.

For each problem, we have created 18 different test sets, each consisting of 20 instances. The initial prediction level is set at 80% for MCLSP and 60% for MSMK, which are effective at estimating the prediction level that eliminates infeasibility. Within the PredOpt framework, the time of generating predictions and finding a prediction level that leads to a feasible solution for the relaxed problem is below one second, which is very fast compared to solving the full-sized problem with the determined prediction level. All computational results referring to solution times are presented in CPU seconds.

3.6.1 Quality of Predictions for MCLSP

Table 3.1 presents the results for a set of MCLSP instances. Here the model is trained only with $T = 40$-period instances, but it is used to test instances from $T = 40$ up to $T = 200$. This shows that the trained models can easily generalize in time dimension with the presented local attention structure. The first set of test instances with $T = 40$ are solved with a mean solution time of 2.3 seconds with CPLEX. In all instances in this test set, the time to generate predictions and infeasibility elimination loop is fairly short compared to the optimization (resolution) loop of the PredOpt framework shown in Figure 3.2. For example, for the first two sets of instances with $T = 40, 60$ in Table 3.1, the optimal level of predictions that do not cause infeasibility is calculated within 0.01 seconds using the relaxation of the problem. Then, with the 80% prediction level, the first set of instances are solved in 9-fold solution time faster than CPLEX with only a 0.01% optimality gap. The PredOpt reaches the same average objective function value over 20 test instances faster when compared with the solution strengthened with the (ℓ,S) inequalities or the heuristic of Absi and van den Heuvel (2019). We do not report an optimality gap with the (ℓ,S) inequalities since they solve the test instances to optimality without a solution time limit, but the optimality gap of the heuristic is much higher than the optimality gap of PredOpt.

The PredOpt is statistically faster than the heuristic since the p-value is smaller than 0.001.

For all periods, the time of the PredOpt framework is much less than the CPLEX solution time. The time improvements increase as the number of periods increases and problems get harder. For example, in the last data set with $T = 200$, the solution time is reduced from more than 10 minutes to slightly above 3 seconds with an optimality gap of 0.02%. Also, the accuracy of the models does not deteriorate as the number of periods multiplies, confirming that the encoder-decoder with the local attention structure is able to capture the problem characteristics. The optimality gap is kept under 0.03% for all cases in the test sets used in Table 3.1.

As a summary of Table 3.1, our model trained with shorter periods can predict 5-fold longer problems with the local attention structure. For all test instances, the PredOpt provides a faster solution than the CPLEX, (ℓ,S) inequalities and the heuristic with a better optimality gap than that of the heuristic. The optimality gap stays constant over all test instances with an average of 0.02%, and the PredOpt captures the characteristics of the problem even if the size of the test instances grows significantly.

Table 3.2 presents a harder set of test instances with 12 items. The model is only trained using $T = 30$ and $I = 12$ instances, but the results are presented for solving models with periods ranging from $T = 30$ to $T = 150$. Similar to Table 3.1, the time improvements get significantly better as the number of periods in the test problem multiplies. For example, the data set with $T = 125$ is solved with an average of more than 1 hour using CPLEX at the default setting. The PredOpt reduces the solution time to under 7 seconds with an optimality gap below 0.1% without infeasibility in the test set. For the last data set with $T = 150$, the solution time is reduced from almost 7 hours to just under 5 seconds with an optimality gap of 0.11%. This translates into a solution time reduction with a factor of 7,236. For all test sets,

Table 3.1 Average Results of Experiments for MCLSP with 8 Items

T	40	60	80	100	150	200
timeCPX	2.3	4.2	4.9	30.9	62.2	659.2
timePredOpt	0.3	0.4	0.6	1.2	2.1	3.4
timeLS	3.0	5.6	7.5	12.5	35.9	66.6
timeHeur	10.6	10.5	12.0	13.1	19.9	23.7
timeImpPredOpt	9	12	8	22	28	183
timeImpLS	1	1	1	1	1	6
timeImpHeur	0	0	0	2	3	32
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	99.6	99.7	99.7	99.6	99.7	99.7
optGapPredOpt(%)	0.01	0.01	0.02	0.03	0.02	0.02
optGapHeur(%)	1.08	1.11	1.03	1.08	0.41	1.05

the accuracy is maintained as the time period increases. Also, the p-value for the Wilcoxon signed-rank test is fairly small with a lower optimality gap compared to the heuristic for all cases, showing that the PredOpt reaches a better objective function value in a shorter time than the heuristic. While (ℓ,S) inequalities are effective in reducing the solution time for harder instances, they are still significantly slower than the PredOpt at the cost of a very small optimality gap.

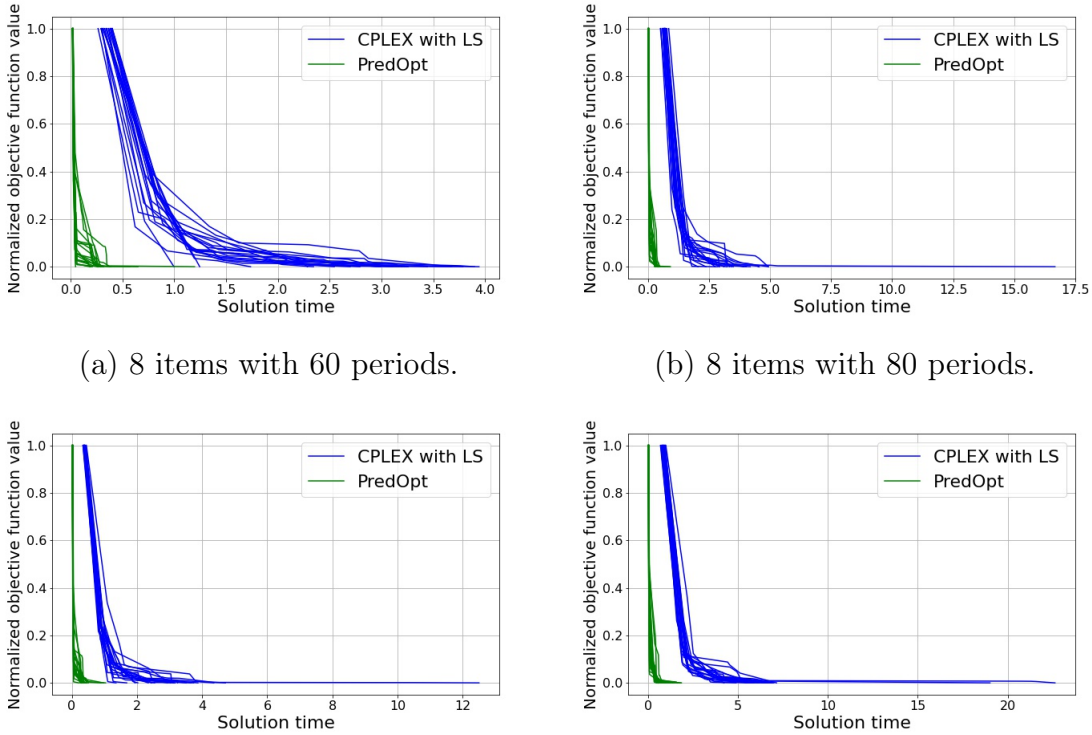
To sum up the results in Table 3.2, the increase in the time improvement factors increases significantly for harder problems. The optimality gap stays quite steady as we increase the number of periods in the test set, which highlights the potential of the PredOpt framework to successfully predict for much longer instances. Also, the PredOpt framework is faster and better than the specially-designed heuristic for the MCLSP.

PredOpt Reduces Objective Value Faster than CPLEX It is important to compare the solution speed of the PredOpt with CPLEX since, in some cases, CPLEX

Table 3.2 Average Results of Experiments for MCLSP with 12 Items

T	30	50	75	100	125	150
timeCPX	1.1	10.3	80.9	86.8	3,982.5	25,085.3
timePredOpt	0.3	0.6	1.5	2.0	6.3	4.6
timeLS	2.4	5.9	12.0	18.2	2,259.4	53.1
timeHeur	18.0	14.9	17.5	18.7	22.2	27.6
timeImpPredOpt	4	14	47	40	696	7,236
timeImpLS	0	2	4	5	49	453
timeImpHeur	0	1	4	5	172	867
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	99.5	99.3	99.2	99.0	99.0	98.9
optGapPredOpt(%)	0.04	0.05	0.06	0.10	0.09	0.11
optGapHeur(%)	0.52	0.86	0.83	1.14	0.87	0.43

quickly identifies a good solution but takes a long time to prove the optimal one (Accorsi et al., 2022). In Figure 3.3, the progress during the solution process, in terms of a normalized objective function value averaged over 20 test instances, is presented to visualize the improvement of the PredOpt framework over CPLEX with (ℓ,S) inequalities. Figure 3.3a-3.3b present the progress for Table 3.1's second and third sets of test instances with $T = 60, 80$. Figure 3.3c-3.3d present the progress for Table 3.2's second and third sets of test instances with $T = 50, 75$. As discussed in the previous results, Figure 3.3 demonstrates that the time of PredOpt is less than that of CPLEX with (ℓ,S) inequalities to reach a particular objective function value. The graphs show that PredOpt reduces the objective function value much faster than CPLEX with (ℓ,S). In all solution progress plots, the objective values and solution times for PredOpt and CPLEX with (ℓ,S) are visually quite distinguished, showing that the PredOpt framework helps improve the objective function quicker than CPLEX even in the first few seconds of the solution process.



(a) 8 items with 60 periods.

(b) 8 items with 80 periods.

(c) 12 items with 50 periods.

(d) 12 items with 75 periods.

Figure 3.3 Progress of CPLEX with (ℓ,S) inequalities and PredOpt objective values during the first few seconds of the solution process. All solution times are given in CPU seconds.

3.6.2 Quality of Predictions for MSMK

This section presents the results for the MSMK, similar to Section 3.6.1. Table 3.3 presents the results for the model trained with the 8 items, 30 periods, and 5 resource constraints. The model is used to predict the optimal solution of instances with a wide range of periods from $T = 30$ up to $T = 200$. Unlike Section 3.6.1, we do not compare with an exact algorithm like (ℓ,S) inequalities but compare the solution performance with the CPLEX performance and the heuristic of Bertsimas and Demir (2002). In Table 3.3, the solution time of 30-period instances is reduced by a factor of 6 with respect to the CPLEX solution time. As the instances become harder, the time improvement increases significantly. For the last data set with 200 periods,

solution time is reduced from more than 3.5 hours to 3 seconds with four orders of magnitude solution time reduction by PredOpt over CPLEX. The accuracy of the variables selected by the PredOpt framework is somewhat lower than the MCLSP instances presented in Tables 3.1 and 3.2. Therefore, the resulting optimality gaps are at a slightly higher level. We believe that the harder combinatorial nature of the MSMK possesses a larger learning challenge. Also, the existence of multiple tighter constraints with pure binary variables per period for MSMK, i.e., Equation (3.2b), compared to a single binding constraint per period involving continuous variables, i.e., Equation (3.1c), for MCLSP, makes MSMK a more challenging problem to predict. However, the optimality gaps are still below 0.98%, which is a much better solution performance than the heuristic, which gives an optimality gap above 2.7%. Also, the PredOpt achieves a better objective function statistically faster than the heuristic since all p-values for the Wilcoxon signed-rank test are smaller than 0.001.

As a summary of Table 3.3, the PredOpt framework outperforms the heuristic in both time and optimality gap aspects. The solution time is reduced by four orders of magnitude with respect to CPLEX and one-to-two orders of magnitude with respect to the heuristic. This shows that our framework is beneficial for tackling instances that are hard to solve.

In Table 3.4, results for the model trained with the 10 items, 30 periods, and 4 resource constraints are presented. These results show similarities with the results presented in Table 3.3. The solution times are reduced significantly for all test instances with periods from $T = 30$ to $T = 100$. In the final data set with $T = 100$, the solution time is limited by four hours with CPLEX, resulting in an optimality gap of 0.002%. The solution time is reduced from 4 hours to under a second with an optimality gap of 0.68%, resulting in a very significant solution time reduction. For the same data set, the heuristic can only achieve a solution with a 2.4% optimality gap in 4 seconds. To sum up Table 3.4, the p-values for the Wilcoxon signed-rank

Table 3.3 Average Results of Experiments for MSMK with 8 Items

T	30	50	80	100	150	200
timeCPX	0.9	4.1	67.7	772.8	4,579.8	12,862.5
timePredOpt	0.2	0.2	0.4	2.4	0.5	2.7
timeHeur	7.7	14.1	24.2	43.2	73.6	121.3
timeImpPredOpt	6	22	207	2,854	12,449	15,218
timeImpHeur	0	0	3	17	62	107
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	92.9	93.6	92.9	91.6	92.3	93.8
optGapPredOpt(%)	0.75	0.67	0.80	0.91	0.98	0.71
optGapHeur(%)	2.74	2.92	3.07	3.01	2.87	2.94

test are very small for the instances that are hard to solve, ensuring that the PredOpt framework performs better than the heuristic in both the time and optimality gap aspects.

Table 3.4 Average Results of Experiments for MSMK with 10 Items

T	30	50	70	80	90	100
timeCPX	2.5	41.7	1,051.4	4,408.8	9,057.8	14,409.7
timePredOpt	0.2	0.4	0.5	0.5	52.5	0.7
timeHeur	8.0	14.9	22.2	30.9	34.2	40.9
timeImpPredOpt	14	170	2,027	8,673	12,580	28,451
timeImpHeur	0	3	49	138	262	355
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	93.0	92.5	92.4	93.1	91.2	93.2
optGapPredOpt(%)	0.80	0.81	0.67	0.72	1.29	0.68
optGapHeur(%)	2.46	2.37	2.35	2.49	2.51	2.40

3.6.3 Generalization: Quality of Predictions with Item-wise Expansion Algorithm

In this section, we present results for item-wise generalization by applying Algorithm 1 presented in Section 3.4.3. Table 3.5 presents the results of the item-wise generalization for MCLSP. The first three test sets in Table 3.5 are solved by using the trained model with $T = 40$ and $I = 8$, and the last three test sets are solved by using the trained model with $T = 30$ and $I = 12$. For example, the model trained with 8 items and 40 periods is used to predict the test set with 32 items and 200 periods in the first column of results. Using Algorithm 1 within the PredOpt, the solution time is reduced by a factor of 62 with only a 0.01% optimality gap. The overall results show similarities with previously presented tables. The time improvements get better as problems get harder. In the fourth data set, solution time is reduced from more than 75 minutes to under 15 seconds with an optimality gap of only 0.05%. This translates into a solution time reduction with a factor of 406. Also, all p-values are smaller than 0.01, ensuring that PredOpt can achieve a statistically faster solution time when compared to the relax-and-fix heuristic of Bertsimas and Demir (2002). These results highlight that the PredOpt framework can be successfully used to predict instances with both longer planning horizons and a larger number of items.

Table 3.6 presents a set of results for the item-wise expansion results for the MSMK, similar to the results of item-wise generalization for MCLSP in Table 3.5. The first three sets of results in Table 3.6 are calculated using the trained model used in Table 3.3, and the remaining three sets of results are calculated with the trained model used in Table 3.4. Here, we reduce the number of periods as we increase the number of items for both cases to test the limits of our periodical attention-based learning framework. Compared to the item-wise generalization of MCLSP, the MSMK has a slightly growing optimality gap as the number of items increases and the period decreases. For example, the third data set with 32 items achieves an optimality gap

Table 3.5 Average Results of Item-wise Generalization Experiments for MCLSP

Train Items	8			12		
Test Items	32	40	80	36	48	60
T	200	150	100	125	100	80
timeCPX	1,045.1	73.2	27.9	4,624.2	1,020.6	115.8
timePredOpt	17.5	9.6	6.3	12.4	13	8.9
timeLS	290.3	95.9	52.4	68.4	46.5	31.8
timeHeur	36.2	34.6	38.3	28.1	27.6	26.6
timeImpPredOpt	62	7	5	406	73	11
timeImpLS	3	1	1	74	23	3
timeImpHeur	28	2	1	163	35	4
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	99.8	99.8	99.9	99.3	99.5	99.6
optGapPredOpt(%)	0.01	0.01	0.01	0.05	0.03	0.03
optGapHeur(%)	1.18	0.45	1.20	0.94	1.18	1.2

of 1.33% with the PredOpt framework and model trained with 8 items. However, this result is obtained in a very small fraction of CPLEX solution time, i.e., 74 minutes of CPLEX compared to 0.3 seconds with PredOpt. For all experiments, PredOpt is significantly faster than the heuristic since all p-values are smaller than 0.01. Also, the optimality gaps of PredOpt are lower than the heuristic except for the third and sixth data sets. The increased optimality gap in the third and sixth data sets occurs because a lower number of periods in the test set reduces the effectiveness of the attention mechanism, and the increasing number of items reduces the success of the item-wise generalization algorithm. On the other hand, PredOpt performs better than CPLEX and the heuristic for all instances in terms of the solution time. These

Table 3.6 Average Results of Item-wise Generalization Experiments for MSMK

Train Items	8			10		
Test Items	16	24	32	20	30	40
T	30	20	15	30	20	15
timeCPX	466.1	827.6	4,494.3	3,964.5	6,225.0	4,210.7
timePredOpt	0.4	0.3	0.3	0.5	0.5	0.3
timeHeur	9.5	7.5	5.8	10.2	6.7	5.1
timeImpPredOpt	1,130	3,149	14,728	9,029	13,268	13,536
timeImpHeur	49	104	780	372	955	874
p-value	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001	≤ 0.001
accuracy(%)	91.2	90.7	88.7	94	93.2	90.5
optGapPredOpt(%)	0.93	1.18	1.49	0.66	0.67	1.77
optGapHeur(%)	1.88	1.33	1.14	1.5	1.25	1.02

results show that PredOpt is a good alternative to exact solvers and heuristics to use when a fast and accurate solution is needed.

3.7 Conclusions and Future Work

In this study, we present a learning-based framework to solve sequentially dependent decision-making problems. Our PredOpt framework can help solve optimization problems in an industrial setting where problems with similar structures are needed to be solved frequently with slightly different parameters. We proposed a learning framework based on a neural machine translation architecture. In the PredOpt framework, we present a strategy to eliminate any infeasible predictions of decision variables in a fraction of a second. To achieve that, a fast feasibility check is performed with a relaxed and smaller problem, which is also generated by using the same trained neural network. Once the best prediction level is determined, the problem is solved in an integrated way with the commercial solver. Also, we have shown that the models trained on shorter-period problems can be successfully used to predict instances that

have multiple times longer periods using a local attention mechanism. The results show that the solution time can be reduced up to three orders of magnitude with an optimality gap below 0.1%. Also, a statistical test confirms that the solution time of the PredOpt framework is faster than the heuristics. Furthermore, we develop and implement an item-wise generalization algorithm and show that the models trained on a small number of items can predict instances with a much larger number of items.

Our PredOpt framework shows a promising direction in integrating learning-based frameworks with state-of-the-art commercial solvers. The learners can be used to make the easier decisions, and harder decisions can be left to exact approaches to solve. In this spirit, future studies that integrate ML together with traditional OR approaches have the potential to bring out the best of both worlds. Also, we have shown that ML has a potential for generalization from different directions, such as time and number of items. Further studies can focus on performing learning from solutions of smaller and easier problems and extend this knowledge to solve larger and harder problems. Another future direction could be further generalizing the PredOpt framework to predict instances with different underlying distributions.

CHAPTER 4
A DEEP REINFORCEMENT LEARNING FRAMEWORK FOR SOLVING TWO-STAGE STOCHASTIC PROGRAMS

4.1 Introduction

The objective of this study is to develop a deep reinforcement learning methodology for solving scenario-based two-stage stochastic programs. Stochastic programming is the field of mathematical optimization that offers solutions to stochastic systems (Prékopa, 2013). The roots of optimization under uncertainty date back to the 1950s, and it is a multidisciplinary area at the intersection of operations research, statistics, probability, economics, and mathematics. We refer to the book by Birge and Louveaux (2011) for methodologies used in stochastic programming. Since many real-world applications involve some degree of uncertainty, stochastic programming approaches are prevalent in many fields, including but not limited to finance, energy planning, transportation, telecommunications, and agriculture.

Reinforcement learning is a subfield of machine learning where a decision-making agent learns to take actions to maximize a reward signal. Historically, reinforcement learning has been a major focus area in many disciplines, including operations research, control theory, and game theory, with different names. In general, the reinforcement learning environment can be described with a Markov decision process and can be modeled with a 4-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 core problems in the operations research literature, including the traveling salesperson problem (Bello et al., 2016), knapsack problem (Afshar et al., 2020), and vehicle routing problem (Nazari et al., 2018). 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. For that cause, we develop and present two-stage reinforcement learning (2SRL) framework.

Two-stage problems are one of the most famous cases of stochastic programs originated by Dantzig (1955). The decisions in the system are made in two stages, and uncertainty is observed between these two stages. In the first stage, a decision is made without the observation of the uncertain element of the system. Then the realization of the uncertainty is observed, and a second-stage decision is made accordingly. The objective function of the two-stage stochastic programs involves the minimization of the first-stage cost and the expected second-stage cost. Usually, this uncertainty is modeled with the scenarios. A distinct second-stage decision would be taking place for each scenario, but the first-stage decision should be the same for all scenarios. This is owing to the fact that the first-stage decision is a decision made before the observation of uncertainty. Figure 4.1 shows this sequential dependency as a scenario tree graph where the uncertainty is modeled with S scenarios in total. The probability of the realization of each scenario can be equal to $\frac{1}{S}$ or generated using a different probability distribution.

This approach of modeling stochastic programming problems with scenario trees has been used widely in a broad range of problems, including disaster management (Barbarosoğlu and Arda, 2004), supply chain management (Marufuzzaman et al., 2014; Gao and You, 2015), transportation (Liu et al., 2009), biofuel production (Cobuloglu and Büyüktahtakın, 2017), nurse staffing (Kim and Mehrotra, 2015), water resources management (Guo et al., 2010), airline scheduling (Yen and Birge,

2006), waste management (Maqsood and Huang, 2003), airport runway scheduling (Solveling et al., 2011), energy systems planning (Skar et al., 2014; Liu et al., 2010) and risk-averse optimization (Tajeddini et al., 2014; Bushaj et al., 2022a). Stochastic programming, particularly two-stage scenario-based stochastic programs, is a powerful way of modeling many systems. However, stochastic programs involving integers are often NP-hard and require specially designed solution methodologies such as cutting planes and decompositions with complex implementations (Linderoth and Wright, 2003; Fábián and Szőke, 2007). Therefore, there is still a lack of solution approaches for online applications such as ride-sharing (Feng et al., 2021) and electric vehicle charging (Wu and Sioshansi, 2017), where a solution is needed quickly. Online problems commonly arise in revenue management, internet advertising, and scheduling appointments in health care (Hwang et al., 2021). Achieving fast solutions to two-stage stochastic programs can be even life-saving in settings like disaster management (Grass et al., 2020). Furthermore, those developed algorithms must run from scratch for each problem, making it impractical for large-scale real-time applications.

There have been previous attempts to achieve near-optimal solutions to stochastic programs such as vehicle routing problem (Nazari et al., 2018), traveling salesperson problem (Kool et al., 2018), and bin packing problem (Balaji et al., 2019) by applying deep reinforcement learning techniques. 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.

Due to their complex nature, two-stage scenario-based stochastic problems are known to be computationally challenging to solve using off-the-shelf solvers such as CPLEX or Gurobi. 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 (Laporte and Louveaux, 1993; Angulo et al., 2016), regularized decomposition (Ruszczyński, 1986), dual decomposition (Carøe and Schultz, 1999), and more recently stochastic dual dynamic integer programming (Zou et al., 2019).

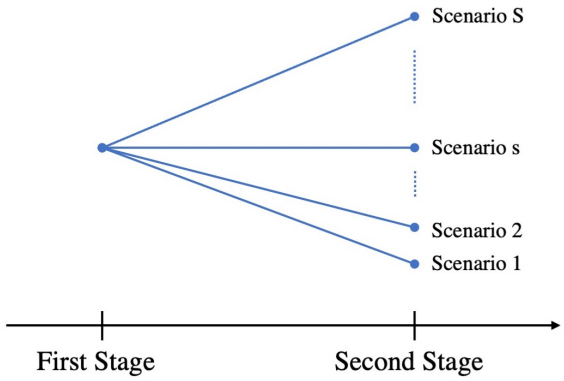


Figure 4.1 Two-stage scenario tree.

Our main contribution is a framework for solving two-stage stochastic optimization problems through developing new reinforcement learning methodologies. We propose using two different reinforcement learning agents for solving each stage of the problem. Agent 2 solves the second-stage problem and is trained before Agent 1, which solves the first-stage problem. This approach prevents the complications that come with multi-agent reinforcement learning training, such as instability, non-stationarity, and different learning speeds. We propose training algorithms for Agent 1 and Agent 2 by refining the well-known policy gradient algorithm REINFORCE (Williams, 1992;

Silver et al., 2014). 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 present our 2SRL framework on the two-stage stochastic knapsack problem, one of the most fundamental and challenging examples of NP-Hard problems. Our research objective is to achieve satisfactory solutions quickly, 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.

4.2 Literature Review

This section introduces the recent progress in solving operations research problems with machine learning, focusing on reinforcement learning. Further, we explain our motivation and contributions in detail.

Reinforcement learning is a branch of machine learning that aims to find actions to maximize a reward. The history of reinforcement learning can be traced back to dynamic programming or Bellman equation from Bellman (1966). For a brief history and a detailed explanation of reinforcement learning, we refer to Sutton and Barto (2018). In the last decade, state-of-the-art for machine learning has been significantly expanded with the rise of deep learning. Even though the idea of using neural networks as function approximators for reinforcement learning is not new, the renewed interest has achieved significant advancements. In a famous example, the deep reinforcement learning agent won the complex game of Go against the human champion by 5 to 0 (Silver et al., 2016). The success of deep reinforcement learning has gained attention from the operations research community, where those techniques are applied to optimization problems. For a review of reinforcement learning for combinatorial optimization problems, we refer to Mazyavkina et al. (2021). For

a review of machine learning for combinatorial optimization problems, we refer to Bengio et al. (2021).

The usage of reinforcement learning for operations research problems is an active research area. In their pioneering study, Bello et al. (2016) 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 (A3C) algorithm. Additionally, the authors introduce two different strategies during inference time by keeping a pool of candidates. Motivated by Bello et al. (2016), Nazari et al. (2018) present a framework for solving the vehicle routing problem, which has a set of dynamic features in addition to static features differently from Bello et al. (2016). Authors propose simplifying the pointer networks to handle static and dynamic features by eliminating the encoder, replacing it with an embedding layer, and attending over the input embeddings along with the decoder hidden state to select an input element when decoding. Also, the usage of embeddings instead of a recurrence ensures input invariance. The results show that the reinforcement learning-based solution outperforms classical heuristics and a specialized solver. Furthermore, the authors present the results for the stochastic vehicle routing problems, and the deep reinforcement learning agent learns better policies than practical baselines. In a similar motivation, Hu et al. (2017) present a reinforcement learning-based methodology using pointer networks to solve the three-dimensional bin packing problem, which is NP-Hard, and achieve a 5% improvement over heuristics.

Khalil et al. (2017a) explore solving combinatorial optimization problems that can be formulated as graphs with reinforcement learning. They utilize a graph embedding network called structure2vec to represent the nodes in the graph, along with a deep Q-learning algorithm. The featured framework is shown to be effective at solving minimum vertex cover, maximum cut, and traveling salesperson problems.

Deudon et al. (2018) attempt to solve the traveling salesperson problem with reinforcement learning, but they rely on a multi-head attention mechanism instead of recurrence, as in Bello et al. (2016). Also, they strengthen their framework with a local search algorithm to improve the solution found by their reinforcement learning agent. Kool et al. (2018) present a reinforcement learning-based framework to learn solutions to combinatorial optimization problems. Their model includes an encoder with a multi-head attention mechanism similar to Deudon et al. (2018), but their decoder is different, and they utilize a greedy rollout baseline. 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 (2019) 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 by choosing a subset of the solution and then applying a rewriting rule to the selected subset. Lu et al. (2019) fuse the power of heuristics with reinforcement learning and present a learn-to-improve framework. Similar to Chen and Tian (2019), they start with a feasible solution and improve it iteratively by choosing a class of the operator and then the operator itself. Tang et al. (2020) present a framework for solving integer programs through cutting planes which is the core of many modern solvers. They present a reinforcement learning formulation and a model for selecting cutting planes. The results show that the agent can generalize to instances of different sizes and types. Also, the trained agent can be used with a branch-and-cut algorithm, which is the core of commercial solvers. He et al. (2021) 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 their cyclical framework. Secondly, a mixed-integer process based on a

constructive heuristic or dynamic programming is performed. The results show that such integrated methodologies have a promising potential for solving combinatorial optimization problems, which are often very hard to solve. Li et al. (2021) present a framework for solving multiobjective optimization problems using reinforcement learning. Their methodology involves decomposing the problem into subproblems, each collaboratively solved by a learning agent using a parameter transfer strategy during the training phase. Authors use pointer networks along with an actor-critic training algorithm. The results outperform the classical solution algorithms. Hubbs et al. (2020) present a comprehensive study where various types of operations research problems are solved with reinforcement learning and state that it can outperform classical heuristics in many cases. Also, an open-source library named OR-Gym is presented for further exploration by researchers.

Kong et al. (2018) train reinforcement learning agents to solve online combinatorial optimization problems. They demonstrate the capability of deep reinforcement learning on three different problems, including an online knapsack problem, and show that trained agents can make decisions that are consistent with specially-designed classical algorithms. Balaji et al. (2019) apply a deep reinforcement learning-based solution approach for three classical online stochastic optimization problems and present benchmark instances. They utilize off-the-shelf reinforcement learning methods, and the trained agent is superior to or competitive with the established baselines.

Afshar et al. (2020) suggest using a stage aggregation strategy to reduce the state space of the knapsack problem, which leads to learning faster and better solutions. The trained agent can be used for smaller instances without additional training. Gu et al. (2020) 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. (2020)

present a framework where action selection during policy evaluation is formulated as a mixed-integer program. The proposed framework is applied to the capacitated vehicle routing problem and has been shown to be competitive with existing reinforcement learning-based approaches. Bushaj and Büyüktahtakın (2022) present a framework for solving the multi-dimensional knapsack problem. Their framework starts with a heuristic to improve the performance of the deep reinforcement learning agent by ordering the items based on their importance. Then they run the k-means algorithm for the constraints to find a good initial solution. 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.

Also, the solution algorithms using machine learning methodologies have gained attention for tackling two-stage stochastic programs. In a recent study, Frejinger and Larsen (2019) 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. Similarly, Larsen et al. (2022b) predict an even less detailed solution for the same two-stage problems as Frejinger and Larsen (2019) using multilayer perceptrons. Abbasi et al. (2020) 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. (2021) 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. (2018) attempt to solve a two-stage stochastic problem using clustering to generate scenarios and utilize recurrent neural networks to generate probabilities for scenarios. Bengio et al. (2020) 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. Dumouchelle et al. (2022) propose a framework to solve two-stage stochastic

programs. First, they train a network to process scenarios of the model. Then a surrogate formulation based on a neural network is utilized to estimate the recourse cost during the solution.

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, including stochastic programming. 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 an expert and can be time-consuming.

4.2.1 Key Contributions of the Study

We fill the described research gap by providing the 2SRL framework for solving two-stage stochastic problems with deep reinforcement learning and demonstrating our methodology on a stochastic version of the well-known knapsack problem. Our objective with this study is to present a general reinforcement learning-based framework to generate high-quality solutions quickly without the need for developing a special solution methodology. Our study is motivated by the wide applicability of two-stage scenario-based stochastic programs and the promising results of reinforcement learning for combinatorial optimization. Our contributions are described next.

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 to solve each stage of the problem. In our 2SRL framework, Agent 2 is trained before Agent 1 to solve second-stage subproblems given any first-stage decision. This approach has a flavor of cutting plane algorithms in traditional solution approaches, but it is

substantially different in providing a reinforcement learning algorithm rather than a cutting plane approach. In addition, 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. 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 our detailed training algorithm to show how actor and critic networks of Agent 2 are used during Agent 1 training.

We present the quality of our 2SRL framework by presenting the time improvement factor and optimality gap compared to a commercial solver, state-of-the-art classical solution methodology, heuristics, and a random solution approach. 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 to many scenarios, highlighting our approach's computational impact.

4.3 Two-stage Stochastic Knapsack Problem

In this section, we introduce the mathematical formulation for the two-stage stochastic knapsack problem with its scenario formulation and give a brief overview of its classical solution methodologies. The variations of stochastic knapsack problems often arise in the industry with application in portfolio selection (Morita et al., 1989), telecommunications (Chius et al., 1996), and transportation planning (Cohn and Barnhart, 1998).

The stochastic knapsack has different formulations that include quadratic objective (Lisser and Lopez, 2010), probabilistic capacity constraints (Gaivoronski

et al., 2011), and risk-aversion measures (Merzifonluoglu and Geunes, 2021). The multi-dimensional knapsack problem is considered to have a very general structure to binary and integer problems and is commonly used to demonstrate the computational performance of stochastic programming solution algorithms as in Angulo et al. (2016) and Büyüktahtakın (2022). We focus on the maximization version of the knapsack problem, where the aim is to find a subset of items such that the sum of their sizes is not more than capacity and the sum of their values is maximized. The two-stage stochastic knapsack problem can be presented in the form of:

$$\max_{x\in\{0,1\}^{n_1}} \quad c^\intercal x + \mathbb{E}_\xi[Q(x,\xi)] \tag{4.1a}$$

$$\text{s.t.} \quad Ax \leq b, \tag{4.1b}$$

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 P. 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 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 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:

$$\max_{y\in\{0,1\}^{n_2}} \quad q^\intercal y \tag{4.2a}$$

$$\text{s.t.} \quad Tx + Wy \leq h. \tag{4.2b}$$

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:

$$\max_{x \in \{0,1\}^{n_1}, y^s \in \{0,1\}^{n_2} \forall s \in \{1,\ldots,S\}} \quad c^\intercal x + \sum_{s=1}^{S} p^s q^{s\intercal} y^s \tag{4.3a}$$

$$\text{s.t.} \quad Ax \leq b \tag{4.3b}$$

$$T^s x + W^s y^s \leq h^s, \tag{4.3c}$$

where p^s represents the probability of realization for scenario s. The linear equivalent of the problem can be very large with the increasing number of scenarios to solve directly, even with state-of-the-art solvers such as CPLEX or Gurobi. Therefore specially-designed solutions algorithms are usually required for solving two-stage stochastic knapsack and two-stage stochastic problems in general.

Traditional solution approaches for solving two-stage stochastic problems include the L-shaped method (Laporte and Louveaux, 1993; Angulo et al., 2016), dual decomposition (Carøe and Schultz, 1999; Lubin et al., 2013), branch-and-bound (Ahmed et al., 2004), progressive hedging (Rockafellar and Wets, 1991; Gade et al., 2016), and Gomory cuts (Gade et al., 2014). More recently, stochastic dual dynamic integer programming (SDDiP) was proposed by Zou et al. (2019) 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 on two-stage stochastic programs and general solution approaches, we refer to Birge and Louveaux (2011) and Küçükyavuz and Sen (2017).

4.4 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 to provide a background on the actor-critic algorithm in Section 4.4.1. Then, in Section 4.4.2, we make a clear explanation of each agent's communication and information exchange during training by giving the detailed algorithm based on REINFORCE (Williams, 1992) together with the scenario sampling strategy.

4.4.1 Pointer Networks

As proposed by Vinyals et al. (2015b), pointer networks have been essential for many different tasks, including text summarization (See et al., 2017), intelligent code completion (Li et al., 2017), and airline itinerary prediction (Mottini and Acuna-Agost, 2017). 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.

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 (Bahdanau et al., 2014; Luong et al., 2015), 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, and changing the vocabulary might require a training iteration. 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 traveling salesperson problem, a city from the input sequence is selected at each decoding step. 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.

As described, the encoder is a recurrent neural network that processes the input elements of the problem. Long Short-Term Memory or LSTM (Hochreiter and Schmidhuber, 1997) is a gated recurrent neural network architecture that has been a popular choice to be used in encoder and decoder of various tasks. Our neural architecture builds upon the network presented by Bello et al. (2016), but we utilize bidirectional LSTM to better capture the input characteristics.

In our learning paradigm, we resort to an algorithm known as actor-critic training to be described in detail later in Section 4.4.2. 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.

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 (4.1), 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 (4.2), where q_k and W_k represent the problem parameters for item k with sizes 1 and m_2, respectively. Here, $\bar{x}$ 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. The following equations are used to describe Agent 1's learning task that has n_1 items, which may also apply to Agent 2's problem by changing the item number to n_2. The encoder processes the input sequence in forward and backward layers and generates the forward and backward hidden states:

$$\overrightarrow{h}^{e}_{1}, \ldots, \overrightarrow{h}^{e}_{n}, \ldots, \overrightarrow{h}^{e}_{n_1} = LSTM^{e}_{forward}(input_1, \ldots, input_n, \ldots, input_{n_1}) \quad (4.4a)$$

$$\overleftarrow{h}^{e}_{1}, \ldots, \overleftarrow{h}^{e}_{n}, \ldots, \overleftarrow{h}^{e}_{n_1} = LSTM^{e}_{backward}(input_1, \ldots, input_n, \ldots, input_{n_1}) \quad (4.4b)$$

Decoding is done by selecting an item and assigning the value of the corresponding binary variable to 1, one at a time. This is called a decoder step. The decoder uses the input data of the previously selected item m' to generate a hidden state for the current step m:

$$\overrightarrow{h}^{d}_{m} = LSTM^{d}(input_{m'}) \quad (4.5)$$

Glimpse is a computational mechanism that compares the current decoder hidden state $\overrightarrow{h}^{d}_{m}$ with the encoder hidden states and takes a linear combination of the encoder hidden states to use in the pointing mechanism. We concatenate the forward and backward hidden states of the encoder $[\overrightarrow{h}^{e}_{n}, \overleftarrow{h}^{e}_{n}]$ to use in both the glimpse and pointer mechanism. For the glimpse computation at step m, attention

scores are calculated using the learned set of parameters s^g as:

$$a_n = \frac{exp(s^g([\overrightarrow{h}^e_n, \overleftarrow{h}^e_n], \overrightarrow{h}^d_m))}{\sum_{n'=1}^{n_1} exp(s^g([\overrightarrow{h}^e_{n'}, \overleftarrow{h}^e_{n'}], \overrightarrow{h}^d_m))} \quad \forall n = 1, \ldots, n_1. \tag{4.6a}$$

During the decoding, we employ a feasibility mask to only select feasible items. Items are considered to be infeasible if they are selected in a previous decoding step or violate the constraints by exceeding the capacity once they are added. Here, v_g, W^g_{ref}, and W^g_q are learned weights of the model. The similarity score for item n is calculated as:

$$s^g([\overrightarrow{h}^e_n, \overleftarrow{h}^e_n], \overrightarrow{h}^d_m) = \begin{cases} v_g{}^\intercal \tanh(W^g_{ref}[\overrightarrow{h}^e_n, \overleftarrow{h}^e_n] + W^g_q \overrightarrow{h}^d_m) & \text{if feasible} \quad (4.7a) \\ -\infty & \text{otherwise} \quad (4.7b) \end{cases}$$

After the scores are calculated for each feasible item, which is determined based on the previously discussed feasibility mask, glimpse g_m is computed to be used in the pointer mechanism. The glimpse is calculated by taking a weighted average: $g_m = \sum_{n=1}^{n_1} a_n * [\overrightarrow{h}^e_n, \overleftarrow{h}^e_n]$. Glimpsing can increase the performance of the model without a significant computational burden (Vinyals et al., 2015a). Then, for the item selection with the pointer mechanism, similarity scores are calculated for each item n as:

$$s^p([\overrightarrow{h}^e_n, \overleftarrow{h}^e_n], g_m) = \begin{cases} v_p{}^\intercal \tanh(W^p_{ref}[\overrightarrow{h}^e_n, \overleftarrow{h}^e_n] + W^p_q g_m) & \text{if feasible} \quad (4.8a) \\ -\infty & \text{otherwise} \quad (4.8b) \end{cases}$$

The model parameters s^p, v_p, W^p_{ref}, and W^p_q are learned with backpropagation. Finally, for current step m, the probability of selecting item n is calculated as:

$$p_n = \frac{exp(s^p([\overrightarrow{h}^e_n, \overleftarrow{h}^e_n], g_m))}{\sum_{n'=1}^{n_1} exp(s^p([\overrightarrow{h}^e_{n'}, \overleftarrow{h}^e_{n'}], g_m))} \quad \forall n = 1, \ldots, n_1. \tag{4.9a}$$

4.4.2 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.

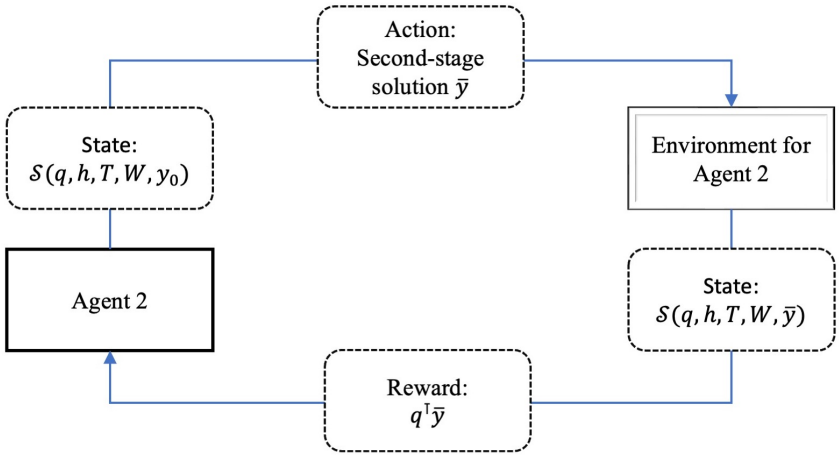
In our 2SRL framework, we opted to use two agents that make decisions about their respective stages instead of having a single agent make decisions for both stages. In many two-stage stochastic problems, the set of decisions that need to be made in the first stage can be highly different from second-stage decisions. 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. We demonstrate our framework through a multi-dimensional two-stage stochastic knapsack problem as described in Section 4.3, where a set of binary decisions is taken for both stages. Even if we have similar type of decision variables, by having a two-agent framework, we can solve problems with a different number of constraints in each stage.

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. In addition to the second-stage feedback, Agent 2 is separately pre-trained before Agent 1 for various reasons. Firstly, stability can be challenging when two agents act in the same environment, especially when they are competing (Buşoniu et al., 2010). 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,

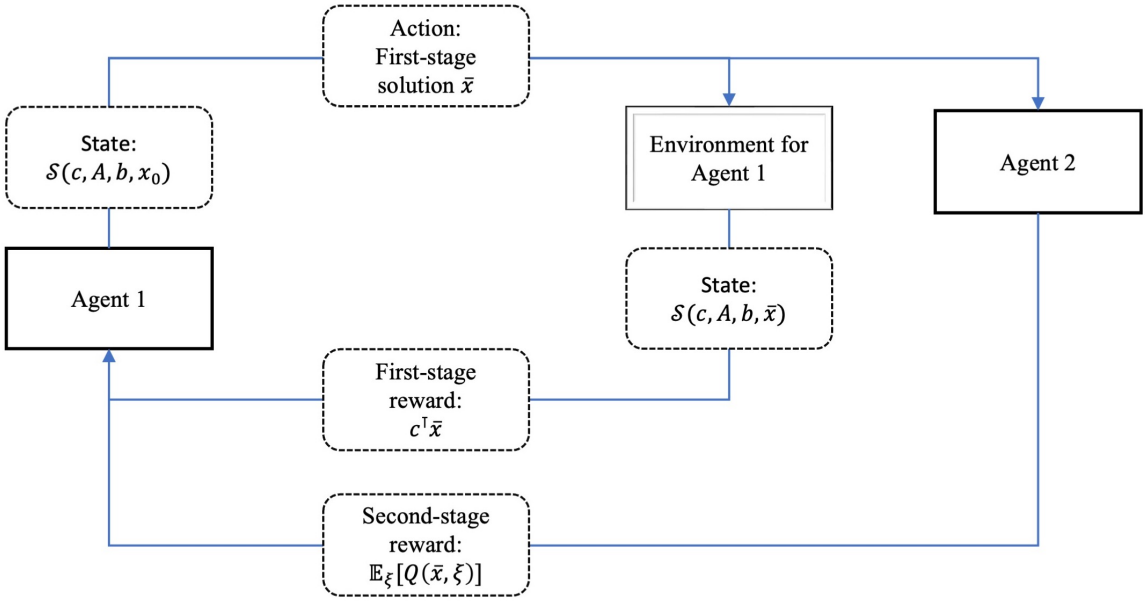
and this might result in competition between them. To eliminate that possibility, we train both agents sequentially. Furthermore, the non-stationarity of multi-agent environments is a challenge in multi-agent reinforcement learning since each agent is learning simultaneously. Each agent tries to solve a moving-target problem meaning that the optimal policy for an agent changes as the other agents' optimal policy changes (Buşoniu et al., 2008; Nguyen et al., 2020). In our framework, we handle this prospect by learning sequentially. Agent 1 starts learning only after Agent 2 has fully trained. Also, the varying learning speed of different agents increases the need for hyperparameter tuning and coordination between agents (Buşoniu et al., 2010). Furthermore, the literature on multi-agent deep reinforcement learning has been only recently developing. To combat the complications mentioned above, our agents are trained successively rather than simultaneously.

Figure 4.2 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 4.2a. 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 (4.2) 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 is 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.

The training of Agent 1 is presented in Figure 4.2b. 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



(a) Agent 2 training overview.



(b) Agent 1 training overview.

Figure 4.2 2SRL training overview.

the first stage involves the minimization of both the first-stage and expected second-stage costs. Here, we aim to make sure that Agent 1 is not just considering the first-stage cost but also is aware that its decisions will impact the reward it is getting from the second stage. Therefore, the decisions made in both stages are connected. 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 (4.1) 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 is 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. In addition, this approach for training Agent 1 presents a paradigm similar to classical solution methodologies such as cutting planes. The main idea of the cutting plane method is to iteratively generate first-stage solutions by approximating the second-stage cost (Ruszczyński and Shapiro, 2003; Rahmaniani et al., 2017). We are inspired from this highly adopted concept and provide Agent 1 with the expected second-stage cost when learning to predict first-stage variables.

Algorithm 2 presents the training procedure used for Agent 2 and Algorithm 3 outlines the training methodology for Agent 1. Here, we resort to the policy gradient algorithm based on REINFORCE (Williams, 1992). REINFORCE algorithm has been used together with an actor-critic training paradigm, which is found to be useful in reducing the variance of gradients (Bello et al., 2016).

Agent 2 and Simulating Subproblems Our aim here is to train Agent 2 that can solve second-stage scenario problems given a set of second-stage problem parameters. However, this raises a complication for Agent 2. Problem (4.2) includes the first-stage decision variables in constraints (4.2b), 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 (4.2b) by the amount of capacity used by the first-stage decisions. To represent this, a set of first-stage coefficients $\mathcal{F} \in [0, 1]$ 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 fixed number of randomly generated two-stage problems given in (4.3) are solved to optimality. Then, using the optimal first-stage decisions x^*, we calculate coefficient $\mathsf{f}^s = \frac{T^s x^*}{h^s}$, where f^s represents the fraction of the capacity used in the first-stage problem in Equation (4.3c) for all $s \in \{1, \ldots, S\}$. Those calculated 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 (4.3c) to simulate the remaining capacity as $h^s(1 - \frac{T^s x}{h^s})$.

Input parameters to Algorithm 2 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 2 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 (4.3) 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, 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

Algorithm 2 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

1: Initialize actor network parameters θ_2^A
2: Initialize critic network parameters θ_2^C
3: **for** epochs = 1 to E **do**
4: **for** steps = 1 to T **do**
5: $KP^i \leftarrow SampleProblem()\ \forall i \in \{1, \dots, B\}$
6: $f^i \leftarrow SampleCoefficient(\mathcal{F})\ \forall i \in \{1, \dots, B\}$
7: $KP_2^{i,s} \leftarrow CalculateSecondStageProblem(KP^i, f^i)\ \forall i \in \{1, \dots, B\},\ \forall s \in \{1, \dots, S\}$
8: $KP_2^{i,s'} \leftarrow SampleScenario(KP_2^{i,s})\ \forall i \in \{1, \dots, B\}$
9: $y^i \leftarrow SampleSolution(p_{\theta_2^A}(.|KP_2^{i,s'}))\ \forall i \in \{1, \dots, B\}$
 Update the actor network:
10: $\tilde{z}_2^i \leftarrow \theta_2^C(KP_2^{i,s'})\ \forall i \in \{1, \dots, B\}$
11: $g_{\theta_2^A} \leftarrow \frac{1}{B} \sum_{i=1}^{B} (z(y^i|KP_2^i) - \tilde{z}_2^i) \nabla_{\theta_2^A} \log p_{\theta_2^A}(y^i|KP_2^{i,s'})$
12: $\theta_2^A \leftarrow ADAM(\theta_2^A, g_{\theta_2^A})$
 Update the critic network:
13: $\mathcal{L}_2^C \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|\tilde{z}_2^i - z(y^i|KP_2^{i,s'})\|_2^2$
14: $\theta_2^C \leftarrow ADAM(\theta_2^C, \nabla_{\theta_2^C} \mathcal{L}_2^C)$
15: **end for**
16: **end for**

coefficient f^i:

$$\max_{y^{i,s} \in \{0,1\}^{n_2}} \quad q^{i,s\intercal} y^{i,s} \tag{4.10a}$$

$$\text{s.t.} \quad W^{i,s} y^{i,s} \leq h^{i,s}(1 - f^i). \tag{4.10b}$$

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 $z(y^i|KP_2^{i,s'})$ 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 (Kingma and Ba, 2014). 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 $z(y^i|KP_2^{i,s'})$. 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.

Training Agent 1 In this subsection, we present the detailed trained algorithm for Agent 1 to solve first-stage problems after Agent 2 is trained to solve second-stage problems. Algorithm 3 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 (4.1a). 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 2; batch size B, number of epochs E, steps per epoch T, and the number of scenarios S are inputs of Algorithm 3. 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 3 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 (4.3) 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:

$$\max_{x^i \in \{0,1\}^{n_1}} \quad c^{i\intercal} x^i \tag{4.11a}$$

$$\text{s.t.} \quad A^i x^i \leq b^i, \tag{4.11b}$$

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

Algorithm 3 REINFORCE for Agent 1

Input: Batch size B, number of epochs E, steps per epoch T, number of scenarios S, trained actor network θ_2^{A*}, trained critic network θ_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**

    $KP^i \leftarrow SampleProblem()\ \forall i \in \{1, \ldots, B\}$

    $KP_1^i \leftarrow CalculateFirstStageProblem(KP^i)\ \forall i \in \{1, \ldots, B\}$

    $x^i \leftarrow SampleSolution(p_{\theta_1^A}(.|KP_1^i))\ \forall i \in \{1, \ldots, B\}$

    $\tilde{z}_1^i \leftarrow \theta_1^C(KP_1^i)\ \forall i \in \{1, \ldots, B\}$

    $KP_2^{i,s} \leftarrow CalculateSecondStageProblem(KP^i, x^i)\ \forall i \in \{1, \ldots, B\},\ \forall s \in \{1, \ldots, S\}$

    $y^{i,s} \leftarrow SampleSolution(p_{\theta_2^{A*}}(.|KP_2^{i,s}))\ \forall i \in \{1, \ldots, B\}, \forall s \in \{1, \ldots, S\}$

    Update the actor network:

    $z_2^i \leftarrow \frac{1}{S} \sum_{i=1}^{S} z(y^{i,s}|KP_2^{i,s})\ \forall i \in \{1, \ldots, B\}$

    $\tilde{z}_2^i \leftarrow \frac{1}{S} \sum_{i=1}^{S} \theta_2^{C*}(KP_2^{i,s})\ \forall i \in \{1, \ldots, B\}$

    $g_{\theta_1^A} \leftarrow \frac{1}{B} \sum_{i=1}^{B} (z(x^i|KP_1^i) + z_2^i - \tilde{z}_1^i - \tilde{z}_2^i) \nabla_{\theta_1^A} \log p_{\theta_1^A}(x^i|KP_1^i)$

    $\theta_1^A \leftarrow ADAM(\theta_1^A, g_{\theta_1^A})$

    Update the critic network:

    $\mathcal{L}_1^C \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|\tilde{z}_1^i - z(x^i|KP_1^i)\|_2^2$

    $\theta_1^C \leftarrow ADAM(\theta_1^C, \nabla_{\theta_1^C} \mathcal{L}_1^C)$

  end for

end for

problem is isolated from the two-stage problem, similarly to step 7 of Algorithm 2. 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:

$$\max_{y^{i,s} \in \{0,1\}^{n_2}} \quad q^{i,s\intercal} y^{i,s} \tag{4.12a}$$

$$\text{s.t.} \quad W^{i,s} y^{i,s} \leq h^{i,s} - T^{i,s} x^i \tag{4.12b}$$

Here, we do not sample scenarios, unlike Algorithm 2, since no training iteration is performed for Agent 2. In step 10, second-stage decision $y^{i,s}$ for each scenario $\forall s \in \{1, \ldots, S\}$ generated for each problem $\forall i \in \{1, \ldots, B\}$ given in the above formulation (4.12) using the actor network θ_2^{A*} of Agent 2 from Algorithm 2. This is achieved by stochastic policy $p_{\theta_2^{A*}}$, trained in Algorithm 2, 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 $z(y^{i,s}|KP_2^{i,s}) = q^{i,s\intercal} y^{i,s}$. 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 θ_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 $z(x^i|KP_1^i) = c^{i\intercal} x^i$. 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.

4.5 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.

4.5.1 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. (2016). 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 $n_1, n_2 \in \{10, 20, 30\}$. Also, the number of resource constraints for the first stage and the number of resource constraints for the second stage are $m_1, m_2 \in \{5, 10, 15\}$. 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 4.1 presents the results with the number of items $n_1, n_2 \in \{10, 15, 20\}$ and $m_1, m_2 = 5$.

Table 4.2 presents the results with the number of items $n_1, n_2 \in \{20, 30, 40\}$ and $m_1, m_2 = 10$. Table 4.3 presents the results with number of items $n_1, n_2 \in \{30, 45, 60\}$ and $m_1, m_2 = 15$. For all results in Tables 4.1, 4.2, and 4.3, 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. (2019) to solve scenario-based stochastic problems involving integers. The SDDiP is considered to be a state-of-the-art solution methodology for a wide range of problems and achieved significant improvements in solution times. Thus, we utilize the SDDiP approach as a benchmark solution method in our experiments for comparison to the 2SRL. The SDDiP is a stochastic nested decomposition algorithm that can solve general two-stage and multi-stage stochastic programs with binary state variables. Recently, it has been used to solve different types of stochastic programming problems, including hydropower scheduling (Hjelmeland et al., 2018), power infrastructure planning (Lara et al., 2020), and lot-sizing (Thevenin et al., 2022). The SDDiP iterates over three different cuts to approximate cost functions: Benders' cuts, integer optimality cuts, and Lagrangian cuts. The algorithm can be stopped if the difference between lower and upper bounds is not improved for a certain number of iterations. We use Python implementation of the SDDiP solution algorithm developed by Ding et al. (2019) together with Gurobi. We limit the SDDiP solution time to 2-hour or 20 stable iterations, whichever comes first. We also utilize two heuristics to solve the knapsack problem presented. First, we utilize an LP-based adaptive-fixing (AF) heuristic by Bertsimas and Demir (2002). For the AF heuristic, we fix 0.1% of variables at each iteration, instead of a single variable, to reduce the computational burden. Second, we utilize the greedy primal effective capacity heuristic (PECH) designed by Akçay et al. (2007). While the heuristics are not specifically designed to solve a two-stage problem, they deliver high-quality solutions. Also, we generate a random feasible solution similar to Bello et al. (2016) for comparison purposes.

4.5.2 Model Architecture

As explained in Section 4.4.1, 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 and 2 layered neural networks with 128 units and ReLU activation function. Also, we utilize a dropout technique with a rate of 0.3 to achieve a better generalization performance (Srivastava et al., 2014).

Similar to Bello et al. (2016), we make use of a softmax temperature with a temperature hyperparameter of 1.5. In this step, logit values calculated in Equation (4.8a) are divided by this predetermined parameter. Also, the logit clipping approach is taken, which is found helpful by Bello et al. (2016) for performance gains. In this step, logit values calculated Equation (4.8a) are clipped between $[-10, 10]$. 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 2. 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. This trade-off is very beneficial since making a forward pass using actor networks is very small compared to very long solution times of two-stage stochastic programs. 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. For results in Tables 4.1, 4.2, and 4.3, we have sampled 100 solutions for the problems with less than or equal to 100 scenarios and sampled 500 solutions for problems that have more than 100 scenarios.

4.5.3 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, heuristics, 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 with a 2-hour solution time limit or until it reaches 20 stable iterations (Ding et al., 2019).
- **timeAF**: Average solution time of test instances using the AF heuristic of Bertsimas and Demir (2002).
- **timePECH**: Average solution time of test instances using the PECH of Akçay et al. (2007).
- **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 4.5.1 *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****, AF heuristic solution for* ***optGapAF****, PECH heuristic solution for* ***optGapPECH****, and random solution for* ***optGapRand****) is given by:*

$$\textbf{optGap2SRL}(\%) = \frac{|obj2SRL - objGRB|}{objGRB} \times 100. \tag{4.13}$$

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

Definition 4.5.2 *The solution time improvement factor **timeImp2SRL** resulting from the 2SRL framework (**timeImpSDDiP** from using the SDDiP, **timeImpAF** from using the AF heuristic, **timeImpPECH** from using the PECH heuristic, and **timeImpRand** from using a random solution) is defined as:*

$$\textbf{timeImp} = \frac{timeGRB}{time2SRL}. \tag{4.14}$$

Definition 4.5.3 *A one-sided Wilcoxon signed-rank test (Wilcoxon, 1945) 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:*

$$H_0 : median(timeAF - time2SRL) < 0 \tag{4.15a}$$

$$H_1 : median(timeAF - time2SRL) > 0 \tag{4.15b}$$

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 AF heuristic.

4.6 Computational Results

This section presents the computational results for the 2SRL, along with a comparison with Gurobi, SDDiP, AF, and PECH heuristics, 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 20 instances for each presented case in each column of Tables 4.1, 4.2, and 4.3. 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 4.1 presents the results for the 2SRL trained with instances having 10 scenarios, 10 items in the first stage, and 10 items in the second stage. The first

dataset given in the second column of Table 4.1 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 when compared to Gurobi. The remaining three sets of instances in Table 4.1 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, heuristics outperform 2SRL in terms of optimality gap, but 2SRL dominates the heuristics 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%. The AF heuristic, however, results in a better average optimality gap of 2.92% but it is significantly slower than 2SRL with an average time improvement of around 4,500. Moreover, the PECH heuristic results in a 5.91% optimality gap but again takes longer than 2SRL and even AF, achieving a time improvement of 229. Therefore, 2SRL can be preferred over the heuristics 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 AF heuristic, which is significantly faster than the PECH. 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 4.2 presents another set of results for 2SRL. Here, the model is trained with 20-item problems with 10 scenarios. The first test set presented in the second column of Table 4.2 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 instances in Table 4.1, but the optimality gaps are

Table 4.1 Average Results of Experiments for 2SRL Trained with 10 Items

#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
timeAF	0.0	0.3	0.5	2.0	4.1	1.4	0.5	1.7
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
timeImpAF	13	16,498	1,123	3,567	1,054	4,451	269	8,032
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
optGapAF(%)	3.18	2.68	3.06	2.68	3.00	2.92	2.43	2.02
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

higher for the second and fourth sets of instances since they have a much larger set of items. We believe this 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 great flexibility for solving instances with a varying number of variables. For all instances in Table 4.2, the p-values are smaller than 0.01, ensuring that 2SRL results in a faster solution than the AF 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 AF heuristic reduces the solution time to only 8.6 seconds, with an impressive gap of 2.44%. However, this solution might not be enough 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. Also, for all cases, the AF heuristic outperforms the PECH heuristic in terms of both solution time and quality.

Table 4.2 Average Results of Experiments for 2SRL Trained with 20 Items

#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
timeAF	0.0	0.5	0.7	4.3	8.6	2.8	0.8	3.4
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
timeImpAF	93	14,967	9,821	1,761	886	5,506	1,785	6,566
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
optGapAF(%)	3.12	1.27	2.68	1.64	2.44	2.23	1.80	1.44
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

Table 4.3 presents the 2SRL results for a harder set of instances. Here, the Agents are trained using 30-item problems with 10 scenarios. The results show that the 2SRL outperforms both heuristics in terms of solution time at the cost of a higher optimality gap. Also, the solution quality of 2SRL deteriorates as the number of items in the test set differs significantly from the number of items that the models are trained with. However, for the largest problem with 1,000 scenarios, 2SRL reduces the solution time to under a second with an optimality gap below 10%. Moreover,

all p-values for the Wilcoxon signed-rank test are smaller than 0.01, which ensures that 2SRL is faster than the AF heuristic. These results highlight the potential of 2SRL for fast solutions while advising caution for the solution quality of instances with significantly different characteristics.

Table 4.3 Average Results of Experiments for 2SRL Trained with 30 Items

#sc	10	50	100	500	1000	Avg	Mdn	Std
#items	30	60	30	45	30	39	30	12
timeGRB	92.4	7,200.0	7,200.0	7,200.1	7,200.7	5,778.7	7,200.0	2,859.7
time2SRL	0.0	0.1	0.0	0.4	0.4	0.2	0.1	0.2
timeSDDiP	2,574.2	7,225.7	5,069.5	7,245.4	7,240.2	5,871.0	7,211.4	2,470.2
timeAF	0.1	1.1	1.2	7.1	11.6	4.2	1.3	4.6
timePECH	0.5	38.2	38.3	2,106.7	3,835.7	1,203.9	40.8	1,572.4
timeRand	0.0	0.0	0.0	0.2	0.3	0.1	0.0	0.1
timeImp2SRL	16,893	98,682	187,307	17,192	16,930	67,401	17,192	71,276
timeImpSDDiP	0	1	2	1	1	1	1	1
timeImpAF	1,513	7,269	6,236	1,048	644	3,342	1,076	3,577
timeImpPECH	199	190	190	3	2	117	16	256
timeImpRand	27,397	238,178	243,354	31,890	24,203	113,004	33,883	113,381
optGapGRB(%)	0.00	0.23	0.41	0.43	0.63	0.34	0.32	0.25
optGap2SRL(%)	12.81	20.27	11.97	15.55	9.87	14.09	13.98	4.27
optGapSDDiP(%)	0.01	0.03	0.01	0.04	0.03	0.03	0.03	0.02
optGapAF(%)	2.45	1.05	1.74	1.30	1.86	1.68	1.33	0.90
optGapPECH(%)	4.36	5.08	4.54	4.75	4.50	4.65	4.17	2.03
optGapRand(%)	49.47	48.16	46.27	47.07	47.98	47.79	47.90	4.29

The results presented in Tables 4.1, 4.2, and 4.3 highlight the strength and potential impact of the 2SRL framework. For all 15 test datasets with different configurations, the 2SRL framework outperforms the AF and PECH significantly in terms of solution times. Therefore, 2SRL creates an opportunity to be used in online applications by sampling a 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 heuristics. Considering the trade-off between the solution quality and solution time, 2SRL finds itself a place in the approaches favoring the solution time.

4.7 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 and, thus 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. By presenting the 2SRL framework, we aim to quickly generate adequate solutions without analyzing problem characteristics. 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. Agent 2 is trained to solve the second-stage problem, and it is trained before Agent 1, which solves the first-stage problem. 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 a first-stage coefficient. Additionally, we have presented a scenario sampling strategy for training that reduces the correlations and improves the training efficiency. We have utilized pointer networks with a feasibility mask that can predict problems with a varying number of items. 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 regardless of the number of items or scenarios, which provides a significant advantage in terms of the generalizability of the results. This flexibility can open 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.

We hope that the presented 2SRL framework pioneers and facilitates 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. Additionally, more experiments in various two-stage programs can be performed to test the robustness of the 2SRL framework. Furthermore, 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. Also, we can utilize a supervised learning paradigm to provide optimal decision variables explicitly along with the reward. Moreover, reinforcement learning agents can be integrated into classical operations research solution approaches, such as Lagrangean and Benders decompositions to improve their performance.

CHAPTER 5

A NON-ANTICIPATIVE LEARNING-OPTIMIZATION FRAMEWORK FOR SOLVING MULTI-STAGE STOCHASTIC PROGRAMS

5.1 Introduction

In this chapter, we present a study at the crossing of Machine Learning (ML) and Operations Research (OR). In recent years, significant effort and interest have been put in the flourishing area of using ML for solving OR problems. While significant results have been achieved, there is still a need for frameworks that can handle special requirements of various OR problems. Here, we focus on this direction and address OR problems involving uncertainty. We present a framework for solving scenario-based multi-stage stochastic programs by combining learning, heuristics, and mathematical solvers.

The uncertainty is crucial in many OR problems and must be addressed for accurate and realistic representations of systems of interest. While the modeling of uncertainty can take many forms, we focus on scenario-based problems, which can be modeled in two or more stages. In general, scenario-based programs are considered to be complex to solve. Multi-stage stochastic optimization models are considered to be much harder than two-stage stochastic models. The sequence ξ_t, $t = 2, \ldots, T$ is the stochastic process, and the decision made at period t is x_t, $t = 1, \ldots, T$. In a multi-stage stochastic program, a decision x_1 is taken before the observation of an uncertainty ξ_2, but then the decision process repeats itself:

$$\text{decide}(x_1) \rightarrow \text{observe}(\xi_2) \rightarrow \text{decide}(x_2) \rightarrow ... \rightarrow \text{decide}(x_T) \rightarrow \text{observe}(\xi_T)$$

The decision x_t is made at period t and may depend on the data of the process up to and including period t, but not on the future because of the non-anticipativity of the unrealized future uncertainty outcomes. Then the T-stage multi-stage stochastic programs can be expressed as:

$$\min_{\substack{A_1x_1=b_1\\x_1\geq 0}} c_1^\intercal x_1 + \mathbb{E}[\min_{\substack{B_2x_1+A_2x_2=b_2\\x_2\geq 0}} c_2^\intercal x_2 + \mathbb{E}[\cdots + \mathbb{E}[\min_{\substack{B_Tx_{T-1}+A_Tx_T=b_T\\x_T\geq 0}} c_T^\intercal x_T]]] \quad (5.1a)$$

where $x_t \in \{0,1\}^{n_t}$. The first-stage matrices c_1, A_1, and b_1 are known with sizes of $n_1 \times 1$, $m_1 \times n_1$, and $m_1 \times 1$, respectively. For the other stages $t = 2, \ldots, T$, matrices c_t, B_t, A_t, and b_t have sizes of $n_t \times 1$, $m_t \times n_{t-1}$, $m_t \times n_t$ and $m_t \times 1$, respectively. At stage t, the number of variables is $n_t \in \mathbb{Z}^+$ and the number of constraints is $m_t \in \mathbb{Z}^+$ for all $t = 1, \ldots, T$. Some or all of these matrices can contain uncertainty.

Multi-stage stochastic programs are very practical in modeling various applications, including airline revenue management (Möller et al., 2008), capacity planning (Huang and Ahmed, 2009), epidemic control planning (Yin and Büyüktahtakın, 2021; Yin et al., 2023; Yin and Büyüktahtakın, 2022), invasive species control and surveillance (Kıbış et al., 2021; Bushaj et al., 2021), and risk-averse optimization (Homem-de Mello and Pagnoncelli, 2016; Bushaj et al., 2022a). Despite their modeling power, they pose a computational challenge and usually require specially designed algorithms and heuristics. Exact solution approaches include Lagrangian relaxation (Chen et al., 2002), Dantzig-Wolfe decomposition (Singh et al., 2009), and branch-and-price (Lulli and Sen, 2004).

In this study, we propose a learning strategy and a testing framework for solving multi-stage stochastic programming problems in a fast setting. OR problems are solved repeatedly on daily bases or more frequently in various areas including but not limited to logistics (Schmidt and Wilhelm, 2000), energy (Vespucci et al., 2012), healthcare (Guerriero and Guido, 2011), and air transport industries (Barnhart et al., 2003). Stochastic programs can be used to generate realistic pictures of systems and provide benefits over deterministic problems (Zakaria et al., 2020). However, their size and solution time can limit their benefit. Hence, problem-specific methods are often required for applications that are solved frequently. This process requires an expert

and can be time-consuming. We address this issue and provide a general learning framework for solving scenario-based multi-stage stochastic programs.

Furthermore, multi-stage stochastic problems require a property of non-anticipativity. Simply put, it ensures that we cannot use information from future periods. The non-anticipativity constraints are fundamental for multi-stage stochastic problems, and they can grow rapidly with the number of scenarios. The underlying stochastic process can be represented with scenarios, with a finite number of realizations. The framework that we are proposing is specifically designed to handle scenario-based multi-stage problems and non-anticipativity requirements. We propose a new type of neural network: Non-anticipative Encoder-Decoder with Attention or NEDA, where the Encoder-Decoder with Attention part of the algorithm is based on the neural translation architecture of Luong et al. (2015).

We propose the ScenPredOpt learning-optimization framework, building on the PredOpt developed in Chapter 3 for deterministic multi-period problems. The PredOpt utilizes an encoder-decoder model with attention to learning from the solutions of deterministic multi-stage problems. The PredOpt framework also eliminates the infeasibility challenge arising from using predictions partially in Chapter 2. The PredOpt framework predicts the values of both binary decision variables and tight or close-to-tight constraints. Those identified tight constraints are used to create a relaxation of the model, which is used to find a level of predictions that do not cause an infeasible solution. Then, once a suitable level is found, the problem is solved full-scale to generate the solution. The solution time can be reduced by three orders of magnitude with a small optimality gap below 0.1%. Furthermore, models can predict longer instances with more items than they are trained with.

This study presents NEDA to tackle to challenge of non-anticipativity of the predictions by reconstructing the attention calculations of the model presented in Luong et al. (2015). NEDA ensures that at each node, the same decision will be

made for a subset of scenarios that share that same node of the scenario tree at any stage t. Also, a new training paradigm is developed for the proposed NEDA based on deterministic instances that are much easier to solve than scenario-based stochastic instances. We utilize the presented framework by enabling a heuristic to improve the solution time further. The heuristic can capture solution characteristics of a stochastic problem that are not identified by a learning model since it is trained with deterministic instances. The heuristic based on linear programming (LP) further reduces the solution time while still maintaining solution quality. Also, we present a new and improved item-wise expansion strategy.

We present the computational results for ScenPredOpt through stochastic versions of two fundamental OR problems: capacitated lot-sizing and knapsack. In the stochastic multi-item capacitated lot-sizing (SMCLSP), several items are considered for production in a planning horizon. The aim is to minimize the sum of production, setup, and inventory costs while satisfying the demand. The lot-sizing is considered to be central in production planning with a diverse range of variations and applications (Jans and Degraeve, 2008). The stochastic multi-stage multi-dimensional knapsack (SMSMK) is a dynamic and stochastic version of the classical knapsack problem. The objective is to find stable solutions by maximizing the profit and stability bonus while satisfying the resource requirements. The stability of the solutions over time can be highly crucial for many cases since frequently changing decisions can be costly or unimplementable. Some examples where stability might be required include generating human-interpretable dashboards, routing, anomaly detection, and audience-aware advertising (Cohen et al., 2016). Even though we present the computational results of our framework for these two problems, we do not modify our solution framework for them. Therefore, our framework is general and can be applied to other scenario-based multi-stage stochastic programs.

The main research contribution of this study is a prediction-optimization framework that can solve scenario-based multi-stage stochastic programs. We address the challenge of non-anticipativity with a new neural architecture called NEDA. A training mechanism based on deterministic instances is presented to reduce training time significantly. Also, an improved framework, ScenPredOpt, is presented to utilize predictions from the neural network model. ScenPredOpt integrates a heuristic based on LP relaxation to improve solution time reductions further. Moreover, we propose an improved item-wise generalization algorithm that considers the variability of the predictions. We present the results of the ScenPredOpt framework and compare them with cutting-edge solution algorithms and heuristics. Our goal is to reduce the solution times of multi-stage stochastic programming problems where they are solved repeatedly. Our motivation is to develop a general framework that can solve such problems without a problem-specific design and generate fast and close-to-optimal solutions to solve multi-stage stochastic programs with many scenarios and periods.

5.2 Literature Review

In recent years, the field of ML for solving OR has drawn significant attention. For an excellent review on the subject, we refer to Bengio et al. (2021). Specifically, using ML for solving scenario-based two-stage stochastic programs has gained traction. The closest study to ours is presented by Frejinger and Larsen (2019). They utilize a learning framework based on a neural translation architecture to solve two-stage stochastic programs. Even though they share the same motivation as ours, i.e., reducing the solution times of stochastic programs, their methodology is significantly different from ours. While we present a new neural architecture that can handle non-anticipativity, Frejinger and Larsen (2019) do not consider the non-anticipativity aspect of the problem. Frejinger and Larsen (2019) present a training paradigm based on fully solved stochastic problems, while we present a training strategy based

on deterministic and easy-to-solve instances. Also, Frejinger and Larsen (2019) ensure the feasibility of the predictions by a probability mask, but we present an iterative methodology to select a prediction level that eliminates infeasibility. Additionally, while they predict an averaged solution description for the recourse decision, we output the fully detailed solution for all stages of the problem. In a similar vein, Larsen et al. (2022b) present a framework to solve two-stage stochastic programs. Authors work on a problem in which the solution for the second stage is computationally demanding. Therefore, they predict a higher level and less detailed solution description instead of predicting fully-detailed second-stage decisions using multilayer perceptrons.

Abbasi et al. (2020) present a methodology for solving two-stage stochastic optimization problems using well-known machine learning algorithms and a case study on blood transshipment problems with uncertain demand. Abbasi et al. (2020) train their models on the solutions of fully-solved stochastic problems and predict only the first-stage decision variables. The first-stage variables are directly actionable; the second-stage variables appear to handle uncertainty and are not actionable. They use classical ML algorithms such as classification and regression trees, k-nearest neighbors, random forest, and neural networks. In practice, their solution framework overperforms the existing policy but has an optimality gap of 14%. Wu et al. (2021) utilize a conditional variational autoencoder to solve graph-based two-stage stochastic optimization problems. An encoder is used to generate low-dimensional representations of scenarios, which can be used through a decoder for tasks like scenario reduction and objective prediction. The presented methodology can be used for larger problems and more scenarios than they are trained with. In Crespo-Vazquez et al. (2018), a methodology based on ML is presented to solve the wind and storage power plant participation problem defined as a two-stage stochastic programming model. They utilize multivariate clustering to generate a set of scenarios from

historical data, and the probabilities of scenarios are calculated by a trained Long Short-Term Memory (LSTM) model. Bengio et al. (2020) present a framework to solve two-stage stochastic programs by predicting a representative scenario of all uncertainty. Therefore, the problem can be solved with a representative scenario deterministically instead of all scenarios and achieve the same solution faster than the Gurobi solver. Also, by only generating a surrogate problem, first-stage feasibility is ensured since the surrogate problem is solved with Gurobi in their framework.

Zheng et al. (2021) propose an encoder-decoder-based framework to solve the online route-planning problem that is formulated as a two-stage stochastic program. For supervised learning, the labels are generated by a heuristic algorithm, and probability masking is applied to ensure feasibility. Dumouchelle et al. (2022) utilize neural networks to solve two-stage problems by estimating the expected second-stage cost, which can be done in two ways. In the single-cut version, the expected cost is predicted for a set of scenarios, and in the multi-cut version, the expected cost is predicted for a single scenario. Then, those estimations from the neural network are used in an approximate formulation containing the first-stage decisions. The resulting framework can be used to solve large problems in seconds. Larsen et al. (2022a) present a study for solving two-stage programs by utilizing ML to speed-up hard-to-solve second-stage problems. They propose to use multilayer perceptrons and predict the recourse objective function value when executing the well-known L-shaped method.

In a recent study, Nair et al. (2020) present a learning framework to enhance mixed-integer programming (MIP) solvers using two neural networks. One network is trained to perform multiple partial assignments for its integer variables, which generates smaller sub-problems that can be parallelized. The model is trained using feasible solutions instead of optimal solutions. Another network is trained for variable selection during the branch-and-bound algorithm to generate a bound for

the objective function value with a smaller tree. Shen et al. (2021) present a new methodology to enhance branch-and-bound algorithms using ML. In the first step, a graph convolutional network is trained using optimal solutions to problems that can be expressed as graphs. Then for testing, the trained model outputs a value for each decision variable representing the probability of being in the optimal solution. In the next step, a probabilistic branching technique with a guided depth-first search is proposed to utilize the predicted optimal solutions. Liu et al. (2022) propose a learning framework by predicting the size of the local branching neighborhood. In the first step, the size of branching is predicted as a regression task using graph neural networks. In the second stage, the predicted size is dynamically adapted within the local branching algorithms using a trained reinforcement learning model. The results show that the size parameter can be learned and results in significant performance gains.

Ding et al. (2020) present a study at the intersection of ML and OR. In their framework, a graph convolutional network is trained to predict values of binary decision variables. Then those predictions can either be used to generate a heuristic prediction with a local branching type cut or result in an exact solution with a root branching rule. In a recent study, Jiménez-Cordero et al. (2022) present a methodology for constraint generation that is used to warm-start the solution process. They train an ML model to predict invariant constraints, which is the set of constraints that cannot be removed from the integer programs without changing the feasible region. Shen et al. (2022) aim to improve the solution time of large-scale optimization problems by presenting an ML-based pricing heuristic for the column generation algorithm. The ML model is trained using the optimal solutions to the pricing problem in the column generation algorithm, which is the bottleneck of the algorithm. By having an ML model predicting the optimal solution to the pricing

problem, the efficiency of the branch-and-price is increased significantly, which is an exact method.

In Kotary et al. (2021), authors are interested in learning OR solution using ML when multiple optimal solutions exist. Also, as a result of randomization within the combinatorial optimization techniques, the learning task may face a challenge. To overcome this, the problem of optimal dataset design is introduced with a heuristic to find solutions with the smallest total variation. Paulus et al. (2021) present a framework where integer programming solvers are integrated into neural network architectures as layers that can learn. The main idea is to provide gradients for both the cost terms and the constraints of an integer program. Therefore, the proposed model can learn the cost and constraints of the problem without specifying it explicitly. Huang et al. (2022) suggest a methodology to rank cuts in a cutting plane algorithm since a good set of cuts can significantly reduce the solution time. The main idea is to learn a scoring function that can measure the efficiency of the cuts that can be generalized to other instances. The proposed framework can be used during branch-and-cut algorithms with a solver.

5.2.1 Key Contributions of the Study

An extensive amount of interest has recently been put toward learning to solve optimization problems; however, there is still a lack of research for solving scenario-based multi-stage stochastic programs. Specifically, deep learning-based supervised methodologies can be considered to solve challenges of training, feasibility, and non-anticipativity in multi-stage stochastic programs. Our motivation is two-fold. First, the recent advances at the intersection of machine learning and mathematical optimization programs show a promising direction in solving combinatorial optimization problems (Liu et al., 2022; Larsen et al., 2022b). Second, the vast-applicability and computational difficulties of scenario-based multi-stage

stochastic programs inspire us to tackle the computational complexity of the problem. Our goal is to significantly enhance learning-optimization solution algorithms for scenario-based multi-stage stochastic programs by presenting a general framework. Our contributions are summarized next.

To the best of our knowledge, this is the first study that utilizes an encoder-decoder model to learn the solutions of scenario-based multi-stage stochastic programs and utilize predicted solutions within a mathematical programming solver. We present an innovative attention-based encoder-decoder model called NEDA, in which the hidden states of the models are adjusted based on the scenario tree to ensure the non-anticipativity of the predicted decision variables. We propose a novel training paradigm for NEDA based on deterministic instances and scenario sampling. This strategy prevents solving stochastic optimization problems to generate training labels, which is computationally intractable.

We introduce the ScenPredOpt framework to handle multi-stage programs by building on the PredOpt presented in Chapter 3. In ScenPredOpt, we utilize a general LP-based heuristic to speed up the solution further. The ScenPredOpt framework is designed to handle general scenario-based multi-stage programs with binary variables by integrating decisions made by learning models, heuristics, and commercial solvers. Moreover, we establish an item-wise generalization algorithm to predict for problems with a large number of items by accounting the variability of the prediction. We test the success of the ScenPredOpt algorithm with a varying number of periods, items, and scenarios. Results show that it outperforms heuristics, and the solution time can be improved up a factor of 599 with a gap of only 0.08%.

Section 5.3 presents the formulation of SMCLSP and SMSMK and a brief review discussion of the traditional solution approaches. Section 5.4 outlines the details of the NEDA, scenario sampling-based training, ScenPredOpt framework, item-wise generalization algorithm. Then in Section 5.5, implementation details, including

instance generation, training, and evaluation, are presented. Section 5.6 demonstrates the computational results along with generalization experiments. Finally, Section 5.7 concludes the study with suggestions for future directions.

5.3 Problems

5.3.1 Stochastic Multi-item Capacitated Lot-Sizing Problem

Lot-sizing applications have been central for many industries, including but not limited to glass, chemical, pharmaceutical, steel, paper, and manufacturing. The lot-sizing problem has many variations that include setup times, multiple machines, cyclical schedules, and perishable inventories (Jans and Degraeve, 2008). In this study, we present a scenario-based version of the classical MCLSP. Here, some or all parameters of the problem can be uncertain and can be represented with scenarios. The objective is to minimize the total cost of production, setup, and inventory costs, while satisfying the demand for each item under all scenarios. The MCLSP, therefore, and the SMCLSP is NP-Hard (Bitran and Yanasse, 1982).

SMCLSP can be expressed as an MIP, with the number of periods T, items I, and scenarios S. The parameters of the problems are assumed to be nonnegative and as follows: production cost p_{it}^s, setup cost f_{it}^s, inventory cost per unit h_{it}^s, demand d_{it}^s, and capacity c_t^s $\forall i \in \{1,\ldots,I\}$, $\forall t \in \{1,\ldots,T\}$, $\forall s \in \{1,\ldots,S\}$. The probability of each scenario is represented by ω^s. The set of scenarios that share the same scenario path with scenario s up to and including period t for item i is represented with Ψ_{it}^s. Nonnegative continuous variables x_{it}^s and v_{it}^s represent the produced and inventory units at the end of each period, respectively. The binary variable y_{it}^s takes value 1 if item i is produced at period t for scenario s, and 0 if not. The SMCLSP formulation:

$$\min \sum_{s=1}^{S} \omega^s \sum_{i=1}^{I} \sum_{t=1}^{T} (p_{it}^s x_{it}^s + f_{it}^s y_{it}^s + h_{it}^s v_{it}^s) \tag{5.2a}$$

$$\text{s.t. } v_{i,t-1}^s + x_{it}^s - d_{it}^s = v_{it}^s \quad \forall i = 1,\ldots,I,\ \forall t = 1,\ldots,T,\ \forall s = 1,\ldots,S \tag{5.2b}$$

$$\sum_{i=1}^{I} x_{it}^{s} \leq c_t^s \quad \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S \tag{5.2c}$$

$$x_{it}^{s} \leq y_{it}^{s} c_t^s \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S \tag{5.2d}$$

$$x_{it}^{s} = x_{it}^{s'} \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S, \ s' \in \Psi_{it}^{s} \tag{5.2e}$$

$$y_{it}^{s} = y_{it}^{s'} \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S, \ s' \in \Psi_{it}^{s} \tag{5.2f}$$

$$v_{it}^{s} = v_{it}^{s'} \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S, \ s' \in \Psi_{it}^{s} \tag{5.2g}$$

$$x_{it}^{s}, v_{it}^{s} \geq 0 \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S \tag{5.2h}$$

$$y_{it}^{s} \in \{0, 1\} \quad \forall i = 1, \ldots, I, \ \forall t = 1, \ldots, T, \ \forall s = 1, \ldots, S. \tag{5.2i}$$

The objective function (5.2a) minimizes the expected cost of production, setup, and inventory over all scenarios, items, and periods. Constraints (5.2b) assure the flux on inventory in a periodical setting, while demand is satisfied for each item i. Constraints (5.2c) limit the produced amount for each item by a shared capacity for all items, and constraints (5.2d) administer setup cost for item i produced in period t. Constraints (5.2e), (5.2f), and (5.2g) are non-anticipativity constraints for x_{it}^{s}, y_{it}^{s}, and v_{it}^{s}, respectively. Then constraints (5.2h) establish the nonnegativity of x_{it}^{s} and v_{it}^{s}. Finally, constraints (5.2i) put binary restrictions for y_{it}^{s}.

The variations of lot-sizing problem and solution approaches have been studied extensively in the literature. Notably, an exact solution framework was presented by Florian et al. (1980). Further, another important exact approach with valid (ℓ,S) inequalities and a separation algorithm was introduced by Barany et al. (1984). More recently, Büyüktahtakın et al. (2018b) present dynamic programming-based inequalities to improve solving the multi-item capacitated lot-sizing problem. We refer to Pochet and Wolsey (2006) for a review on lot-sizing. For general information on stochastic lot-sizing problems, we refer to the review by Tempelmeier (2013). Stochastic lot-sizing problems can be much more challenging, and thus solution approaches such as dynamic programming (Huang and Küçükyavuz, 2008) and

progressive hedging (Haugen et al., 2001) are used. Also, more general approaches that are specifically developed the solve multi-stage scenario-based programs can be utilized to solve SMCLSP. One of the most recent and highly successful approaches is developed by Zou et al. (2019). Their developed SDDiP framework utilizes different types of cutting planes and achieves a state-of-the-art solution framework for solving multi-stage stochastic programs in different settings (Lara et al., 2020; Yu and Shen, 2020). Heuristic approaches based on the relax-and-fit approach are highly used for solving both deterministic and stochastic versions of the MCLSP (Helber and Sahling, 2010; Toledo et al., 2015; Absi and van den Heuvel, 2019; Beraldi et al., 2006).

5.3.2 Stochastic Multi-stage Multi-dimensional Knapsack Problem

The stochastic multi-stage multi-dimensional knapsack problem is a periodical version of the well-known knapsack problem that also includes uncertainty. In this setting, the aim is to keep a stable solution over time by maximizing the profit and stabilization bonus while ensuring the capacity constraints are not violated. Even in a single-dimensional setting, the problem is NP-Hard (Bampis et al., 2022). Consideration of stability over time plays a vital role in applications such as periodically changing prices, energy, raw materials, and resources.

SMSMK is formulated as a binary integer program for a number of periods T, items I, and scenarios S. The binary variable x_{it}^s denotes the decision of inserting item $i \in \{1, \ldots, I\}$, at period $t \in \{1, \ldots, T\}$, in scenario $s \in \{1, \ldots, S\}$ by taking a value of 1, and 0 otherwise. Binary variable y_{it}^s is introduced as the stabilization bonus and assigned a value of 1 if the decision at period t and $t+1$ are identical, i.e. $x_{it}^s, x_{i,t+1}^s = 0$, or $x_{it}^s, x_{i,t+1}^s = 1$. Otherwise, y_{it}^s takes value 0. For item i, period t, and scenario s profit is denoted by p_{it}^s, and the bonus is denoted by b_{it}^s except for $t \neq T$ since $T-1$ is the last period the stability bonus is added. The problem considers J different knapsack constraints, and weights are denoted by w_{ijt}^s for each

item $i \in \{1,\dots,I\}$, knapsack constraint $j \in \{1,\dots,J\}$, period $t \in \{1,\dots,T\}$, and scenario $s \in \{1,\dots,S\}$. For each resource constraint, the capacity of the knapsack is denoted by c^s_{jt}. Ψ^s_{it} is the set of scenario indexes that share the same path with scenario s up to and including period t for item i. Also, ω^s is the probability of each scenario. We modify the formulation in Bampis et al. (2022) to include scenarios and present the problem as:

$$\max \sum_{s=1}^{S} \omega^s \Big(\sum_{i=1}^{I}\sum_{t=1}^{T} p^s_{it}x^s_{it} + \sum_{i=1}^{I}\sum_{t=1}^{T-1} b^s_{it}y^s_{it}\Big) \tag{5.3a}$$

$$\text{s.t.} \sum_{i=1}^{I} w^s_{ijt}x^s_{it} \le c^s_{jt} \quad \forall t = 1,\dots,T,\ \forall j = 1,\dots,J,\ \forall s = 1,\dots,S \tag{5.3b}$$

$$y^s_{it} \le -x^s_{i,t+1} + x^s_{it} + 1 \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T-1,\ \forall s = 1,\dots,S \tag{5.3c}$$

$$y^s_{it} \le x^s_{i,t+1} - x^s_{it} + 1 \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T-1,\ \forall s = 1,\dots,S \tag{5.3d}$$

$$x^s_{it} = x^{s'}_{it} \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T,\ \forall s = 1,\dots,S,\ s' \in \Psi^s_{it} \tag{5.3e}$$

$$y^s_{it} = y^{s'}_{it} \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T-1,\ \forall s = 1,\dots,S,\ s' \in \Psi^s_{it} \tag{5.3f}$$

$$x^s_{it} \in \{0,1\} \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T,\ \forall s = 1,\dots,S \tag{5.3g}$$

$$y^s_{it} \in \{0,1\} \quad \forall i = 1,\dots,I,\ \forall t = 1,\dots,T-1,\ \forall s = 1,\dots,S. \tag{5.3h}$$

The objective function (5.3a) maximizes the expected profit and stability bonus over all scenarios. Constraints (5.3b) are knapsack constraints that limit the weights of selected items by capacity. Following constraints (5.3c) and (5.3d) secure the enforcement of bonus if $x_{it} = x_{i,t+1}$. Precisely, they represent the linear equivalent of $y^s_{it} = 1 - |x^s_{i,t+1} - x^s_{it}|$. Constraints (5.3e) and (5.3f) ensure the non-anticipativity for x^s_{it} and y^s_{it}, respectively. Finally, binary restrictions for x^s_{it} and y^s_{it} are given in constraints (5.3g) and (5.3h).

The knapsack problem has been studied in the OR literature extensively in various forms. The knapsack problem has a wide range of applications, and many types of complex OR problems can be expressed as knapsack-type sub-problems

(Varnamkhasti, 2012). Examples include multiple objectives (Ishibuchi et al., 2014), online knapsack (Cygan et al., 2016), stochastic knapsack (Kosuch and Lisser, 2011), and probabilistic constraints (Gaivoronski et al., 2011). Numerous exact and heuristic approaches have been developed to solve different versions of the knapsack problem. We refer to Cacchiani et al. (2022) for a discussion on recent advances in knapsack problems. As there are not many solution approaches specially developed for SMSMK (5.3a)-(5.3h), we utilize a well-known solution framework, namely progressive hedging (PH), for scenario-based multi-stage problems. Introduced by Rockafellar and Wets (1991), PH uses the idea of relaxing non-anticipativity constraints and solves sub-problems independently. Later, a punishment for violating non-anticipativity is added to the objective for each sub-problem. PH performs well in various stochastic integer problems (Gul et al., 2015; Veliz et al., 2015).

5.4 Methodology

This section presents the details of the non-anticipative neural machine translation model designed to predict multi-stage stochastic optimization problems. Then we introduce the ScenPredOpt framework, which builds and improves on the infeasible elimination strategy presented in Chapter 3. The ScenPredOpt combines learning-based decision-making, heuristics, and mathematical optimization solvers and achieves remarkable solution time reductions, as demonstrated in Section 5.6.

5.4.1 Non-anticipative Encoder-Decoder with Attention

A machine translation system is used to translate from one language to another. In recent years, translation systems based on neural networks have gained significant attention and achieved remarkable results (Stahlberg, 2020). In such systems, the sequence $x_1, x_2, \ldots, x_m$ is the input from the source language and is translated to the target language by generating the output sequence $y_1, y_2, \ldots, y_n$. The neural machine translation model is trained using pairs of input and output sequence pairs

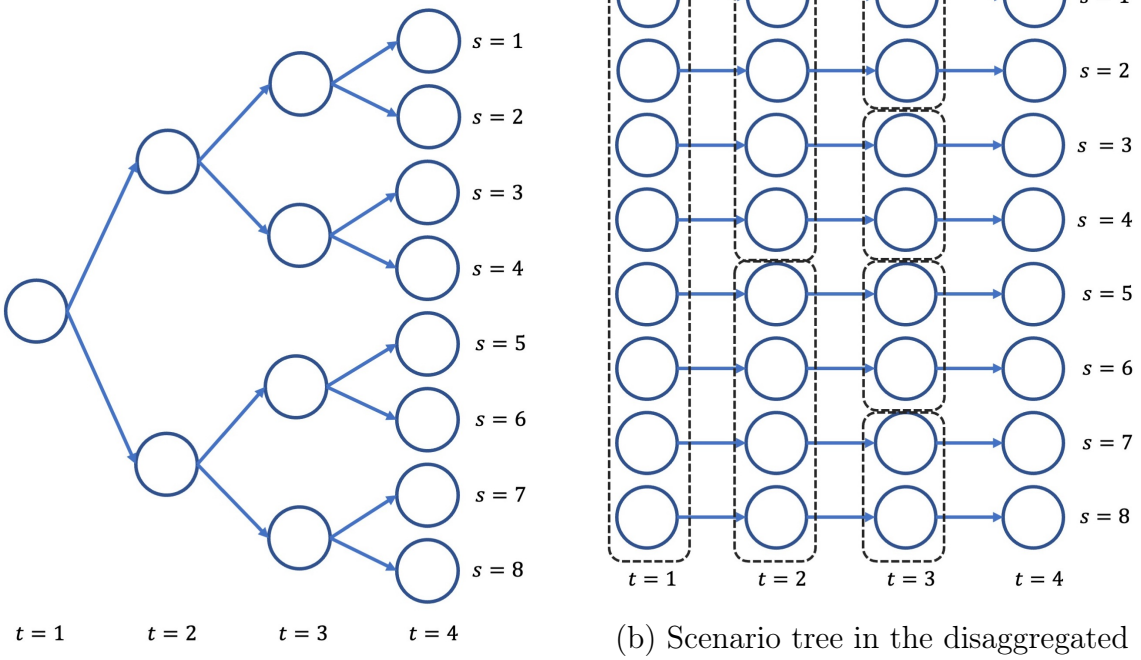
by maximizing the conditional probability: $P(y \mid x) = \prod_{t=1}^{n} P(y_t \mid y_{i|i<t}, x)$. Readers can be referred to Stahlberg (2020) for a detailed review of neural machine translation systems. We develop our NEDA based on the architecture presented by Luong et al. (2015). Its adaptation to predict deterministic multi-stage programs is presented in Chapter 3. Our approach is different than those former approaches by formulating a novel attention mechanism and ensuring the non-anticipativity of the predicted variables.

The neural architecture presented in Luong et al. (2015) includes three main components. The encoder is responsible for processing the input sequence to generate a high-dimensional representation of the input sequence at each time step. Ideally, all characteristics, such as meaning, grammar, and semantics of the input sequence, are captured by this high-dimensional representation. The decoder is responsible for generating the output sequence one word at a time using the representation from the encoder and already translated words. Both the encoder and decoder are recurrent neural networks, usually with the choice of computing units being LSTMs. The link between the encoder and decoder is established through a module called attention, which allows the decoder to focus selectively on crucial information from the encoder. Sequence-to-sequence models are suitable for predicting periodic optimization problems since they can handle information flow between the periods of the problem. This can be a challenge with traditional ML algorithms that do not consider patterns across time. Also, they accomplish good performance in a variety of tasks. For detailed information on the original neural network, we refer to Luong et al. (2015), and for its adaptation, we refer to Chapter 3.

As previously mentioned, Chapter 3 focuses on predicting the decision variables of multi-stage deterministic problems to be integrated with a commercial solver. The PredOpt framework presented in Chapter 3 can reduce the solution time up to three orders of magnitude with optimality gaps lower than 0.1%. However,

the PredOpt framework is not out-of-the-box reconcilable with the scenario-based multi-stage stochastic problems. The main incompatibility between the learning network presented in Chapter 3 and scenario-based multi-stage stochastic problems is the non-anticipative nature of the decision variables. Figure 5.1 demonstrates a scenario tree representation of a problem with four stages, two scenarios per stage, and, therefore, a total of eight scenarios. In Figure 5.1a, the scenario tree is presented in the standard form, and in Figure 5.1b, the scenario tree is presented in disaggregated form. In the scenario formulation of the problem, the variables are defined independently for each scenario, as in Figure 5.1b. However, they are connected with a non-anticipativity constraint since two variables that share the same scenario path up to stage t should be assigned the same decisions for the implementability of the solutions. The grouping of the constraints is shown with dashed lines for each stage in Figure 5.1b. The decision variables should take the same value if they share the same realizations of the scenarios up to the node's stage, which is also known as time consistency. For example, all decisions for scenario one should be identical; therefore, there is only one scenario group. In stage two, there are two scenario groups: The upper group for scenarios one to four and the lower group for scenarios five to eight.

The learning network can generate predictions using the data in tabular and extended formats as presented in Figure 5.1b. However, this raises a critical complication for the problem of interest with non-anticipativity. This is because the attention-based neural network uses the processed input information from both preceding and succeeding periods. The former case is harmless since the information from the preceding periods is the same for all scenarios that have the same parent node. However, for succeeding periods, the child nodes will have different parameters to represent changing scenario data. In effect, the original input data and, therefore, the processed input data are likely to be different for each distinct child node.



(a) Scenario tree in the standard form.

(b) Scenario tree in the disaggregated form.

Figure 5.1 An example scenario tree that contains four stages and eight scenarios. Black dashed rectangular shows the scenario groups with the same decisions.

Since those generated hidden states will be used within the attention mechanism, the predictions can end up being different for each scenario group that requires non-anticipativity. This would lead to a violation of the non-anticipative structure of the problem and would result in an unimplementable decision. For example, we can consider predicting the first scenario group consisting of scenarios one to four in the second stage. The parameters and, therefore, the input data for the first stage would be the same for scenarios one to four. On the other hand, the data of scenarios one and two are identical, and the data of scenarios three and four are identical, considering only stage three. Assuming the model makes a decision by only considering the preceding and succeeding periods, i.e., the attention window of size one, the predictions made would be the same for scenarios one and two and would be the same for scenarios three and four. These two sets of predictions would likely be different from each other at stage two. This would violate the fundamental

non-anticipativity requirement. In this small example, if the size of the attention window is two, then four different predictions can be made for the first scenario group, i.e., scenarios one to four of the second stage. The learner network makes use of input data for the fourth period as well as the third period, which would have different parameters for child nodes. Having this violating condition makes the predictions for the scenario-based multi-stage stochastic program infeasible and inexecutable. In order to eliminate this violation, we propose a novel neural network architecture called Non-anticipative Encoder-Decoder with Attention or NEDA.

The proposed NEDA model is based on the network presented in Luong et al. (2015) and modified to handle the non-anticipative nature of the issue of interest. The main idea is to generate the same encoder hidden state information for all succeeding periods that share the same parent node. As an example, in Figure 5.1b, again, we can consider generating a second-stage prediction for the first scenario group consisting of scenarios one to four. To generate a prediction that ensures non-anticipativity, the hidden states on the next periods within the attention windows should be the same. This is accomplished by averaging the encoder hidden states of child nodes. The encoder processes all input sequences at once and generates the hidden states, which ideally seize the characteristics and features of the problem parameters. The forward encoder layer $LSTM^e_{forward}$ generates the hidden state $\overrightarrow{h}^e_{t,s}$ and the backward encoder layer $LSTM^e_{backward}$ generates the hidden state $\overleftarrow{h}^e_{t,s}$. Given current period t and an attention window length D, we can represent the generated hidden state for each scenario $s \in \{1, \ldots, S\}$ as:

$$\overrightarrow{h}^e_{t-D,s}, \ldots, \overrightarrow{h}^e_{t,s}, \ldots, \overrightarrow{h}^e_{t+D,s} = LSTM^e_{forward}(x_{t-D,s}, \ldots, x_{t,s}, \ldots, x_{t+D,s}) \quad (5.4a)$$

$$\overleftarrow{h}^e_{t-D,s}, \ldots, \overleftarrow{h}^e_{t,s}, \ldots, \overleftarrow{h}^e_{t+D,s} = LSTM^e_{backward}(x_{t-D,s}, \ldots, x_{t,s}, \ldots, x_{t+D,s}) \quad (5.4b)$$

Those hidden states of the forward and backward LSTM layers within the attention window are concatenated and represented as a single encoder hidden state $h^e_{t,s} =$

$[\overrightarrow{h}^{e}_{t,s}, \overleftarrow{h}^{e}_{t,s}]$ $\forall t \in \{t-D, \ldots, t, \ldots, t+D\}$ $\forall s \in \{1, \ldots, S\}$. Let $\Omega_{t,s}$ denote the indices of scenarios that share the same scenario realization up to the current period t with scenarios s. Therefore, for each scenario s in the problem, averaged hidden states $h^{e}_{t,s}$ can be calculated as:

$$h^{e}_{t,s} = \frac{\sum_{s \in \Omega_{t,s}} h^{e}_{t,s}}{|\Omega_{t,s}|} \tag{5.5}$$

The explanation behind this step is to determine all groups of scenarios that have a similar hidden state as the current prediction period t. If those hidden states are close to each other, then the average of the hidden states would be close to each other. In the end, the model would be making a similar prediction of binary variables. However, if the hidden states vary from each other, the trained NEDA model would get varying hidden states throughout the periods. Therefore, the attention scores and context calculated in succeeding steps would lead to differently predicted binary variables for the same scenario clusters violating non-anticipativity. To prevent that, the NEDA model would make an unsure prediction using the averaged hidden states, and the determination of those decision variables would be left to the ScenPredOpt framework, therefore, to the commercial solver to find the best obtainable values.

The decoder cell state is initialized as the scenario average and produces the output sequence. The decoder $LSTM^{d}$ also generates the decoder hidden state $h^{d}_{t,s}$ for the current period t using the decision made in the previous period:

$$h^{d}_{t,s} = LSTM^{d}(y_{t-1,s}) \tag{5.6}$$

Then attention module is used to further incorporate averaged encoder hidden state $h^{e}_{i,s}$ by making a comparison with the current decoder hidden state $h^{d}_{t,s}$. The attention score $a_{i,s}$ is calculated as:

$$a_{i,s} = \frac{exp(score(h^{d}_{t,s}, h^{e}_{i,s}))}{\sum_{t'=t-D}^{t+D} exp(score(h^{d}_{t,s}, h^{e}_{t',s}))} \quad \forall i \in \{t-D, \ldots, t+D\}, \ \forall s \in \{1, \ldots, S\} \tag{5.7}$$

and the score is calculated by the following formula:

$$score(h^d_{t,s}, h^e_{i,s}) = {h^d_{t,s}}^\intercal W_\alpha h^e_{i,s} \quad \forall i \in \{t-D, \ldots, t+D\} \tag{5.8}$$

Using the attention scores, a weighted average of encoder hidden states is taken to calculate the context:

$$cn_{t,s} = \sum_{i=t-D}^{t+D} a_{i,s} \times h^e_{i,s} \tag{5.9}$$

Finally, context $cn_{t,s}$ is concatenated with the decoder hidden state and passed through a linear layer to generate a prediction for scenario s in period t. Note that for all scenarios that share the same scenario realization up to period t, the predicted values would be the same since the predictions are calculated from the same averaged hidden states that are calculated with Equation (5.5).

5.4.2 The ScenPredOpt Framework

Training and Scenario Sampling Our ScenPredOpt framework for stochastic multi-stage problems significantly differs from the PredOpt framework for deterministic multi-period optimization presented in Chapter 3. In PredOpt, the learner is responsible for the decisions being made in each stage in a single deterministic path, but in our approach, the learner is responsible for decisions in all periods to solve problems with a parallel scenario path with non-anticipativity constraints. As previously mentioned, the non-anticipativity of the decisions is ensured with the presented NEDA network.

One of the major challenges arising from using the deterministic PredOpt framework for solving scenario-based multi-stage stochastic programs is the generation of training instances. In Chapter 3, optimal solutions to the training set are generated using a commercial solver. On average, a training problem is generated and solved in a few seconds. The encoder-decoder models require a lot of training data in the

order of millions to learn the optimal solutions. On average, such easy training instances are solved within a few seconds to optimality in high-performance computing clusters. While this process is computationally challenging, it is manageable since the models are trained on shorter and easier instances, which have much less significant processor and memory requirements to solve. The generation of training instances is parallelized to reduce training instance generation time significantly. However, this instance generation approach would be unsuitable for scenario-based multi-stage stochastic programs, even for short-period instances, as they are much more complex and significantly harder to solve than deterministic multi-stage problems. To overcome this computation challenge, we propose a training strategy based on scenario sampling. In this strategy, at each epoch of training, we sample a single scenario for each instance and perform a training step using that scenario data and its optimal solution.

Our proposed training based on scenario sampling achieves a few computational advantages. First, considering the model is trained by the optimal solutions of several millions of instances, it can be computationally infeasible to solve such a large number of multi-stage stochastic problems with multiple scenarios. However, solving the problem with only a single scenario, i.e., a single scenario path in Figure 5.1b, only takes a few seconds with a solver. In this setting, the non-anticipativity constraints in the formulation are removed, and therefore, each subproblem of scenarios can be solved independently in a fast manner. Secondly, by sampling a problem with a single scenario path, we increase the learning efficiency in a similar fashion to experience replay (Mnih et al., 2013). In the scenario formulation, the parameters of a single instance can be very similar if they share a large scenario path, e.g., scenarios 1 and 2 in Figure 5.1a. This powerful correlation can reduce training efficiency and cause instability (Zha et al., 2019). By sampling a single scenario of the individual problem for each epoch, we overcome this challenge. Therefore, the model is not trained on

scenario-based multi-stage problems, but on deterministic multi-stage problems where only a single scenario is considered throughout the planning horizon. For each epoch during training, a problem containing a single scenario and its solution is used for a training step.

Similar to Chapter 3, the model is learned to predict the tight or close-to-tight constraints as well as the binary variables. The aim is to reduce the problem's size; therefore, the solution time during the variables' determination loops is presented in the next section. The tight or close-to-tight constraints are included in the formulation, and others are excluded for faster computation. For the SMCLSP, constraints (5.2c) are labeled as tight if $\sum_{i=1}^{I} x_{it}^{s} \geq \eta c_{t}^{s}$ for a given tightness coefficient $\eta \in [0, 1]$. If $x_{it}^{s} \geq \eta y_{it}^{s} c_{t}^{s}$, the constraints (5.2d) are assumed to be tight. Similarly, for SMSMK, constraints (5.3b) are labeled as tight if $\sum_{i=1}^{I} w_{ijt}^{s} x_{it}^{s} \geq \eta c_{jt}^{s}$. The remaining constraints related to bonus enforcement are removed from relaxation since they do not affect the feasible region. During the training, the model learns tight or close-to-tight constraints, and this information is used during testing to speed up the ScenPredOpt framework.

ScenPredOpt Algorithm A major difference between the PredOpt for deterministic multi-stage problems presented in Chapter 3 and the ScenPredOpt for scenario-based multi-stage problems in Algorithm 4 is the addition of a loop with LP relaxation of the problem. A large number of approaches utilize LP relaxations for solving a multitude of problems involving integers. Commonly, the main idea is to solve a much easier LP relaxation of the problem and fix some of the integer variables in the solution to achieve a reduction in solution time. LP-based heuristics with a variable fixing strategy have been employed extensively in some of the most well-known and extensively studied problems in OR, such as lot-sizing (Denizel and Süral, 2006), knapsack (Chen and Hao, 2014), vehicle routing (Cacchiani et al., 2014),

and facility location (Guastaroba and Speranza, 2014). With a similar motive, we propose fixing binary variables based on LP relaxation of the problem in an iterative way to maintain feasibility. Preliminary results showed that the LP-based approach has been able to further reduce solution time without decreasing the solution quality. The NEDA model is trained on deterministic instances based on scenario sampling and not on the exact solutions of stochastic programs. We believe that including the LP relaxation solution of the original problem helps ScenPredOpt capture discoveries that were not identified in the model due to training on deterministic instances and averaging hidden states. For various problems, different heuristics can be integrated instead of the LP relaxation-based heuristic to suit the problem needs better.

Algorithm 4 presents the details of the ScenPredOpt algorithm. For testing, the ScenPredOpt algorithm takes a trained NEDA model M, input data of the test set α, initial level (percent) for model-predicted binary variables θ_M, initial level (percent) for LP relaxation-assigned binary variables θ_{LP}, reduction in the level (percent) for model-predicted binary variables λ_M, and reduction in the level (percent) for LP relaxation-assigned binary variables λ_{LP}. The output of the framework is the best objective function value determined by the ScenPredOpt framework. Also, four operators are defined. First, the operator F^{MIP} is an MIP solver, and the inputs to F^{MIP} are problem parameters α, binary variables predicted by the model and used at a certain level β_M^f, binary variables assigned by LP relaxation and used at a certain level β_{LP}^f, and the set of constraints C. MIP solver F^{MIP} returns either z as the best solution found or $\varnothing$ that denotes an infeasible problem. Second, the operator $F^{MIP-FEAS}$ is a feasibility-checker for an MIP and takes a similar set of arguments as MIP solver F^{MIP}. Here, it is assumed when the sets β_M^f or β_{LP}^f are $\varnothing$, and the problem is solved without fixing those variables. Also, instead of using the set of all constraints C, the feasibility check is performed using a set of predicted tight constraints $\widetilde{C}$. The feasibility-checker operator $F^{MIP-FEAS}$ terminates when a

first feasible solution is found, or the problem is certified to be infeasible, which is denoted by $\varnothing$. Third, the operator F^{LP} is an LP solver and takes the same arguments as F^{MIP} except LP-assigned binary variables β^f_{LP} since they are not yet determined. Differently from MIP solver F^{MIP}, LP solver F^{LP} removes binary variable restrictions from formulation and solves the problem as an LP. Finally, the operator P is the top prediction generator and takes a pair of inputs as a set of variables: all of the predicted binary variables by model β_M and prediction level θ_M, or LP relaxation assigned binary variables β_{LP} and prediction level θ_{LP}. Here, the predicted variables are selected up to the desired percent of all variables by a function of $max(\beta_M, 1-\beta_M)$, where $\beta_M \in [0,1]$ is a predicted value for model-predicted variables. Similarly, LP-assigned variables are determined by using $max(\beta_{LP}, 1-\beta_{LP})$, where $\beta_{LP} \in [0,1]$. The calculated values are ordered from smallest to largest and selected up to the input prediction level or prediction percent of the total variables. This approach determines a set of variables closest to binary values (0 or 1); thus, the model is more confident in its prediction. The selected variables are labeled as 0 or 1 based on their closeness. The output of this operator is a set of fixed variables: β^f_M or β^f_{LP}, depending on the input.

The algorithm starts with predicting the binary variables β_M and tight constraints $\widetilde{C}$ in steps 1 and 2, respectively. In step 3, the set of variables predicted and fixed by the model, β^f_M, is determined with an initial level θ_M. Next, the prediction level feasibility loop is introduced in step 4, and it is continued until a feasible solution is found using the MIP feasibility checker $F^{MIP-FEAS}$. Note that in this step, we use model-predicted variables and a set of tight constraints $\widetilde{C}$, but do not include any LP-assigned binary variables β^f_{LP}. Within this loop, if a feasible solution is not found, prediction level θ_M is reduced by λ_M, and a smaller set of β^f_M is generated. Once this loop is ended, a linear relaxation of the problem is solved using LP solver F^{LP} in step 8 using determined predictions β^f_M and all constraints

Algorithm 4 ScenPredOpt Algorithm

Input: Trained NEDA model M
Input data of the test set α
Initial level for model-predicted binary variables θ_M
Initial level for LP relaxation-assigned binary variables θ_{LP}
Reduction in the level for model-predicted binary variables λ_M,
Reduction in the level for LP relaxation-assigned binary variables λ_{LP}

Output: Optimal objective function value z

Operator: MIP solver $F^{MIP}(\alpha, \beta_M^f, \beta_{LP}^f, C)$
MIP feasibility checker $F^{MIP-FEAS}(\alpha, \beta_M^f, \beta_{LP}^f, \widetilde{C})$
LP solver $F^{LP}(\alpha, \beta_M^f, \beta_{LP}^f, C)$
Select top predictions $P(\beta_M, \theta_M)$ or $P(\beta_{LP}, \theta_{LP})$

ScenPredOpt

1: Predict binary decision variables: $\beta_M = M(\alpha)$
2: Predict the set of tight constraints: $\widetilde{C} = M(\alpha)$
3: Initialize model-predicted binary variables: $\beta_M^f = P(\beta_M, \theta_M)$
Loop to determine β_M^f with tight constraints
4: **while** $F^{MIP-FEAS}(\alpha, \beta_M^f, \varnothing, \widetilde{C}) = \varnothing$ **do**
5: Reduce the used level for model-predicted binary variables: $\theta_M = \theta_M - \lambda_M$
6: Get top predictions to fix: $\beta_M^f = P(\beta_M, \theta_M)$
7: **end while**
Determine LP-relaxation-assigned binary variables
8: Solve LP relaxation: $\beta_{LP} \Leftarrow F^{LP}(\alpha, \beta_M^f, \varnothing, C)$
9: Initialize LP relaxation-assigned binary variables: $\beta_{LP}^f = P(\beta_{LP}, \theta_{LP})$
Loop to determine β_{LP}^f with tight constraints
10: **while** $F^{MIP-FEAS}(\alpha, \beta_M^f, \beta_{LP}^f, \widetilde{C}) = \varnothing$ **do**
11: Reduce used level for LP-assigned binary variables: $\theta_{LP} = \theta_{LP} - \lambda_{LP}$
12: Get top predictions to fix: $\beta_{LP}^f = P(\beta_{LP}, \theta_{LP})$
13: **end while**
Loop to solve the model with fixed variables β_M^f and β_{LP}^f with all constraints
14: **while** $F^{MIP}(\alpha, \beta_M^f, \beta_{LP}^f, C) = \varnothing$ **do**
15: Reduce used level for model-predicted binary variables: $\theta_M = \theta_M - \lambda_M$
16: Reduce used level for LP-assigned binary variables: $\theta_{LP} = \theta_{LP} - \lambda_{LP}$
17: Get top predictions to fix: $\beta_M^f = P(\beta_M, \theta_M)$
18: Get top predictions to fix: $\beta_{LP}^f = P(\beta_{LP}, \theta_{LP})$
19: **end while**
20: Return solution found in step 14 $z = F^{MIP}(\alpha, \beta_M^f, \beta_{LP}^f, C,)$

C instead of predicted tight constraints $\widetilde{C}$. The resulting variables constitute the set of LP-assigned binary variables β_{LP}. Then in step 9, the set of fixed variables is determined with an initial level for LP-assigned binary variables θ_{LP}. Here, β^f_{LP} excludes the set of model-predicted variables β^f_M, i.e., $\beta^f_{LP} \cap \beta^f_M = \varnothing$ always holds. In the following step 10, a new loop begins and continues until a feasible solution is found. This loop is necessary to overcome possible infeasibility resulting from fixing binary variables from the LP relaxation. Similar to the loop in step 4, this loop solves the problem using model-predicted variables β^f_M and a set of tight constraints $\widetilde{C}$, but also includes fixing LP-assigned binary variables β^f_{LP} during the solution. Within the loop, the level for LP-assigned binary variables θ_{LP} is reduced by λ_{LP}, and the set of LP-assigned fixed binary variables β^f_{LP} is updated. Once a feasible solution is found, a new loop begins in step 14 to achieve a solution to report by the ScenPredOpt. However, this loop utilizes the complete formulation of the problem considering all constraints C and not just the predicted set of tight constraints $\widetilde{C}$ and either solves to problem fully or reports infeasibility. This is to ensure feasibility is maintained with the original problem. However, in practice, the previous loops in steps 4 and 10 are well-performing, and the algorithm usually terminates without using steps 14 to 18. The loop defined by steps 14 to 18 is only used when it is necessary to ensure feasibility and is not run often by ScenPredOpt since the former loops find a feasible solution with high success. If a feasible solution is not found, both fixed levels for model-predicted binary variables θ_M and LP relaxation-assigned binary variables θ_{LP} are reduced by λ_M and λ_{LP}, respectively. Then, in the next iteration, both variables predicted by model β^f_M and variables assigned by LP relaxation β^f_{LP} are updated. Then, in step 20, the best solution is determined by the ScenPredOpt, and the corresponding objective function value is returned from the solution found in the loop starting at step 14.

5.4.3 Generalization with Item-wise Expansion

Here, we present an item-wise generalization algorithm that enables a NEDA model trained with a few items to predict problems with a large set of items. This presents an important aspect of the ScenPredOpt solution framework since then models can be trained using small-sized problems that are easier to solve. Therefore, a significant time reduction can be achieved when generating training instances and model training time. In Chapter 3, we introduced an item-wise generalization algorithm and show that the model is very successful at generalizing in the item dimension. Here, we improve the item-wise generalization algorithm by considering the variability in the predictions made.

Similar to Algorithm 1 in Chapter 3, we make predictions at each step of the algorithm using a subset of items and consolidate them to use in the ScenPredOpt framework. Algorithm 5 presents the details of the improved item-wise generalization algorithm. The inputs to the algorithm are trained model M, the number of items during model training I^M, input data of the test set α, set of items in the test set $i \in \{1, \ldots, I\}$, and threshold prediction count δ. The output is a set of predictions $\hat{\beta}_i$ $\forall i \in \{1, \ldots, I\}$. In step 1, we initialize prediction count γ_i to be zero, for each item i. Then, cumulative prediction β_i is initialized as 0, for each item i. In step 3, we initialize the set of saved predictions π_i for each item $i \in \{1, \ldots, I\}$, which is a set to save predictions made at each iteration. Next, an iteration is started until each item i is predicted at least threshold prediction count δ times. In step 5, we sample a subset of items B with a size equal to the number of items that the model is trained with, i.e., $|B| = I^M$. Later, we generate predictions $\hat{\theta}_B$ using the trained model M. The input to the model is the data of the selected subset α_B, in which the right-hand coefficients are scaled back to account for an increased number of items. For SMCLSP, we adjust Equation (5.2c) as $c_t^s = c_t^s \times \frac{\sum_{i \in B} d_{it}^s}{\sum_{i=1}^{I} d_{it}^s}$, $\forall t = 1, \ldots, T$, $\forall s = 1, \ldots, S$,. For SMSMK, we adjust Equation (5.3b) as $c_{jt}^s = c_{jt}^s \times \frac{\sum_{i \in B} w_{ijt}^s}{\sum_{i=1}^{I} w_{ijt}^s}$, $\forall t = 1, \ldots, T$, $\forall j = 1, \ldots, J$,

$\forall s = 1, \ldots, S$. In step 7, we sum the current predictions $\hat{\theta}_i$ with $\hat{\beta}_i$ for the selected subset B. In step 8, we save the current predictions $\hat{\theta}_i$ by concatenating them with the existing predictions π_i. Then, the prediction count γ_i is increased for each item $i \in B$. In step 11, we calculate standard deviation σ_i for all items $i \in \{1, \ldots, I\}$ using the set of saved predictions π_i. Then, in step 12, we divide the cumulative predictions $\hat{\beta}_i$ with their respective counts γ_i. Then, for each item $i \in \{1, \ldots, I\}$, we calibrate the predictions with calculated standard deviations. The main idea is to push variables with higher variability closer to 0.5 so that they are determined by the ScenPredOpt instead of directly fixing it. Here, we intend to hold variables with lower variability in high regard. For example, if two variables have similar predictions, we prefer the variables that are consistently predicted instead of the more uncertain and unstably predicted ones. Therefore, we add or subtract the variability from predictions based on their closeness to 0 or 1. Then, those variables with higher deviations would be left to the solver. Also, we add or subtract squared standard deviation instead of directly the standard deviation since we found out empirically that using the latter is a severe punishment for variability and the former performs better. Moreover, the squared standard deviation σ_i^2 is smaller than the standard deviation σ_i since all predictions are between 0 and 1. In step 15, if the resulting prediction $\hat{\beta}_i$ is smaller than 0.5, we add the squared standard deviation σ_i^2. Then, we ensure that the prediction is not more than 0.5, otherwise, it might be predicted as 1 within the ScenPredOpt framework. Else, the squared standard deviation σ_i^2 is subtracted from the prediction $\hat{\beta}_i$ to push more uncertain predictions to the middle ground to be determined by the commercial solver. Similarly, we ensure that the final prediction value is not less than 0.5.

Algorithm 5 Improved Item-wise Generalization Prediction Algorithm

Input: Trained model M, number of items during model training I^M, input data of the test set α, set of items in the test set $i \in \{1, \ldots, I\}$, and threshold prediction count δ

Output: Predicted value for item $\forall i \in \{1, \ldots, I\}$, $\hat{\beta}_i$, for the test set

Improved Item-wise Generalization

Initialize prediction count of each item to be zero: $\gamma_i = 0, \forall i \in \{1, \ldots, I\}$
Initialize prediction of each item to be zero: $\beta_i = 0, \forall i \in \{1, \ldots, I\}$
Initialize set of saved prediction for each item as empty: $\pi_i, \forall i \in \{1, \ldots, I\}$
while $\exists\ \gamma_i \leq \delta\ i \in \{1, \ldots, I\}$ **do**
    Sample a subset of items, B, with the number of items in the subset equal to the number of items in the trained model I^M: $B \subset \{1, \ldots, I\}$ and $|B| = I^M$
    Make a forward pass using the input data of the selected subset B of items: $\hat{\theta}_B = M(\alpha_B)$
    Sum predictions for subset B of selected items: $\hat{\beta}_i := \hat{\beta}_i + \hat{\theta}_i, \forall i \in B$
    Save predictions for subset B of selected items: $\pi_i := [\pi_i, \hat{\theta}_i], \forall i \in B$
    Increase prediction counts for subset B of selected items: $\gamma_i = \gamma_i + 1, \forall i \in B$
end while
Calculate standard deviation of predictions for each item i using saved predictions: $\sigma_i = StdDev(\pi_i), \forall i \in \{1, \ldots, I\}$
Calculate the prediction value for each item $i \in \{1, \ldots, I\}$ by dividing the sum of saved predictions with respective counts: $\hat{\beta}_i := \hat{\beta}_i \div \gamma_i$
for For each item $i \in \{1, \ldots, I\}$ **do**
    if $\hat{\beta}_i \leq 0.5$ **then**
        Calibrate prediction for item i by summing the squared standard deviation: $\hat{\beta}_i := \min\{0.5, \hat{\beta}_i + \sigma_i^2\}$
    else
        Calibrate prediction for item i by subtracting the squared standard deviation: $\hat{\beta}_i := \max\{0.5, \hat{\beta}_i - \sigma_i^2\}$
    end if
end for

5.5 Implementation and Experimentation Details

In this section, the details of generating instances, experimentation setup, and evaluation metrics are presented. A high-performance computing cluster is used for instance generation, training, and testing. The cluster runs on Linux 3.10.0 with Intel Xeon Gold 6226R 2.90 GHz, 96 GB of memory, and an NVIDIA Tesla T4 GPU. Test instances are solved using Gurobi Optimizer 9.5 instead of CPLEX 20.1.0 since it was faster on preliminary stochastic multi-stage test problems. Python 3.8.5 and PyTorch 1.7.1 are used for training and testing. Gurobi is used with Python API. All reported solution times are in CPU seconds. All models are trained using GPUs.

5.5.1 Instance Generation

The parameters of instances are sampled from nonnegative uniform integer distribution between integers a and b represented by $U[a, b]$. For testing instances, we also employ a scenario capping strategy to reduce the exponentially growing number of scenarios with increasing time periods. In this setting, after the capping period, no new scenario is generated, and the data of the remaining stages are assumed to be the same or deterministic. Before the capping period, for each scenario node, the set of succeeding scenarios is generated to be the same. For example, in the fourth stage of Figure 5.1a, the parameters for scenarios 1,3,5, and 7 are identical, and the parameters for scenarios 2,4,6, and 8 are identical since we set the capping period to three in this example. In Appendix B, we present further details of the generated test instances.

SMCLSP Instances: Instance generation for SMCLSP is implemented based on the approach given in Büyüktahtakın et al. (2018b). The hardness of the SMCLSP is controlled by two main factors: capacity-to-demand ratios $c \in \{10, 14\}$ and setup-to-holding cost ratio $r = 1,000$. The production cost p_{it}^s is drawn from $U[1, 200]$, inventory cost h_{it}^s is sampled from $U[1, 100]$, and demand d_{it}^s is drawn from

$U[500, 1500]$. The overall means of d and h are represented by $\bar{d}$ and $\bar{h}$, respectively. The capacity c_t is drawn from $U\left[0.8c\bar{d}, 1.2c\bar{d}\right]$ and setup cost f_{it}^s is drawn from $U\left[0.9r\bar{h}, 1.1r\bar{h}\right]$. Two different models are trained for SMCLSP: The first model is trained with $T = 40$ periods and $I = 8$ items. The second model is trained with $T = 30$ periods and $I = 12$ items. For both models, datasets with some combinations of $T \in \{10, 15\}$ and $S \in \{32, 64, 81, 125, 243, 512\}$ are used for testing. For item-wise expansion, we used $T \in \{15, 10\}$, $S \in \{32, 64, 81\}$, and $I \in \{24, 32, 16\}$ for the first model with $I = 8$ and $T \in \{15, 10\}$, $S \in \{32, 64, 81\}$, and $I \in \{36, 48, 24, \}$ for the second model with $I = 12$. For SMCLSP, 18 test sets are generated, each including 20 test instances.

SMSMK Instances: The profit p_{it}^s is drawn from $U[1, 1000]$, the stability bonus b_{it}^s is drawn from $U[1, 1000]$, item weights w_{ijt} is drawn from $U[1, 1000]$, and the capacity c_{jt}^s is drawn from $U\left[0.5\sum_{i=1}^{I} w_{ijt}, 0.8\sum_{i=1}^{I} w_{ijt}\right]$. Two models are trained for SMSMK: The one with $T = 30$ periods with $I = 8$ items and the second with $T = 30$ periods with $I = 10$ items. Test set for both models have some combination of $T \in \{10, 15\}$ and $S \in \{32, 64, 81, 125, 243, 512\}$. Also, both models are tested with the given parameters for the two-stage problem. For the first model trained with $I = 8$, instances with $T \in \{15, 10\}$, $S \in \{32, 64, 81\}$, and $I \in \{24, 32, 16\}$ are used for testing item-wise expansion. For the second model trained with $I = 10$, test set with $T \in \{15, 10\}$, $S \in \{32, 64, 81\}$, and $I \in \{30, 40, 20\}$ are generated with item-wise expansion. A total of 30 test sets, each with 20 instances, have been generated.

5.5.2 Implementation Specifications

For SMCLSP, we implement SDDiP based on Ding et al. (2019) with the default setting and a time limit of 7200 seconds until 20 stable iterations are achieved. Additionally, we implement a heuristic based on Absi and van den Heuvel (2019) to generate a baseline for comparison. Within the heuristic, the number of fixed

periods is assigned as $\frac{T}{20}$, and the periods with binary variables are assigned as $\frac{T}{10}$. For SMSMK, we utilize the PH approach presented by Watson and Woodruff (2011) to compare our solution quality. Their PH algorithm is an enhanced version of the classical PH algorithms that address the convergence issues. SDDiP is not applicable for SMSMK since SDDiP requires the complete or relatively complete recourse condition and the SMSMK is not completely recourse. Also, we utilize the heuristic solution presented in Bertsimas and Demir (2002) for benchmarking as there is not an existing heuristic approach specifically developed for SMSMK given by Equation (5.3). Their heuristic solves an LP relaxation of the problem at each iteration. Due to the computational challenge of this heuristic, we set 0.1% of binary variables to 0 instead of a single variable at each step of the iteration. This modification results in a much faster heuristic solution at the cost of a slightly increased optimality gap for instances with a very large number of variables. Our objective with this modification is to present a fairer time comparison between ScenPredOpt and the heuristic of Bertsimas and Demir (2002).

We set the initial level for model-predicted binary variables $\theta_M = 50\%$ for SMCLSP and $\theta_M = 40\%$ for SMSMK. For both problem types, we assign the initial level for LP relaxation-assigned binary variables $\theta_{LP} = 5\%$. Also, the reduction in the level for model-predicted binary variables $\lambda_M = 10\%$ and the reduction in the level for LP relaxation-assigned binary variables $\lambda_{LP} = 1\%$. The time to generate predictions is less than one second for all instances and is included in $timeScenPredOpt$. For all tables, we present the number of stages T and the total number of scenarios $\#sc$.

5.5.3 Model Training

The NEDA models are trained with longer period problems than the instances in the test set to capture and learn more sequential decision structures with the attention structure. The instances used in the training set are deterministic and solved in 1.7

seconds on average. The SMCLSP models in Tables 5.1 and 5.2 has 128 and 64 hidden units in the decoder for both forward and backward layer, respectively. The former SMCLSP model is trained for 18 hours using a training set with $T = 40$. The latter SMCLSP model is trained using $T = 30$ problems in 30 hours. The SMSMK models with results presented in Tables 5.4 and 5.5 are trained using instances with $T = 30$ and contain 256 and 128 hidden units in both forward and backward layers, respectively. The encoders contain 2 bidirectional LSTM layers while the decoder contains a unidirectional 2-layer LSTM. We have utilized a standard learning schema by applying training, validation, and test sets that contain 3,500,000, 10,000, and 20 instances, respectively (Alpaydin, 2020). Furthermore, a dropout schema (Srivastava et al., 2014) is used to regularize the learning with a random rate of $\{0.25, 0.30, 0.35\}$, which is known to limit overfitting. We make use of the popular Adam optimizer with an initial learning rate of 0.01 for SMCLSP and 0.001 for SMSMK (Kingma and Ba, 2014).

5.5.4 Evaluation Methodology

We have utilized the following metrics to measure the success of ScenPredOpt solution time and its quality compared to exact and heuristic approaches. To a large extent, we employ the metrics given in Chapter 3:

- **timeGRB**: Average solution time in CPU seconds for SMCLSP or SMSMK with Gurobi 9.5 at its default settings.
- **timeScenPredOpt**: Average solution time in CPU seconds for SMCLSP or SMSMK with ScenPredOpt framework.
- **timeHeur**: Average solution time in CPU seconds for SMCLSP or SMSMK with the heuristic of Absi and van den Heuvel (2019) for SMCLSP and the heuristic of Bertsimas and Demir (2002) for SMSMK.
- **timeSDDiP**: Average solution time in CPU seconds for SMCLSP with SDDiP.
- **timePH**: Average solution time in CPU seconds for SMSMK with PH algorithm.

Additionally, we define the following metrics:

Definition 5.5.1 *The optimality gap between the ScenPredOpt solution $\hat{x}^*$ (heuristic solution for **optGapHeur**, SDDiP solution for **optGapSDDiP**, and PH solution for **optGapPH**) and Gurobi solution x^*. Let $Z(\bullet)$ be the corresponding operator to calculate the function value given any solution. Then the optimality gap is:*

$$\textbf{optGapScenPredOpt}(\%) = \frac{|Z(\hat{x}^*) - Z(x^*)|}{Z(x^*)} \times 100. \tag{5.10}$$

Definition 5.5.2 *The solution time improvement factor accomplished by employing the ScenPredOpt framework (heuristic time improvement for **timeImpHeur**, SDDiP time improvement for **timeImpSDDiP**, and PH time improvement for **timeImpPH**) compared to the Gurobi solution time is:*

$$\textbf{timeImpScenPredOpt} = \frac{timeGRB}{timeScenPredOpt}. \tag{5.11}$$

Definition 5.5.3 *One-sided Wilcoxon signed-rank test is used to measure if both samples are from the same population. It is a non-parametric alternative to the t-test and appropriate for statistically comparing solution times of ML-based OR approaches (Accorsi et al., 2022). We conclude that ScenPredOpt is statistically faster than the heuristic if the **p-value** is smaller than 0.01. The hypotheses are:*

$$H_0 : median(timeHeur - timeScenPredOpt) < 0 \tag{5.12a}$$

$$H_1 : median(timeHeur - timeScenPredOpt) > 0 \tag{5.12b}$$

5.6 Results

This section provides a discussion of the computational experiments. We compare the ScenPredOpt framework with mathematical solver, exact, and heuristic approaches. The test set for each specified characteristic of an instance includes 20 test instances, and all the results presented in the tables are average values for those 20 instances. All instances are solved using Gurobi 9.5 with a 2-hour time limit (Gurobi Optimization, LLC, 2022).

5.6.1 Quality of Predictions for SMCLSP

Table 5.1 presents the detailed computational results for the first set of SMCLSP instances. The 8-item model is trained using scenario-sampled deterministic instances with $T = 40$ periods and 8 items. The local attention structure helps the model to generalize instances with a varying number of periods. In the first set of Table 5.1, the test has 32 scenarios and $T = 10$ stages. Gurobi achieves an average solution time of more than 3,000 seconds, while SDDiP reduces the solution time to nearly 450 seconds with a gap of 0.03% to the Gurobi value. The heuristic significantly reduced the solution time to almost 8 seconds, with a relatively high optimality gap of 2.13%. The ScenPredOpt remedies the trade-off between solution time and quality and achieves a 3.6-second solution with only a 0.16% optimality gap. Here, solution time is reduced by a factor of 600, when compared to Gurobi. For the third set of instances with 81 scenarios, a similar solution time improvement is achieved by ScenPredOpt with a gap of only 0.08%. The Gurobi solution time of the instances is not necessarily proportional to the number of scenarios or stages; rather, it is most likely related to the structure of scenario trees. However, ScenPredOpt can solve the 8-item SMCLSP for all cases much faster than Gurobi, SDDiP, and heuristic. Also, compared to the heuristic, the optimality gaps are much smaller. The p-values for the Wilcoxon signed-rank test are all below 0.01, ensuring that ScenPredOpt is faster than the heuristic. The time improvement factor values for ScenPredOpt get lower as problems get harder because the solution time is set to a 2-hour time limit. Gurobi would require a much longer solution time than two hours for harder instances, as proven by the increasing optimality gap. Since the ScenPredOpt has no solution time set, the solution with ScenPredOpt starts taking a long time, and the time improvement values appear to be shrinking. The time improvement values would be much higher if the benchmark Gurobi instances had not terminated early on.

Table 5.1 Average Results of Experiments for SMCLSP with 8 Items

T	10	10	10	15	15	15	Avg	Mdn	Std
#sc	32	64	81	125	243	512	176	103	165
timeGRB	3,042	4,582	5,773	6,508	7,203	6,850	5,660	7,200	2,919
timeScenPredOpt	4	13	16	53	77	95	43	18	60
timeSDDiP	454	738	924	4,342	2,151	1,715	1,721	1,203	1,553
timeHeur	8	89	80	341	749	743	335	103	385
timeImpScenPredOpt	600	376	599	264	145	105	348	141	498
timeImpSDDiP	6	8	10	2	4	4	6	4	6
timeImpHeur	319	148	150	34	13	10	112	14	207
optGapGRB(%)	0.02	0.07	0.07	0.13	0.18	0.11	0.10	0.07	0.09
optGapScenPredOpt(%)	0.16	0.07	0.08	0.09	0.51	0.36	0.21	0.09	0.35
optGapSDDiP(%)	0.03	0.03	0.03	0.02	0.02	0.03	0.03	0.02	0.01
optGapHeur(%)	2.13	1.88	1.78	1.14	1.18	1.14	1.54	1.45	0.61

Table 5.2 shows the computational results with 12-item SMCLSP. Here, the model is trained using $T = 30$-period instances with 12 items. The results show a similar trend as in Table 5.1, favoring ScenPredOpt over the heuristic. Gurobi solves the first three sets of instances in Table 5.2 on an average of more than 6,000 seconds. ScenPredOpt solves all three of those instances in less than a minute achieving a small optimality gap of less than 0.41%. The SDDiP halves solution time with a small optimality gap, but still, it does not reach a fast solution like ScenPredOpt. The last three sets of instances are significantly challenging since none of the individual instances can be solved to optimality in the set 2-hour time limit using Gurobi, hence the average Gurobi solution time of approximately 7,200 seconds. For such instances, heuristic time is close to or more than 1,000 seconds, whereas the time of ScenPredOpt is closer to 100 seconds with a significantly lower optimality gap compared to the heuristic.

All p-values of the Wilcoxon test are smaller than 0.01, suggesting that ScenPredOpt is faster with a much smaller optimality gap than the heuristic.

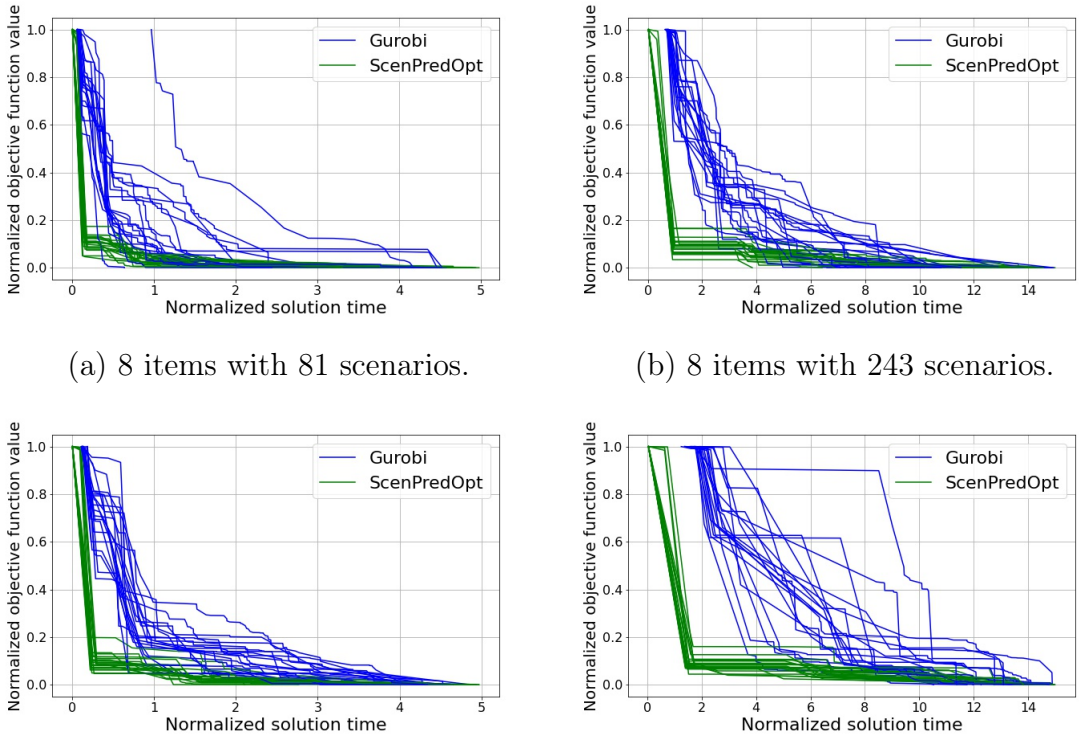
Therefore, ScenPredOpt achieves a notable computational advance by generating fast, high-quality solutions to challenging scenario-based multi-stage stochastic programs. Also, for both Tables 5.1 and 5.2, the overall average and median solution time with the ScenPredOpt is smaller than the heuristic solution time with a lower standard deviation. Therefore, we can highlight that ScenPredOpt outperforms the heuristic in the solution.

Table 5.2 Average Results of Experiments for SMCLSP with 12 Items

T	10	10	10	15	15	15	Avg	Mdn	Std
#sc	32	64	81	125	243	512	176	103	165
timeGRB	6,187	6,185	6,158	7,200	7,202	7,201	6,689	7,200	1,815
timeScenPredOpt	43	17	24	58	150	95	64	41	80
timeSDDiP	3,548	3,681	3,790	7,131	7,145	7,197	5,415	7,082	2,192
timeHeur	212	242	289	967	996	1,355	677	610	566
timeImpScenPredOpt	322	623	462	165	89	115	296	159	348
timeImpSDDiP	2	2	2	1	1	1	2	1	1
timeImpHeur	101	90	83	10	8	6	49	10	77
optGapGRB(%)	0.07	0.07	0.10	0.17	0.15	0.10	0.11	0.11	0.07
optGapScenPredOpt(%)	0.26	0.30	0.41	0.17	0.25	0.87	0.38	0.31	0.33
optGapSDDiP(%)	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.01
optGapHeur(%)	2.12	1.92	1.91	1.17	1.31	1.18	1.60	1.56	0.54

Comparison of ScenPredOpt and Gurobi in the first few seconds Figure 5.2 presents the progress of normalized objective function values during the first few seconds of the solution for ScenPredOpt and Gurobi. Specifically, Figure 5.2a-5.2b show the solution progress for the third and fifth set of instances in Table 5.1, with 81 and 243 scenarios, respectively. Figure 5.2c-5.2d show the solution progress for the third and fifth set of instances in Table 5.2, having 81 and 243 scenarios, respectively. All four figures highlight that the ScenPredOpt reduces the objective function value

at a faster rate compared to Gurobi. Also, the progress of objective value stabilizes earlier, underlining the success of the ScenPredOpt.



(a) 8 items with 81 scenarios.

(b) 8 items with 243 scenarios.

(c) 12 items with 81 scenarios.

(d) 12 items with 243 scenarios.

Figure 5.2 Progress of Gurobi and ScenPredOpt objective values during the first few seconds of the solution process. All solution times are given in CPU seconds.

Instance by Instance Comparison In Table 5.3, we present the instance-by-instance comparison for 10 SMCLSP instances with larger scenarios than the previously-presented ones. The problem contains 8 items, 13 periods, and 2 scenarios per stage without scenario capping. Therefore, each instance contains 4,096 scenarios. The results are generated using the 8-item model in Table 5.1. The Gurobi does not reach to an optimal solution for any of the problems within the given 2-hour solution time limit. Also, SDDiP cannot reach a better solution than Gurobi in a given 20-stable iteration limit. The solution time of ScenPredOpt changes between 80 and 882 seconds. The heuristic has a higher solution time than the ScenPredOpt for all 10

instances within the range of 1,222 to 2,086 seconds. This translates to a higher time improvement factor of the ScenPredOpt compared to the heuristic for all instances. Also, optimality gaps between the ScenPredOpt and heuristic differ considerably in favor of the ScenPredOpt. Table 5.3 demonstrates that the ScenPredOpt is preeminent not only in averages but also superior in instance-by-instance cases compared to the heuristic's solution time and optimality gap and exact approaches' solution time. Furthermore, Table 5.3 presents a comparison between ScenPredOpt and the PredOpt introduced in Chapter 3. The solution time of PredOpt is denoted by timePredOpt, the time improvement factor achieved by PredOpt is denoted by timeImpPredOpt, and the resulting optimality gap is denoted by optGapPredOpt. In 8 over 10 instances, ScenPredOpt outperforms PredOpt in terms of both solution time and optimality gap. In instance 2, ScenPredOpt has a better optimality gap than PredOpt, but with a slightly higher solution time. The third instance presents the opposite case where ScenPredOpt has a better solution time with a larger optimality gap compared to PredOpt. The averages highlight the superiority of ScenPredOpt over PredOpt, with the former having a lower average solution time, higher time improvement, and lower optimality gap.

To summarize the results presented in Tables 5.1, 5.2, and 5.3 and Figure 5.2, ScenPredOpt achieves a better solution than the heuristic in a statistically faster process. The optimality gap of state-of-the-art SDDiP is lower than ScenPredOpt at the cost of a significantly increased solution time. Therefore, ScenPredOpt can be a favorable alternative to Gurobi and SDDiP by providing a fast and high-quality solution at a fraction of the computational cost without requiring expert knowledge.

5.6.2 Quality of Predictions for SMSMK

Table 5.4 presents the first set of results for the SMSMK. The model is trained with $T = 30$-period instances. For the first set with 32 scenarios, solution time is reduced

Table 5.3 Detailed Results of Experiments for SMCLSP with 8 Items

#instance	1	2	3	4	5	6	7	8	9	10	Avg
timeGRB	7,201	7,201	7,207	7,201	7,208	7,201	7,201	7,201	7,201	7,201	7,202
timeScenPredOpt	80	89	374	166	882	85	90	100	100	92	206
timePredOpt	92	74	505	237	1,405	157	136	101	143	168	302
timeSDDiP	1,297	856	3,317	1,260	2,261	1,454	845	1,368	896	1,121	1,468
timeHeur	2,086	1,222	1,591	2,074	1,581	1,637	2,033	1,917	1,751	1,493	1,738
timeImpScenPredOpt	90	81	19	43	8	85	80	72	72	78	63
timeImpPredOpt	78	97	14	30	5	46	53	71	50	43	49
timeImpSDDiP	6	8	2	6	3	5	9	5	8	6	6
timeImpHeur	3	6	5	3	5	4	4	4	4	5	4
optGapGRB(%)	0.15	0.02	0.05	0.08	0.05	0.10	0.14	0.11	0.04	0.10	0.09
optGapScenPredOpt(%)	0.13	0.58	0.07	0.66	0.22	0.29	0.05	0.43	0.01	0.74	0.32
optGapPredOpt(%)	0.16	0.65	0.04	0.76	0.27	0.44	0.07	0.60	0.07	0.92	0.40
optGapSDDiP(%)	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.03	0.03	0.03
optGapHeur(%)	1.87	1.16	1.80	1.17	0.97	1.24	0.90	1.93	1.86	1.94	1.48

from more than 1,000 seconds with Gurobi to only one second with ScenPredOpt. This is much faster compared to the heuristic with a better optimality gap, i.e., a 2.73% gap with the heuristic compared to 1.09% with ScenPredOpt. The PH improves the solution time of Gurobi for most cases with a slight optimality gap, but it is much slower compared to the heuristic or ScenPredOpt. For example, PH more than halves the solution time with only a 0.02% optimality gap in almost 2,000 seconds in the fourth test set of Table 5.4. The ScenPredOpt can provide a solution in less than 5 seconds with only a 0.82% optimality gap. While the heuristic outperforms PH in solution time, ScenPredOpt outperforms the heuristic in terms of both solution time and optimality gap. Also, all p-values of the Wilcoxon signed-rank are smaller than 0.01 highlighting the success of the ScenPredOpt over the heuristic.

Table 5.5 presents another set of instances for SMSMK with 10 items. The results highlight the solution quality of the ScenPredOpt over the heuristic and time improvement over the Gurobi and PH, similar to Table 5.4. For example, in the last dataset with 512 scenarios, solution time is reduced to almost a minute from two

Table 5.4 Average Results of Experiments for SMSMK with 8 Items

T	10	10	10	15	15	15	Avg	Mdn	Std
#sc	32	64	81	125	243	512	176	103	165
timeGRB	1,155	2,149	2,144	4,501	7,200	6,949	4,016	6,954	3,348
timeScenPredOpt	1	2	2	5	12	17	6	2	8
timePH	77	708	1,051	1,936	7,355	6,760	2,981	657	3,305
timeHeur	4	9	11	29	179	170	67	17	86
timeImpScenPredOpt	1,013	1,585	1,617	1,206	713	533	1,111	525	1,775
timeImpPH	17	10	3	5	1	1	6	1	19
timeImpHeur	245	233	185	154	46	44	151	48	286
optGapGRB(%)	0.03	0.03	0.07	0.22	1.21	1.01	0.43	0.01	0.65
optGapScenPredOpt(%)	1.09	1.06	1.28	0.82	1.31	1.30	1.14	1.10	0.61
optGapPH(%)	0.09	0.02	0.05	0.02	0.18	0.17	0.09	0.02	0.16
optGapHeur(%)	2.73	3.61	3.11	2.82	3.44	2.71	3.07	3.07	1.15

hours. This significant time improvement factor of 150 is achieved with an optimality gap of only 0.77%. Similar to the formerly presented tables with SMCLSP and SMSMK, all p-values for the Wilcoxon test are smaller than 0.01. Also, the optimality gaps of the heuristic are much larger than the ScenPredOpt for all instances.

To sum up Tables 5.4 and 5.5, ScenPredOpt achieves a significant reduction in solution time compared to Gurobi and PH at the cost of a small optimality gap. Also, it outperforms the heuristic in terms of solution quality and time for all cases. Thus, ScenPredOpt is a promising alternative to other exact and heuristic algorithms in generating fast but high-quality solutions to complex problems that need to be repeatedly and quickly solved.

Quality of Predictions for 2SMK In this section, we present the results for a two-stage multi-dimensional knapsack problem (2SMK). The traditional solution methodologies for two-stage problems can be notably separated from the multi-stage problem. While the latter is usually considered more complex with growing scenario

Table 5.5 Average Results of Experiments for SMSMK with 10 Items

T	10	10	10	15	15	15	Avg	Mdn	Std
#sc	32	64	81	125	243	512	176	103	165
timeGRB	2,351	5,344	5,130	6,400	7,200	7,200	5,604	7,200	2,868
timeScenPredOpt	1	2	4	14	34	69	21	6	34
timePH	1,920	2,489	1,958	2,989	5,670	7,406	3,739	3,037	3,180
timeHeur	3	9	11	29	66	168	48	18	61
timeImpScenPredOpt	2,374	4,502	2,047	660	329	150	1,677	330	2,840
timeImpPH	18	13	8	4	1	1	7	1	19
timeImpHeur	633	597	449	228	115	45	344	143	471
optGapGRB(%)	0.05	0.30	0.26	0.61	0.95	1.22	0.57	0.37	0.58
optGapScenPredOpt(%)	1.10	1.27	1.11	1.01	0.82	0.77	1.01	0.91	0.61
optGapPH(%)	0.08	0.06	0.05	0.06	0.11	0.22	0.10	0.04	0.15
optGapHeur(%)	2.83	2.83	2.30	2.58	2.37	2.11	2.50	2.39	0.97

trees and non-anticipativity constraints, two-stage problems are highly utilized for various applications. Our aim here is to show that our trained model can still be used as an out-of-the-box solution approach if a model has already been trained for a multi-stage problem.

Table 5.6 Average Results of Experiments for 2SMK with 8 Items

T	2	2	2	2	2	2	Avg	Mdn	Std
#sc	4	16	64	256	1,024	4,096	910	160	1,474
timeGRB	0.01	0.05	0.32	1.24	15.78	301.93	53.22	0.53	172.49
timeScenPredOpt	0.01	0.02	0.08	0.33	2.63	28.50	5.26	0.23	11.79
timePH	19.67	91.09	369.65	85.43	1,239.67	7,245.41	1,508.49	30.92	2,769.87
timeHeur	0.02	0.10	0.61	2.49	17.01	218.58	39.80	1.17	83.71
timeImpScenPredOpt	1	2	4	4	8	15	6	2	12
timeImpPH	0	0	0	0	0	0	0	0	0
timeImpHeur	0	0	1	1	1	1	1	0	1
optGapGRB(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
optGapScenPredOpt(%)	0.26	0.98	1.67	2.21	2.19	2.58	1.65	0.27	2.66
optGapPH(%)	0.60	0.11	0.21	0.11	0.11	0.56	0.28	0.00	0.59
optGapHeur(%)	3.76	4.27	4.26	5.05	4.10	2.77	4.03	2.50	4.68

Table 5.7 Average Results of Experiments for 2SMK with 10 Items

T	2	2	2	2	2	2	Avg	Mdn	Std
#sc	4	16	64	256	1,024	4,096	910	160	1,474
timeGRB	0.02	0.06	0.33	3.11	18.75	419.07	73.56	0.53	226.31
timeScenPredOpt	0.01	0.03	0.07	0.61	59.35	31.35	15.24	0.22	101.46
timePH	25.68	2.07	369.30	523.64	1,822.98	7,892.68	1,772.73	33.43	3,023.11
timeHeur	0.02	0.10	0.53	2.92	16.73	235.86	42.70	1.13	90.04
timeImpScenPredOpt	2	2	5	5	5	18	6	4	9
timeImpPH	0	0	0	0	0	0	0	0	0
timeImpHeur	1	1	1	1	1	2	1	1	1
optGapGRB(%)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
optGapScenPredOpt(%)	0.17	1.99	3.15	2.78	2.58	3.64	2.39	1.65	2.53
optGapPH(%)	0.33	0.13	0.08	0.05	0.04	0.92	0.26	0.00	0.51
optGapHeur(%)	2.75	4.29	3.58	4.58	3.28	3.98	3.75	2.77	3.81

Table 5.6 presents the results for the two-stage problem with 8 items, similar to Table 5.4. Table 5.7 demonstrates the results for 2SMK with 10 items, as in Table 5.5. For both Tables 5.6 and 5.7, the first three sets of instances are solved under a second with Gurobi since the number of scenarios and, therefore, the size of the problem is small. Even in these cases, the solution time by the ScenPredOpt is shorter than all other approaches. As instances get harder and larger, the benefits of the ScenPredOpt become clearer. For example, the last dataset of Table 5.6 is solved in more than 5 minutes on average. The ScenPredOpt reduces the average solution time below 30 seconds, achieving a faster solution than the Gurobi, heuristic, and PH. For all cases in Tables 5.6 and 5.7, the p-values for the Wilcoxon test are smaller than 0.01, confirming that the ScenPredOpt outperforms heuristic solution time. Also, the average optimality gap of ScenPredOpt is better than the heuristic for all cases in Tables 5.6 and 5.7, making it a superior solution approach. However, the optimality gaps for two-stage problems in Tables 5.6 and 5.7 seem to be higher than those in the multi-stage problems in Tables 5.4 and 5.5. This is because of the lack of periods required for attention structure in NEDA to capture problem characteristics, which

can likely be remedied by training the model using the solutions of two-stage problems. Also, the solution characteristics might be different for a two-stage problem than a multi-stage problem since, in a long-enough multi-stage problem, decisions tend to be cyclical. Nevertheless, the ScenPredOpt outperforms the heuristic for solving two-stage problems in terms of solution time and quality. It can be used to achieve significant solution time benefits for large-scale problems when a trained multi-stage model is available.

5.6.3 Generalization: Quality of Predictions for Item-wise Expansion Algorithm

In this section, we present the computational results for test sets with a larger number of items than the models are trained with. In the previous section, the results show that the models can maintain high-quality solutions with the increasing number of periods and scenarios. The expansion in the period dimension is organic since encoder-decoder models can handle variable-length input and output structures. Also, the presented NEDA model can handle a varying number of scenarios by averaging different groups of scenarios based on the scenario tree of the problem using Equation (5.5). However, predicting a problem with a different number of items is not straightforward with the NEDA since the predictions are made using a fixed-length output. Algorithm 5 presents an item-wise expansion algorithm that can generate predictions for a problem with a large number of items using a model trained with a small number of items. The main idea is to generate predictions for a subset of items multiple times with the trained model of a fixed number of items and combine them to generate a final prediction for all items considered in the test instance. This algorithm provides significant benefits since a trained model has the potential to make predictions for a problem with any number of periods, scenarios, and items for a certain distribution of input data. Therefore, the ScenPredOpt framework can

provide high levels of flexibility without generating training instances and performing training for the new set of instances with a larger number of items. Moreover, we present a comparison with the item-wise expansion strategy presented in Algorithm 1. The deviation-based item-wise expansion method is a further improvement over the one presented in Chapter 3 as described in Algorithm 1. For those results, we use metrics *timePredOpt*, *timeImpPredOpt*, *optGapPredOpt*, and *optGapPredOpt*.

Table 5.8 present the results of item-wise expansion for the SMCLSP. The first three sets of results are generated using the model trained with 8 items in Table 5.1, and the last three sets of results are generated using the model trained with 12 items in Table 5.2. The results demonstrate that the ScenPredOpt used within the item-wise expansion algorithm can provide high-quality solutions. For example, in the third set of instances with 16 items, predictions from the 8-item model yield a time improvement of 300, which is significantly more than the heuristic time improvement of 65. Also, the optimality gap of the ScenPredOpt is 0.09%, which is 20-times lower than the heuristic optimality gap of 1.84%. Also, in the first, second, fourth, and fifth datasets, the ScenPredOpt outperforms the SDDiP in terms of optimality gap as well as solution time. For all test sets in Table 5.8, the ScenPredOpt outperforms the heuristic in terms of solution time and optimality gap. Also, all p-values for the Wilcoxon signed-rank test are all lower than 0.01, confirming that the ScenPredOpt is statistically faster than the relax-and-fix heuristic. Moreover, ScenPredOpt outperforms PredOpt in terms of optimality gap in 3 out of 6 test sets, and has the same average optimality gap for the remaining 3 instances at the cost of reduced time improvement. Therefore, the improved item-wise algorithm achieves a better optimality gap compared to Algorithm 1 presented in Chapter 3.

Table 5.9 presents the item-wise expansion results for SMSMK similarly to Table 5.8. The first three results are produced with the 8-item model in Table 5.4, and the remaining three sets of results are produced using the 10-item model in Table 5.5.

Table 5.8 Average Results of Generalization Experiments for SMCLSP

T	15	10	10	15	10	10	Avg	Mdn	Std
#sc	32	64	81	32	64	81	59	64	20
Train Items		8			12		10	10	2
Test Items	24	32	16	36	48	24	30	28	10
timeGRB	1,954	3,708	6,273	4,380	2,664	7,200	4,363	7,200	3,439
timeScenPredOpt	36	166	188	93	93	107	114	40	299
timePredOpt	43	115	32	46	75	77	65	34	95
timeSDDiP	7,220	7,232	7,037	7,227	7,249	7,223	7,198	7,223	221
timeHeur	69	298	237	123	232	531	248	103	283
timeImpScenPredOpt	45	29	300	73	19	117	97	35	170
timeImpPredOpt	72	42	406	105	29	142	133	50	210
timeImpSDDiP	0	1	1	1	0	1	1	1	0
timeImpHeur	30	16	65	49	10	25	33	12	47
optGapGRB(%)	0.01	0.02	0.05	0.02	0.02	0.05	0.03	0.02	0.03
optGapScenPredOpt(%)	0.04	0.09	0.09	0.16	0.16	0.19	0.12	0.11	0.10
optGapPredOpt(%)	0.04	0.09	0.09	0.19	0.21	0.24	0.15	0.12	0.12
optGapSDDiP(%)	0.10	0.13	0.04	0.25	0.25	0.06	0.14	0.10	0.10
optGapHeur(%)	1.39	2.13	1.84	1.35	2.17	2.06	1.82	1.82	0.46

The results highlight that the models can be used to predict instances with a varying number of items coming from the same distribution using Algorithm 5. The instances presented in Table 5.9 do not terminate within set solution time limit of 7,200 seconds and achieve significant optimality gaps with Gurobi. For example, Gurobi achieves an optimality gap of 0.67% for the second dataset. The ScenPredOpt terminates in only 13 seconds on average and achieves a 0.70% optimality gap compared to the best solution found by Gurobi. For the same test set, the heuristic terminates in 27 seconds with a larger gap of 1.07%. On average, the ScenPredOpt reduces the solution time by more than 700 with an optimality gap below 1%. The heuristic has an average time improvement factor of 377 with a larger gap. We also conclude that the ScenPredOpt is faster than the heuristic since p-values for the Wilcoxon signed-rank test are below 0.01 except for the fourth test set. Additionally, the improved item-wise strategy

given in Algorithm 5 outperforms the strategy given by Algorithm 1 in terms of the optimality gap for all test sets in Table 5.9. On average, the optimality gap reduction with Algorithm 5 is more than 0.2% at an increased average solution time of 6 seconds.

Table 5.9 Average Results of Generalization Experiments for SMSMK

T	15	10	10	15	10	10	Avg	Mdn	Std
#sc	32	64	81	32	64	81	59	64	20
Train Items		8			10		9	9	1
Test Items	24	32	16	30	40	20	27	27	8
timeGRB	7,200	7,200	7,200	7,200	7,200	7,200	7,200	7,200	0
timeScenPredOpt	11	13	13	16	26	13	15	12	13
timePredOpt	8	7	8	9	13	8	9	7	5
timePH	8,925	7,430	3,148	7,358	9,866	4,540	6,878	7,243	6,058
timeHeur	18	27	21	14	29	19	21	20	7
timeImpScenPredOpt	875	716	809	685	381	752	703	604	440
timeImpPredOpt	1,043	1,261	1,328	1,114	721	1,101	1,095	970	535
timeImpPH	1	1	4	1	1	2	2	1	2
timeImpHeur	424	274	368	541	260	395	377	358	124
optGapGRB(%)	0.69	0.67	0.65	0.54	0.54	0.68	0.63	0.57	0.24
optGapScenPredOpt(%)	0.64	0.70	0.84	0.70	0.59	0.81	0.71	0.68	0.25
optGapPredOpt(%)	0.92	0.93	1.11	0.93	0.76	1.02	0.94	0.88	0.33
optGapPH(%)	0.26	0.34	0.10	0.29	0.62	0.08	0.28	0.12	0.33
optGapHeur(%)	1.29	1.07	1.82	1.04	0.80	1.48	1.25	1.15	0.48

In summary, the ScenPredOpt framework with the improved item-wise expansion algorithm can be used to predict instances with a varying number of items. This provides significant flexibility for model training since training can be performed using problems with a small number of items, which is easier to solve. Also, trained models do not need retraining with the changing number of items. Tables 5.8 and 5.9 provide the generalization results, in which the ScenPredOpt outperforms the heuristics in terms of solution time and quality in 11 out of 12 instances. Furthermore, Algorithm 5 reaches a better or the same optimality gap as Algorithm 1 at the cost of increased solution time.

5.7 Conclusions and Future Work

In this study, we presented a learning-enabled framework for solving scenario-based multi-stage stochastic programs. We address the challenge of non-anticipativity of predicted variables by developing the NEDA based on attentional encoder-decoder models. Also, generating labels for training instances can be untraceable considering significant training data requirements. We overcome this issue by sampling a scenario and solving this deterministic problem to optimality, which is used for training. Furthermore, we presented the ScenPredOpt algorithm to utilize predictions without causing any infeasibility. We integrate an LP-based heuristic to reduce the solution time further and to integrate heuristic solution capabilities that might not be captured with a learning methodology. We present the results of our framework by generating benchmark instances with a commercial solver, exact approaches, and heuristics. The results show that the solution time can be reduced by three orders of magnitude by achieving a small optimality gap of less than 1%. Additionally, we presented an improved item-wise expansion algorithm and tested if the models can be used to predict instances with a larger number of items than they are trained with. The results show that the trained models and ScenPredOpt provide a high amount of flexibility and can be utilized to solve instances with varying stages, scenarios, and items. Our framework is general and can be utilized to solve scenario-based multi-stage stochastic programs with binary variables where such problems need to be solved repeatedly quickly.

Our study shows a way to integrate learning models, heuristics, and existing mathematical solvers successfully to achieve significant solution time reductions. In this way, the best of each approach can be used. The scenario-based multi-stage stochastic programs have a wide range of applications in which learning-based frameworks can be exploited. Further studies can focus on solving multi-stage stochastic programs by integrating scenario generation mechanisms into the

framework. Also, risk-averse stochastic programs can be tackled by adjusting the NEDA model, scenario probabilities, and distribution of uncertain elements. Another direction relates to the integration of heuristics. Instead of general LP-based heuristics, problem-specific heuristics can be integrated into the ScenPredOpt solution framework to achieve better solutions faster.

CHAPTER 6

SUMMARY AND FUTURE DIRECTIONS

This dissertation focuses on using machine learning methodologies for solving optimization problems. We present novel methodologies to reduce the solution times of various types of repeatedly-solved operations research problems. We address the key issues of sequential dependence, infeasibility, generalization, and non-anticipativity.

In Chapter 2, we introduce a study on learning optimal solutions to sequential decision-making problems. Specifically, we utilize LSTMs to predict the values of binary decision variables of the CLSP, which is a fundamental problem in production planning. The bidirectional LSTM can process information in both time directions and, therefore, can learn the sequential dependencies that affect the decision structure of the CLSP. The trained LSTM model is used to generate predictions for an independent test. Within the LSTM-Opt framework, those predictions are used partially by adding selected variables as constraints to the problem. Depending on the hardness of the problem, using prediction at varying levels can be highly advantageous. For harder problems, using predictions at higher levels results in high levels of infeasibility, while using predictions at lower levels achieves significant time reductions without causing much infeasibility or an optimality gap. Unlike harder problems, easier problems benefit more from using predictions at higher levels without causing infeasibility or optimality gap since lower levels of predictions do not improve the solution time significantly. The computational results show that the solution time can be reduced from 70 hours to under 2 minutes with an optimality gap of less than 1% and without any infeasibility. We show that other fundamental ML approaches, such as logistic regression and random forest, cannot capture sequential dependencies as LSTM can, and thus, result in inferior solutions. Also, we have compared

our solution time with other exact approaches, including dynamic programming, dynamic programming-based inequalities, and (ℓ,S) inequalities. The results show that LSTM-Opt can provide solutions very fast at the cost of a small optimality gap, unlike traditional OR methodologies. Also, we present the results with time-wise generalization where the trained model is used to solve particularly longer instances than they are trained with.

In Chapter 3, we build and improve on the LSTM-Opt framework in Chapter 2 and present the PredOpt framework to solve complex sequential decision-making problems to reduce the solution times significantly. We address the feasibility and generalization issues arising LSTM-Opt. In the PredOpt framework, we propose a local attention-based encoder-decoder network to better capture and extend the sequential dependency. PredOpt reduces the computational cost with a local attention mechanism and enables PredOpt to selectively focus on a few close-by periods near the decision point. Therefore, PredOpt can learn from small-period problems to solve much-longer problems. At the core of the PredOpt framework lies a strategy that optimizes prediction levels to eliminate infeasibility. Within this framework, the prediction level is iteratively reduced, and a feasibility check is performed using a relaxation of the problem, which is also generated with the trained encoder-decoder model. The results show that the solution time can be reduced with a factor of 7,236, while achieving an optimality gap of 0.11%. PredOpt outperforms heuristics in terms of both solution quality and time, which is confirmed by a statistical test. Also, we present a novel item-wise expansion algorithm to provide another dimension of generalizability. The proposed algorithm can utilize a model trained with a few items and generate predictions for problems with a large number of items. The results highlight that the trained model can predict instances with much larger periods and items than they are trained with.

Chapter 4 presents a different perspective for using ML for OR by utilizing reinforcement learning. We propose a novel framework, 2SRL, based on deep reinforcement learning to solve scenario-based two-stage stochastic programming problems. Here, unlike learning methodologies in other chapters, the learning agent does not perform a supervised learning task. Recently, reinforcement learning has shown the potential to solve optimization problems. The introduced 2SRL framework tests the limits of recent advancements. In this framework, two different agents learn each stage of the problem sequentially. First, Agent 2 is trained to solve the second-stage problem. Then, Agent 1 is trained together with the feedback of the trained Agent 2 to solve the first-stage problem, through an updated policy gradient equation. We present detailed learning algorithms based on a well-known training paradigm. A pointer network is utilized to ensure generalizability with the item dimension. Also, the model can provide a stable solution quality with the increasing number of scenarios. The results show that the solution time can be reduced by five orders of magnitude with a gap of around 7%. The presented gaps are larger than supervised learning methodologies, but 2SRL provides a significantly better solution over a random solution and outperforms heuristics crafted for the problem in terms of solution time. The presented 2SRL framework can be utilized when an expert is not available to handcraft a heuristic for a fast solution.

Chapter 5 delivers another study based on supervised deep learning to solve scenario-based multi-stage stochastic programming problems. In this chapter, ScenPredOpt presents a novel attention-based encoder-decoder neural network to ensure the non-anticipativity constraints, which is a fundamental requirement for executability. Generating and solving training instances as stochastic programs can be computationally intractable. Therefore, we present a strategy for training based on deterministic instances. ScenPredOpt builds on PredOpt by integrating a heuristic based on linear programming relaxation to improve solution time reductions further.

To the best of our knowledge, this is the first study that tackles general multi-stage stochastic problems by integrating machine learning, heuristics, and mathematical solvers. The results show that the solution time can be reduced with a factor of 599, while achieving an optimality gap of 0.08%. Also, we present the results of predicting two-stage stochastic programs and highlight the superiority of supervised learning over reinforcement learning. Furthermore, the results provide detailed experiments on generalization in the item, stage, and scenario dimensions. In addition, we introduce an improved item-wise expansion algorithm that considers the variability in the predictions made. The proposed ScenPredOpt outperforms other exact and heuristic approaches and can be utilized to solve multi-stage stochastic programs in a fast manner.

In this dissertation, we presented frameworks that can be utilized for solving deterministic and stochastic sequential decision-making problems. Optimization problems with changing parameters are solved repeatedly in numerous practical applications, including manufacturing companies, the pharmaceutical industry, the healthcare industry, finance, airline scheduling, and energy production. Since the problems that are solved within those domains are getting more complex as the data availability increases, a significant benefit can be achieved if they are solved significantly faster. While special solution methodologies can be developed for such problems, it can be time-consuming since it requires expert knowledge and constant adaptation. Therefore, a streamlined process to generate high-quality solutions very quickly can be highly advantageous. Such methodologies are designed in this dissertation to handle lot-sizing and knapsack, which have general mixed-integer and binary structures, in deterministic or stochastic settings with a varying number of variables and constraints.

Another important finding is related to the generalization characteristics of designed learning-based frameworks. Generalization is of great importance since such

properties can provide robustness to proposed frameworks. With the generalization property established, the proposed frameworks can be utilized to solve problems with a varying number of items, stages, and scenarios. Furthermore, generalization can be a foundation point for training since it can provide more flexibility with the training. In this dissertation, we present extensive experimentation and novel algorithms to institute generalization characteristics.

Also, we presented novel architectures that integrate learning, heuristics, and mathematical solvers. Therefore, we aim to bring out the strengths of different methodologies and eliminate the disadvantages. In this dissertation, we address core problems of operations research, such as infeasibility and non-anticipativity. We undertake those issues with the designed ML-OR integration structures and achieve a notable reduction in solution time while maintaining solution quality.

The future holds a lot of potential applications for the growing field of using ML for solving OR. One of the key challenges is to ensure feasibility. Therefore, novel approaches involving variable selection paradigms after generating predictions can be developed to ensure feasibility. Another possible approach could involve incorporating constraints into the learning model. Further studies can investigate special solution requirements of stochastic programs. The non-anticipativity is a crucial property and can be accounted for using post-processing techniques after generating predictions instead of generating predictions that are non-anticipative. Also, the structure of the scenario tree or the distribution of the uncertainty can be directly incorporated into the learning paradigm. The scenarios with different weights can be included for more testing. Also, risk-averse programs can be studied with proposed frameworks, and special solution structures can be developed.

Another future direction involves improving the reinforcement learning-based solution strategies. While they can generate relatively high-quality solutions, supervised learning-based frameworks appeared to have an overall better solution

quality. Therefore, different strategies can be incorporated to further enhance the reinforcement learning paradigm. For example, a training strategy both using reinforcement learning and supervised learning iteratively can improve the learning process while reducing the computational cost. Therefore, the learner can get stronger feedback on the decision quality. Also, reinforcement learning can be used together with mathematical solvers to decide on hard-to-predict variables. Further experiments can involve investigation with different network architectures, such as transformers and various learning paradigms.

Furthermore, generalization can be a point for future study. Novel approaches can be developed for out-of-distribution generalization. In addition, more experimentation can be provided with different types of optimization problems. Also, the learning paradigm can be introduced for various tasks such as heuristic selection, adjusting solver settings, and branching decisions instead of directly predicting decision variables. Additionally, problem-specific heuristics can provide notable benefits over general heuristics within an integrated model. Further studies can consider structural modifications for interactions of learning, heuristics, and mathematical solvers to exploit the integrated structures. Also, learning-based methodologies can be integrated with existing exact solution approaches, such as cutting planes.

APPENDIX A

MODEL TRAINING TIMES AND FURTHER EXPERIMENTS FOR CHAPTER 2

A.1 Results for Training LSTM Models

For each dataset, several LSTM models are trained, and the best achieving model is selected with a process called hyperparameter tuning, as described in Section 2.5.2. Table A.1 presents the LSTM training time in CPU seconds for different datasets. The training times do not follow an identifiable pattern with c; however, they show an increasing trend as each of the T and f parameters increases. For example, the instances are solved faster when $c = 8$ compared to the instances with $c = 3$, but the LSTM training time is higher in three out of four datasets with $c = 8$, indicating that training time is not positively correlated with the hardness of the instances for a given T and f. The training times presented in Table A.1 can be decreased significantly with the use of graphics processing units (GPUs). Those training times are not included in the timeML, timeimp, and timegain(%) because the training of the models can be performed in an offline setting. However, as mentioned above, the prediction generation time using the trained LSTM model is included in the timeML, timeimp, and timegain(%). The training times were 3.9, 4.4, and 5.2 seconds for the logistic regression and 4,430.6, 4,459.4, and 4,186.8 seconds for the random forest model. Even though they are significantly lower than the LSTM training times, their performance is inferior to the LSTM-Opt framework.

A.2 More Results on the Experiments

Table A.2 presents the results for the instances with $T = 90$ and $f = 1,000$, which are relatively easier than the instances with $T = 120$ and $f = 10,000$ in Table 2.1. It can be seen that as the value of capacity multiplier c increases from 3 to 5, the solution

Table A.1 LSTM Training Times for the Model with the Highest Validation Accuracy (in CPU Seconds)

f	T	c	Training Time
1,000	90	3	27,554
		5	19,726
		8	54,648
10,000	90	3	32,086
		5	27,406
		8	70,387
1,000	120	3	37,794
		5	28,913
		8	72,779
10,000	120	3	65,791
		5	75,147
		8	52,724

time decreases. As more variables are predicted, the solution time gain increases considerably with an increase in the number of infeasible instances in the test set. Predicting all binary variables (pred(%)=100) reduces the mean solution time by more than half, but the reduction in solution time comes at a price, which is 1.45% infeasibility in all CLSP instances and 0.13% average optimality gap. In other words, 290 instances had infeasible predictions among 20,000 CLSP test instances, and the average objective value of the feasibly-predicted instances only deviated by 0.13% from the average optimal objective value by CPLEX. The user cuts approach with 100% of variables predicted achieves a time improvement of 2, with zero infeasibility and a slightly higher optimality gap. In the dataset with $c = 5$ of Table A.2, time improvements are similar to the previous dataset with $c = 3$, but for the same level of prediction, we observe less number of instances in the test set for which the predictions are infeasible. When $c = 8$, a low-level prediction increases the average solution

time, but the complete prediction of binary variables (pred(%)=100) achieves zero optimality gap, with only 0.01% of the test instances having infeasible solutions, and solution time is reduced by a factor of 3. The instances presented in Table A.2 are the easiest among all datasets. For those instances, predicting the majority of binary variables could be the most beneficial approach, reducing the solution time without a significant increase in the optimality gap or infeasibility.

Table A.3 presents the results for instances with $T = 90$ and $f = 10,000$. These are harder than the instances shown in Table A.2, as seen from the mean solution time. For the instances with $c = 3$, we observe that predicting 50% of variables reduces the average solution time by a factor of 3, with no infeasibility in the test set, and zero optimality gap. However, as more variables are predicted, solutions may become significantly infeasible. For example, at the full (100%) prediction of the binary variables, more than half of the predictions are infeasible, which is not desired. The user cuts approach remedies this problem by resulting in all feasible solutions for all 20,000 test instances with a significant reduction in the solution time. The dataset with $c = 5$ follows a similar pattern to the dataset with $c = 3$ but demonstrates less infeasibility at the higher level of predictions. Results for the dataset with $c = 8$ are similar to the instances with the same c in Table A.2. Here, the approach that predicts the variables at higher levels achieves the most computational gain. For example, with the 75% prediction, the solution time is improved by a factor of 2, without any loss in optimality or feasibility. It can be seen from the results that predicting variables at higher levels can be beneficial for easier instances in terms of the time gain without sacrificing feasibility or optimality.

Table A.4 demonstrates the results for instances with $T = 120$ and $f = 1,000$, which have a similar solution time to the instances $T = 90$ and $f = 1,000$ in Table A.2. Also, the quality of the predictions follows a similar pattern, as in Table A.2. For all datasets, predictions at the lower proportions do not provide

Table A.2 Summary of Experiments for $f = 1,000$ and $T = 90$

c	pred(%)	timeCPX	timeML	timeimp	timegain(%)	inf(%)	optgap(%)
3	25	0.4	0.4	1	4.7	0.0	0.0
	50		0.3	1	14.1	0.0	0.0
	75		0.3	1	16.6	0.0	0.0
	85		0.3	1	24.4	0.0	0.0
	90		0.3	2	37.0	0.0	0.0
	95		0.2	2	47.0	0.2	0.0
	100		0.1	3	66.8	1.4	0.1
	100(MS)		0.4	1	13.0	0.0	0.0
	100(UC)		0.3	2	37.4	0.0	0.1
5	25	0.3	0.3	1	7.9	0.0	0.0
	50		0.3	1	10.0	0.0	0.0
	75		0.3	1	17.5	0.0	0.0
	85		0.2	1	22.9	0.0	0.0
	90		0.2	1	25.1	0.0	0.0
	95		0.2	1	32.8	0.0	0.0
	100		0.1	3	65.7	0.4	0.0
	100(MS)		0.4	1	-10.7	0.0	0.0
	100(UC)		0.3	1	16.4	0.0	0.0
8	25	0.3	0.4	1	-5.8	0.0	0.0
	50		0.3	1	-5.6	0.0	0.0
	75		0.3	1	0.3	0.0	0.0
	85		0.3	1	6.6	0.0	0.0
	90		0.3	1	14.7	0.0	0.0
	95		0.3	1	20.2	0.0	0.0
	100		0.1	3	70.1	0.0	0.0
	100(MS)		0.3	1	4.5	0.0	0.0
	100(UC)		0.3	1	19.8	0.0	0.0

Table A.3 Summary of Experiments for $f = 10,000$ and $T = 90$

c	pred(%)	timeCPX	timeML	timeimp	timegain(%)	inf(%)	optgap(%)
3	25	3.1	1.9	2	38.1	0.0	0.0
	50		1.1	3	63.0	0.0	0.0
	75		0.4	8	87.9	0.4	0.1
	85		0.3	12	91.4	2.0	0.2
	90		0.2	14	92.8	5.3	0.3
	95		0.2	15	93.4	17.5	0.9
	100		0.1	22	95.6	53.9	1.7
	100(MS)		3.9	1	-28.2	0.0	0.0
	100(UC)		0.3	10	90.3	0.0	1.2
5	25	1.5	1.4	1	8.6	0.0	0.0
	50		0.8	2	46.7	0.0	0.0
	75		0.4	4	71.7	0.1	0.0
	85		0.3	5	78.7	0.3	0.0
	90		0.3	5	81.3	0.9	0.1
	95		0.3	6	82.9	4.1	0.3
	100		0.1	17	94.0	23.3	1.2
	100(MS)		1.5	1	5.7	0.0	0.0
	100(UC)		0.3	5	78.4	0.0	0.7
8	25	0.9	0.7	1	22.0	0.0	0.0
	50		0.6	2	36.2	0.0	0.0
	75		0.4	2	52.2	0.0	0.0
	85		0.3	3	64.6	0.0	0.0
	90		0.3	3	70.9	0.1	0.0
	95		0.2	4	74.3	0.5	0.1
	100		0.1	11	90.6	4.9	0.8
	100(MS)		0.8	1	10.8	0.0	0.0
	100(UC)		0.3	3	67.2	0.0	0.3

significant reductions in solution time, while predicting up to a greater extent can yield noteworthy improvement. For $c = 3$ instances, predicting 100% of variables results in a significant solution time improvement of 4, with only 1.3% of the instances in the test set becoming infeasible. The user cuts approach provides time improvement of 2 without causing infeasibility in any of the 20,000 test instances. For $c = 5$ instances, a significant time improvement is achieved by the full prediction of variables at the cost of 0.35% infeasibility in the test set and 0.04% optimality gap. For the $c = 8$ dataset, the full prediction results in a significant time improvement of 4 with a 0.02% infeasibility and 0.01% optimality gap. The user cuts achieve a small solution time reduction without any infeasibility in the test set.

A.3 Predicting Instances with Different Distributions

Table A.5 presents the results on how the trained LSTM models generalize to the instances with different distributions. In the first dataset, we examine the predictive performance of the LSTM model trained on instances with $c = 3$, $f = 1,000$, and $T = 90$ on the dataset with the same c and T, but $f = 10,000$. Here, the LSTM is trained on easier instances and used to predict the harder instances. We observe a time improvement of 2 at the 25% prediction level, with an optimality gap under 0.5% without any infeasibility in the test set. As the prediction level increases, time improvement increases significantly at the cost of an increased optimality gap. For example, at the 75% prediction level, we improve the CPLEX solution time with a factor of 10 without any infeasibility but with an optimality gap of 3.8%. In the next dataset, we examine the opposite scenario, where instances are trained on a harder dataset with $f = 10,000$ to predict easier instances with $f = 1,000$. At the lower proportions of predictions, the time improvement does not increase significantly, but predictions do also not cause any infeasible solutions. As the level of predictions increases, both the percent infeasibility in the test set and the optimality gap increase

Table A.4 Summary of Experiments for $f = 1,000$ and $T = 120$

c	pred(%)	timeCPX	timeML	timeimp	timegain(%)	inf(%)	optgap(%)
3	25	0.4	0.4	1	13.9	0.0	0.0
	50		0.4	1	16.2	0.0	0.0
	75		0.4	1	16.6	0.0	0.0
	85		0.3	2	36.3	0.0	0.0
	90		0.3	2	40.8	0.0	0.0
	95		0.2	2	49.3	0.2	0.0
	100		0.1	4	73.2	1.3	0.1
	100(MS)		0.4	1	19.9	0.0	0.0
	100(UC)		0.3	2	38.9	0.0	0.1
5	25	0.3	0.3	1	6.4	0.0	0.0
	50		0.3	1	10.9	0.0	0.0
	75		0.3	1	15.0	0.0	0.0
	85		0.3	1	18.3	0.0	0.0
	90		0.3	1	24.9	0.0	0.0
	95		0.3	1	21.1	0.0	0.0
	100		0.1	3	68.4	0.4	0.0
	100(MS)		0.3	1	-3.6	0.0	0.0
	100(UC)		0.3	1	17.4	0.0	0.0
8	25	0.3	0.3	1	-5.4	0.0	0.0
	50		0.3	1	0.2	0.0	0.0
	75		0.3	1	4.7	0.0	0.0
	85		0.3	1	2.1	0.0	0.0
	90		0.3	1	5.5	0.0	0.0
	95		0.2	1	21.8	0.0	0.0
	100		0.1	4	72.2	0.0	0.0
	100(MS)		0.4	1	-17.8	0.0	0.0
	100(UC)		0.3	1	14.8	0.0	0.0

significantly. The UC remedies the infeasibility and achieves an optimality gap slightly above 5%.

Table A.5 Summary of Generalization Experiments to Test Datasets with Different Characteristics

LSTM Train			Test Data			pred	timeCPX	timeML	timeimp	timegain	inf	optgap
c	f	T	c	f	T	(%)				(%)	(%)	(%)
3	1,000	90	3	10,000	90	25	3.1	1.8	2	42.6	0.0	0.3
						50		0.9	3	70.8	0.0	1.0
						75		0.3	10	89.7	0.0	3.8
						85		0.2	12	91.9	0.0	6.0
						90		0.2	14	92.9	0.0	8.0
						95		0.2	17	94.1	0.2	10.5
						100		0.1	30	96.6	1.1	13.6
						100(UC)		0.3	10	90.5	0.0	5.3
3	10,000	90	3	1,000	90	25	0.4	0.4	1	5.6	0.0	0.2
						50		0.3	1	16.1	0.0	1.2
						75		0.3	1	21.2	0.5	4.7
						85		0.3	1	16.0	2.1	7.1
						90		0.3	1	21.4	5.6	9.4
						95		0.3	1	31.3	17.8	13.3
						100		0.1	5	78.7	54.3	16.4
						100(UC)		0.3	1	27.2	0.0	5.6
3	1,000	120	5	10,000	120	25	3.0	1.4	2	53.9	0.0	2.1
						50		0.5	6	82.4	0.0	5.8
						75		0.2	12	91.8	0.0	22.8
						85		0.2	15	93.4	0.0	33.0
						90		0.2	16	93.7	0.0	38.7
						95		0.2	17	94.1	0.0	44.7
						100		0.1	28	96.4	0.0	51.5
						100(UC)		0.3	12	91.4	0.0	26.5
5	10,000	120	3	1,000	120	25	0.4	0.4	1	5.7	8.6	1.3
						50		0.3	1	23.0	33.3	13.7
						75		0.3	2	41.5	81.0	117.8
						85		-	-	-	100.0	-
						90		-	-	-	100.0	-
						95		-	-	-	100.0	-
						100		-	-	-	100.0	-
						100(UC)		0.3	2	40.4	0.0	11.8

The next two datasets have completely different underlying distributions. The LSTM model is trained on easier instances with $c = 3$, $f = 1,000$ and $T = 120$ to predict harder instances with $c = 5$, $f = 10,000$ and $T = 120$. At the 25% prediction level, a time improvement of 2 is achieved at the cost of an optimality gap, which is slightly more than 2%, without any infeasibility in the test set. As the prediction level increases, the optimality gap increases significantly, but solutions remain highly feasible. In the opposite scenario the LSTM model is trained on instances with $c = 5$, $f = 10,000$ and $T = 120$ to predict the instances with $c = 3$, $f = 1,000$ and $T = 120$. Using an entirely different underlying distribution to make predictions leads to an increased level of infeasibility in the test set, where the prediction level of more than 75% causes all infeasible solutions. Therefore, for those predictions, we recommend a lower-level prediction (e.g., 25%) or the use of the CPLEX user cuts approach to remedy the infeasibility problem.

Overall, training on the easier instances to predict harder instances with a similar c value can provide good results at the low level of predictions or with the UC approach. For example, the LSTM model trained with $c = 3$, $f = 1,000$, and $T = 90$ improves the CPLEX solution time with a factor of 3 without any infeasibility and with an optimality gap of 1% at the 50% prediction level when predicting the instances with $c = 3$, $f = 10,000$ and $T = 90$.

APPENDIX B

DETAILS OF THE TEST INSTANCES FOR CHAPTER 5

Here, we present the details of our testing instances for each table in Section 5.6. Table B.1 gives the details for SMCLSP instances presented in Tables 5.1, 5.2, and 5.8. Table B.2 gives the details for SMSMK instances presented in Tables 5.4, 5.5, and 5.9. We denote the number of scenarios $\#sc$, the number of scenarios per period $\#scPerPeriod$, and the period where no further scenarios are generated for the problem as $capPeriod$. These three variables interconnect as: $\#sc = \#scPerPeriod^{capPeriod-1}$. We have utilized a scenario capping strategy to combat the exponentially growing number of scenarios. Additionally, the number of binary variables is denoted by $\#binaryVars$ and the number of continuous variables is denoted by $\#contVars$. Note that SMSMK does not contain any continuous variables. We also present the total number of constraints excluding variable restrictions as $\#const$ and the number of non-anticipativity constraints as $\#nonAntConst$, and the number of items in the test set as $testItems$.

Table B.1 Details of Test Instances for SMCLSP

Table	T	#sc	#scPerPeriod	capPeriod	#binaryVars	#contVars	#const	#nonAntConst	testItems
5.1	10	32	2	6	2,560	5,120	8,536	3,096	8
	10	64	4	4	5,120	10,240	14,984	4,104	8
	10	81	3	5	6,480	12,960	20,586	6,816	8
	15	125	5	4	15,000	30,000	40,131	8,256	8
	15	243	3	6	29,160	58,320	88,221	26,256	8
	15	512	2	10	61,440	122,880	228,888	98,328	8
5.2	10	32	2	6	3,840	7,680	12,644	4,644	12
	10	64	4	4	7,680	15,360	22,156	6,156	12
	10	81	3	5	9,720	19,440	30,474	10,224	12
	15	125	5	4	22,500	45,000	59,259	12,384	12
	15	243	3	6	43,740	87,480	130,509	39,384	12
	15	512	2	10	92,160	184,320	339,492	147,492	12
5.8	15	32	2	6	11,520	23,040	32,808	9,288	24
	10	64	4	4	20,480	40,960	58,016	16,416	32
	10	81	3	5	12,960	25,920	40,362	13,632	16
	15	32	2	6	17,280	34,560	48,972	13,932	36
	10	64	4	4	30,720	61,440	86,704	24,624	48
	10	81	3	5	19,440	38,880	60,138	20,448	24

Table B.2 Details of Test Instances for SMSMK

Table	T	#sc	#scPerPeriod	capPeriod	#binaryVars	#const	#nonAntConst	testItems
5.4	10	32	2	6	4,864	8,272	2,064	8
	10	64	4	4	9,728	15,152	2,736	8
	10	81	3	5	12,312	20,258	4,544	8
	15	125	5	4	29,000	42,879	5,504	8
	15	243	3	6	56,376	90,161	17,504	8
	15	512	2	10	118,784	218,640	65,552	8
5.5	10	32	2	6	6,080	9,620	2,580	10
	10	64	4	4	12,160	17,500	3,420	10
	10	81	3	5	15,390	23,500	5,680	10
	15	125	5	4	36,250	49,380	6,880	10
	15	243	3	6	70,470	104,500	21,880	10
	15	512	2	10	148,480	256,020	81,940	10
5.6	2	4	-	-	72	89	0	8
	2	16	-	-	264	341	0	8
	2	64	-	-	1,032	1,349	0	8
	2	256	-	-	4,104	5,381	0	8
	2	1,024	-	-	16,392	21,509	0	8
	2	4,096	-	-	65,544	86,021	0	8
5.7	2	4	-	-	90	100	0	10
	2	16	-	-	330	388	0	10
	2	64	-	-	1,290	1,540	0	10
	2	256	-	-	5,130	6,148	0	10
	2	1,024	-	-	20,490	24,580	0	10
	2	4,096	-	-	81,930	98,308	0	10
5.9	15	32	2	6	22,272	30,096	6,192	24
	10	64	4	4	38,912	51,008	10,944	32
	10	81	3	5	24,624	36,466	9,088	16
	15	32	2	6	27,840	36,540	7,740	30
	10	64	4	4	48,640	62,320	13,680	40
	10	81	3	5	30,780	43,760	11,360	20

REFERENCES

Abbasi, B., Babaei, T., Hosseinifard, Z., Smith-Miles, K., and Dehghani, M. (2020). Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management. *Computers & Operations Research*, 119:104941.

Absi, N. and van den Heuvel, W. (2019). Worst-case analysis of relax and fix heuristics for lot-sizing problems. *European Journal of Operational Research*, 279(2):449–458.

Accorsi, L., Lodi, A., and Vigo, D. (2022). Guidelines for the computational testing of machine learning approaches to vehicle routing problems. *Operations Research Letters*, 50(2):229–234.

Afshar, R. R., Zhang, Y., Firat, M., and Kaymak, U. (2020). 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.

Ahmed, S. (2010). Two-stage stochastic integer programming: A brief introduction. *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ: Wiley.

Ahmed, S., Tawarmalani, M., and Sahinidis, N. V. (2004). A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377.

Akçay, Y., Li, H., and Xu, S. H. (2007). Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150(1):17–29.

Alpaydin, E. (2020). *Introduction to machine learning.* Cambridge, MA: MIT Press.

Anderson, L., Turner, M., and Koch, T. (2022). Generative deep learning for decision making in gas networks. *Mathematical Methods of Operations Research*, 95(3):503–532.

Angulo, G., Ahmed, S., and Dey, S. S. (2016). Improving the integer l-shaped method. *INFORMS Journal on Computing*, 28(3):483–499.

Atamtürk, A. and Muñoz, J. C. (2004). A study of the lot-sizing polytope. *Mathematical Programming*, 99(3):443–465.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A., Garcia, P. M., Jain, A., Luo, R., Maggiar, A., Narayanaswamy, B., and Ye, C. (2019). Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*.

Bampis, E., Escoffier, B., and Kononov, A. (2020). Lp-based algorithms for multistage minimization problems. In *International Workshop on Approximation and Online Algorithms*, pages 1–15. Cham, Switzerland: Springer.

Bampis, E., Escoffier, B., and Teiller, A. (2022). Multistage knapsack. *Journal of Computer and System Sciences*, 126:106–118.

Barany, I., Van Roy, T. J., and Wolsey, L. A. (1984). Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10):1255–1261.

Barbarosoğlu, G. and Arda, Y. (2004). A two-stage stochastic programming framework for transportation planning in disaster response. *Journal of the Operational Research Society*, 55(1):43–53.

Barnhart, C., Belobaba, P., and Odoni, A. R. (2003). Applications of operations research in the air transport industry. *Transportation Science*, 37(4):368–391.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y., Frejinger, E., Lodi, A., Patel, R., and Sankaranarayanan, S. (2020). 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.

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

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks / a Publication of the IEEE Neural Networks Council*, 5:157–66.

Beraldi, P., Ghiani, G., Grieco, A., and Guerriero, E. (2006). Fix and relax heuristic for a stochastic lot-sizing problem. *Computational Optimization and Applications*, 33(2):303–318.

Bertsimas, D. and Demir, R. (2002). An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science*, 48(4):550–565.

Bertsimas, D. and Stellato, B. (2021). The voice of optimization. *Machine Learning*, 110(2):249–277.

Bertsimas, D. and Stellato, B. (2022). Online mixed-integer optimization in milliseconds. *INFORMS Journal on Computing*, 34(4):2229–2248.

Birge, J. R. and Louveaux, F. V. (1988). A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392.

Birge, J. R. and Louveaux, F. V. (2011). *Introduction to stochastic programming*. New York, NY: Springer.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford, England: Oxford university press.

Bitran, G. R. and Yanasse, H. H. (1982). Computational complexity of the capacitated lot size problem. *Management Science*, 28(10):1174–1186.

Bonami, P., Lodi, A., and Zarpellon, G. (2018). Learning a classification of mixed-integer quadratic programming problems. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604. Cham, Switzerland: Springer.

Boonmee, A. and Sethanan, K. (2016). A glnpso for multi-level capacitated lot-sizing and scheduling problem in the poultry industry. *European Journal of Operational Research*, 250(2):652–665.

Bruno, G., Genovese, A., and Piccolo, C. (2014). The capacitated lot sizing model: A powerful tool for logistics decision making. *International Journal of Production Economics*, 155:380–390.

Bushaj, S. and Büyüktahtakın, İ. E. (2022). A k-means supported reinforcement learning algorithm to solve multi-dimensional knapsack problem. *Under Review for INFORMS Journal of Global Optimization*.

Bushaj, S., Büyüktahtakın, İ. E., and Haight, R. G. (2022a). 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.

Bushaj, S., Büyüktahtakın, İ. E., Yemshanov, D., and Haight, R. G. (2021). Optimizing surveillance and management of emerald ash borer in urban environments. *Natural Resource Modeling*, 34(1):e12267.

Bushaj, S., Yin, X., Beqiri, A., Andrews, D., and Büyüktahtakın, İ. E. (2022b). A simulation-deep reinforcement learning (sirl) approach for epidemic control optimization. *Annals of Operations Research*.

Buşoniu, L., Babuška, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172.

Buşoniu, L., Babuška, R., and De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. *Innovations in Multi-agent Systems and Applications-1*, 1:183.

Büyüktahtakın, İ. E. (2011). Dynamic programming via linear programming. *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ: Wiley.

Büyüktahtakın, İ. E. (2022). Stage-t scenario dominance for risk-averse multi-stage stochastic mixed-integer programs. *Annals of Operations Research*, 309(1):1–35.

Büyüktahtakın, İ. E., des Bordes, E., and Kıbış, E. Y. (2018a). A new epidemics–logistics model: Insights into controlling the ebola virus disease in West Africa. *European Journal of Operational Research*, 265(3):1046–1063.

Büyüktahtakın, İ. E., Feng, Z., Frisvold, G., Szidarovszky, F., and Olsson, A. (2011). A dynamic model of controlling invasive species. *Computers & Mathematics with Applications*, 62(9):3326–3333.

Büyüktahtakin, İ. E., Feng, Z., Olsson, A. D., Frisvold, G., and Szidarovszky, F. (2014). Invasive species control optimization as a dynamic spatial process: an application to buffelgrass (pennisetum ciliare) in arizona. *Invasive Plant Science and Management*, 7(1):132–146.

Büyüktahtakın, İ. E., Feng, Z., and Szidarovszky, F. (2014a). A multi-objective optimization approach for invasive species control. *Journal of the Operational Research Society*, 65(11):1625–1635.

Büyüktahtakın, İ. E. and Haight, R. G. (2018). A review of operations research models in invasive species management: state of the art, challenges, and future directions. *Annals of Operations Research*, 271(2):357–403.

Büyüktahtakın, İ. E. and Hartman, J. C. (2016). A mixed-integer programming approach to the parallel replacement problem under technological change. *International Journal of Production Research*, 54(3):680–695.

Büyüktahtakın, İ. E., Kıbış, E. Y., Cobuloglu, H. I., Houseman, G. R., and Lampe, J. T. (2015). An age-structured bio-economic model of invasive species management: insights and strategies for optimal control. *Biological Invasions*, 17(9):2545–2563.

Büyüktahtakın, İ. E. and Liu, N. (2016). Dynamic programming approximation algorithms for the capacitated lot-sizing problem. *Journal of Global Optimization*, 65(2):231–259.

Büyüktahtakın, İ. E., Smith, J. C., and Hartman, J. C. (2018b). Partial objective inequalities for the multi-item capacitated lot-sizing problem. *Computers & Operations Research*, 91:132–144.

Büyüktahtakın, İ. E., Smith, J. C., Hartman, J. C., and Luo, S. (2014b). Parallel asset replacement problem under economies of scale with multiple challengers. *The Engineering Economist*, 59(4):237–258.

Cacchiani, V., Hemmelmayr, V. C., and Tricoire, F. (2014). A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 163:53–64.

Cacchiani, V., Iori, M., Locatelli, A., and Martello, S. (2022). Knapsack problems-an overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, 143:105693.

Carøe, C. C. and Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24(1-2):37–45.

Carrión, M., Philpott, A. B., Conejo, A. J., and Arroyo, J. M. (2007). A stochastic programming approach to electric energy procurement for large consumers. *IEEE Transactions on Power Systems*, 22(2):744–754.

Chen, X. and Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32:6281–6292.

Chen, Y. and Hao, J.-K. (2014). A "reduce and solve" approach for the multiple-choice multidimensional knapsack problem. *European Journal of Operational Research*, 239(2):313–322.

Chen, Z.-L., Li, S., and Tirupati, D. (2002). A scenario-based stochastic programming approach for technology and capacity planning. *Computers & Operations Research*, 29(7):781–806.

Chius, S. Y., Lu, L., and Cox Jr, L. A. (1996). Optimal access control for broadband services: stochastic knapsack with advance information. *European Journal of Operational Research*, 89(1):127–134.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. *Advances in Neural Information Processing Systems*, 28:577–585.

Cobuloglu, H. I. and Büyüktahtakın, İ. E. (2014). A mixed-integer optimization model for the economic and environmental analysis of biomass production. *Biomass and Bioenergy*, 67:8–23.

Cobuloglu, H. I. and Büyüktahtakın, İ. E. (2015a). Food vs. biofuel: An optimization approach to the spatio-temporal analysis of land-use competition and environmental impacts. *Applied Energy*, 140:418–434.

Cobuloglu, H. I. and Büyüktahtakın, İ. E. (2015b). A stochastic multi-criteria decision analysis for sustainable biomass crop selection. *Expert Systems with Applications*, 42(15-16):6065–6074.

Cobuloglu, H. I. and Büyüktahtakın, İ. E. (2017). A two-stage stochastic mixed-integer programming approach to the competition of biofuel and food production. *Computers & Industrial Engineering*, 107:251–263.

Cohen, E., Cormode, G., Duffield, N., and Lund, C. (2016). On the tradeoff between stability and fit. *ACM Transactions on Algorithms*, 13(1):1–24.

Cohn, A. M. and Barnhart, C. (1998). 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.

Copil, K., Wörbelauer, M., Meyr, H., and Tempelmeier, H. (2017). Simultaneous lot-sizing and scheduling problems: a classification and review of models. *OR Spectrum*, 39(1):1–64.

Coşgun, Ö. and Büyüktahtakın, İ. E. (2018). Stochastic dynamic resource allocation for hiv prevention and treatment: An approximate dynamic programming approach. *Computers & Industrial Engineering*, 118:423–439.

Crespo-Vazquez, J. L., Carrillo, C., Diaz-Dorado, E., Martinez-Lorenzo, J. A., and Noor-E-Alam, M. (2018). A machine learning based stochastic optimization framework for a wind and storage power plant participating in energy pool market. *Applied Energy*, 232:341–357.

Cygan, M., Jeż, Ł., and Sgall, J. (2016). Online knapsack revisited. *Theory of Computing Systems*, 58(1):153–190.

Dantzig, G. B. (1955). Linear programming under uncertainty. *Management Science*, 1(3-4):197–206.

Delarue, A., Anderson, R., and Tjandraatmadja, C. (2020). Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33:609–620.

Denizel, M. and Süral, H. (2006). On alternative mixed integer programming formulations and LP-based heuristics for lot-sizing with setup times. *Journal of the Operational Research Society*, 57(4):389–399.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Cham, Switzerland: Springer.

Ding, J.-Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., and Song, L. (2020). Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1452–1459. Palo Alto, CA: Association for the Advancement of Artificial Intelligence.

Ding, L., Ahmed, S., and Shapiro, A. (2019). A python package for multi-stage stochastic programming. *Optimization Online*.

Donti, P. L., Rolnick, D., and Kolter, J. Z. (2021). Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*.

Dumouchelle, J., Patel, R., Khalil, E. B., and Bodur, M. (2022). Neur2sp: Neural two-stage stochastic programming. *arXiv preprint arXiv:2205.12006*.

Eck, D. and Schmidhuber, J. (2002). Learning the long-term structure of the blues. In *International Conference on Artificial Neural Networks*, pages 284–289. Heidelberg, Germany: Springer.

Eisenstat, D., Mathieu, C., and Schabanel, N. (2014). Facility location in evolving metrics. In *Automata, Languages, and Programming*, pages 459–470. Heidelberg, Germany: Springer.

Eppen, G. D. and Martin, R. K. (1987). Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848.

Fábián, C. I. and Szőke, Z. (2007). Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, 4(4):313–353.

Feng, Y., Niazadeh, R., and Saberi, A. (2021). 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.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181.

Finnah, B., Gönsch, J., and Ziel, F. (2022). Integrated day-ahead and intraday self-schedule bidding for energy storage systems using approximate dynamic programming. *European Journal of Operational Research*, 301(2):726–746.

Fischetti, M. and Fraccaro, M. (2019). Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. *Computers & Operations Research*, 106:289–297.

Florian, M., Lenstra, J. K., and Rinnooy Kan, A. (1980). Deterministic production planning: Algorithms and complexity. *Management Science*, 26(7):669–679.

Frejinger, E. and Larsen, E. (2019). A language processing algorithm for predicting tactical solutions to an operational planning problem under uncertainty. *arXiv preprint arXiv:1910.08216*.

Gade, D., Hackebeil, G., Ryan, S. M., Watson, J.-P., Wets, R. J.-B., and Woodruff, D. L. (2016). Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming*, 157(1):47–67.

Gade, D., Küçükyavuz, S., and Sen, S. (2014). Decomposition algorithms with parametric gomory cuts for two-stage stochastic integer programs. *Mathematical Programming*, 144(1):39–64.

Gaivoronski, A. A., Lisser, A., Lopez, R., and Xu, H. (2011). Knapsack problem with probability constraints. *Journal of Global Optimization*, 49(3):397–413.

Gao, J. and You, F. (2015). 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.

Gicquel, C., Minoux, M., and Dallery, Y. (2008). Capacitated lot sizing models: a literature review. `https://hal.archives-ouvertes.fr/hal-00255830/document`.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Grass, E., Fischer, K., and Rams, A. (2020). An accelerated l-shaped method for solving two-stage stochastic programs in disaster management. *Annals of Operations Research*, 284(2):557–582.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Heidelberg, Germany: Springer.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2047–2052. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Gu, S., Hao, T., and Yao, H. (2020). A pointer network based deep learning algorithm for unconstrained binary quadratic programming problem. *Neurocomputing*, 390:1–11.

Guastaroba, G. and Speranza, M. G. (2014). A heuristic for BILP problems: the single source capacitated facility location problem. *European Journal of Operational Research*, 238(2):438–450.

Guerriero, F. and Guido, R. (2011). Operational research in the management of the operating theatre: a survey. *Health Care Management Science*, 14(1):89–114.

Guha, N., Wang, Z., Wytock, M., and Majumdar, A. (2019). Machine learning for ac optimal power flow. *arXiv preprint arXiv:1910.08842*.

Gul, S., Denton, B. T., and Fowler, J. W. (2015). A progressive hedging approach for surgery planning under uncertainty. *INFORMS Journal on Computing*, 27(4):755–772.

Guo, P., Huang, G. H., Zhu, H., and Wang, X. (2010). A two-stage programming approach for water resources management under randomness and fuzziness. *Environmental Modelling & Software*, 25(12):1573–1581.

Gupta, A., Müller, A. T., Huisman, B. J., Fuchs, J. A., Schneider, P., and Schneider, G. (2018). Generative recurrent networks for de novo drug design. *Molecular Informatics*, 37(1-2):1700111.

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual, version 9.5. https://www.gurobi.com.

Hao, X., Peng, Z., Ma, Y., Wang, G., Jin, J., Hao, J., Chen, S., Bai, R., Xie, M., Xu, M., Zheng, Z., Yu, C., Li, H., Xu, J., and Gai, K. (2020). Dynamic knapsack optimization towards efficient multi-channel sequential advertising. In *International Conference on Machine Learning*, pages 4060–4070. Cambridge, MA: Proceedings of Machine Learning Research.

Hartman, J. C., Büyüktahtakın, İ. E., and Smith, J. C. (2010). Dynamic-programming-based inequalities for the capacitated lot-sizing problem. *IIE Transactions*, 42(12):915–930.

Haugen, K. K., Løkketangen, A., and Woodruff, D. L. (2001). Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132(1):116–122.

He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

He, P., Zhang, W., Xu, X., and Bian, Y. (2015b). Production lot-sizing and carbon emissions under cap-and-trade and carbon tax regulations. *Journal of Cleaner Production*, 103:241–248.

He, Y., Wu, G., Chen, Y., and Pedrycz, W. (2021). A two-stage framework and reinforcement learning-based optimization algorithms for complex scheduling problems. *arXiv preprint arXiv:2103.05847*.

Helber, S. and Sahling, F. (2010). A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, 123(2):247–256.

Hjelmeland, M. N., Zou, J., Helseth, A., and Ahmed, S. (2018). Nonconvex medium-term hydropower scheduling by stochastic dual dynamic integer programming. *IEEE Transactions on Sustainable Energy*, 10(1):481–490.

Ho, J. K. and Manne, A. S. (1974). Nested decomposition for dynamic models. *Mathematical Programming*, 6(1):121–140.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Homem-de Mello, T. and Pagnoncelli, B. K. (2016). Risk aversion in multistage stochastic programming: A modeling and algorithmic perspective. *European Journal of Operational Research*, 249(1):188–199.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

Hopfield, J. J. and Tank, D. W. (1985). "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152.

Hu, H., Zhang, X., Yan, X., Wang, L., and Xu, Y. (2017). Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*.

Huang, G. and Loucks, D. P. (2000). An inexact two-stage stochastic programming model for water resources management under uncertainty. *Civil Engineering Systems*, 17(2):95–118.

Huang, K. and Ahmed, S. (2009). The value of multistage stochastic programming in capacity planning under uncertainty. *Operations Research*, 57(4):893–904.

Huang, K. and Küçükyavuz, S. (2008). On stochastic lot-sizing problems with random lead times. *Operations Research Letters*, 36(3):303–308.

Huang, Z., Wang, K., Liu, F., Zhen, H.-L., Zhang, W., Yuan, M., Hao, J., Yu, Y., and Wang, J. (2022). Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353.

Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020). Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*.

Hwang, D., Jaillet, P., and Manshadi, V. (2021). Online resource allocation under partially predictable demand. *Operations Research*, 69(3):895–915.

IBM ILOG, CPLEX (2016). CPLEX User's Manual, version 12.7.0. https://www.ibm.com/analytics/cplex-optimizer.

Ishibuchi, H., Akedo, N., and Nojima, Y. (2014). Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems. *IEEE Transactions on Evolutionary Computation*, 19(2):264–283.

Jans, R. and Degraeve, Z. (2008). Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 46(6):1619–1643.

Jiménez-Cordero, A., Morales, J. M., and Pineda, S. (2022). Warm-starting constraint generation for mixed-integer optimization: A machine learning approach. *Knowledge-Based Systems*, 253:109570.

Kaminsky, P. and Simchi-Levi, D. (2003). Production and distribution lot sizing in a two stage supply chain. *IIE Transactions*, 35(11):1065–1075.

Kantas, A. B., Cobuloglu, H. I., and Büyüktahtakın, İ. E. (2015). Multi-source capacitated lot-sizing for economically viable and clean biofuel production. *Journal of Cleaner Production*, 94:116–129.

Karimi, B., Ghomi, S. F., and Wilson, J. (2003). The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk99zCeAb.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017a). Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30:6351–6361.

Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y. (2017b). Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 659–666. https://www.ijcai.org/proceedings/2017/0092.pdf.

Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 724–731. Palo Alto, CA: Association for the Advancement of Artificial Intelligence.

Kıbış, E. Y. and Büyüktahtakın, İ. E. (2017). Optimizing invasive species management: A mixed-integer linear programming approach. *European Journal of Operational Research*, 259(1):308–321.

Kıbış, E. Y. and Büyüktahtakın, İ. E. (2019). Optimizing multi-modal cancer treatment under 3d spatio-temporal tumor growth. *Mathematical Biosciences*, 307:53–69.

Kıbış, E. Y., Büyüktahtakın, İ. E., Haight, R. G., Akhundov, N., Knight, K., and Flower, C. E. (2021). A multistage stochastic programming approach to the optimal surveillance and control of the emerald ash borer in cities. *INFORMS Journal on Computing*, 33(2):808–834.

Kim, K. and Mehrotra, S. (2015). A two-stage stochastic integer programming approach to integrated staffing and scheduling with application to nurse management. *Operations Research*, 63(6):1431–1451.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kong, W., Liaw, C., Mehta, A., and Sivakumar, D. (2018). A new dog learns old tricks: Rl finds classic optimization algorithms. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkluJ2R9KQ.

Kool, W., van Hoof, H., and Welling, M. (2018). Attention, learn to solve routing problems! In *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm.

Kopanos, G. M., Puigjaner, L., and Georgiadis, M. C. (2010). Optimal production scheduling and lot-sizing in dairy plants: the yogurt production line. *Industrial & Engineering Chemistry Research*, 49(2):701–718.

Kosuch, S. and Lisser, A. (2011). On two-stage stochastic knapsack problems. *Discrete Applied Mathematics*, 159(16):1827–1841.

Kotary, J., Fioretto, F., and Van Hentenryck, P. (2021). Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34:24981–24992.

Kruber, M., Lübbecke, M. E., and Parmentier, A. (2017). Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Cham, Switzerland: Springer.

Küçükyavuz, S. and Sen, S. (2017). An introduction to two-stage stochastic mixed-integer programming. In *Leading Developments from INFORMS Communities*, pages 1–27. Catonsville, MD: Institute for Operations Research and the Management Sciences.

Lamba, K., Singh, S. P., and Mishra, N. (2019). Integrated decisions for supplier selection and lot-sizing considering different carbon emission regulations in big data environment. *Computers & Industrial Engineering*, 128:1052–1062.

Laporte, G. and Louveaux, F. V. (1993). The integer l-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142.

Lara, C. L., Siirola, J. D., and Grossmann, I. E. (2020). Electric power infrastructure planning under uncertainty: stochastic dual dynamic integer programming (SDDiP) and parallelization scheme. *Optimization and Engineering*, 21(4):1243–1281.

Larsen, E., Frejinger, E., Gendron, B., and Lodi, A. (2022a). Fast continuous and integer l-shaped heuristics through supervised learning. *arXiv preprint arXiv:2205.00897*.

Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., and Lodi, A. (2022b). Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS Journal on Computing*, 34(1):227–242.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Heidelberg, Germany: Springer.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Li, J., Wang, Y., Lyu, M. R., and King, I. (2017). Code completion with neural attention and pointer networks. *arXiv preprint arXiv:1711.09573*.

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

Linderoth, J. and Wright, S. (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2):207–250.

Lisser, A. and Lopez, R. (2010). Stochastic quadratic knapsack with recourse. *Electronic Notes in Discrete Mathematics*, 36:97–104.

Liu, C., Fan, Y., and Ordóñez, F. (2009). A two-stage stochastic programming model for transportation network protection. *Computers & Operations Research*, 36(5):1582–1590.

Liu, D., Fischetti, M., and Lodi, A. (2022). Learning to search in local branching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3796–3803. Palo Alto, CA: Association for the Advancement of Artificial Intelligence.

Liu, P., Pistikopoulos, E. N., and Li, Z. (2010). Decomposition based stochastic programming approach for polygeneration energy systems design under uncertainty. *Industrial & Engineering Chemistry Research*, 49(7):3295–3305.

Liu, X., Zheng, Z., Büyüktahtakın, İ. E., Zhou, Z., and Wang, P. (2021). Battery asset management with cycle life prognosis. *Reliability Engineering & System Safety*, 216:107948.

Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *TOP*, 25(2):207–236.

Louveaux, F. V. (1980). A solution method for multistage stochastic programs with recourse with application to an energy investment problem. *Operations Research*, 28(4):889–902.

Lu, H., Zhang, X., and Yang, S. (2019). A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJe1334YDH.

Lubin, M., Martin, K., Petra, C. G., and Sandıkçı, B. (2013). On parallelizing dual decomposition in stochastic integer programming. *Operations Research Letters*, 41(3):252–258.

Lulli, G. and Sen, S. (2004). A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Management Science*, 50(6):786–796.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Maes, J. and Wassenhove, L. V. (1988). Multi-item single-level capacitated dynamic lot-sizing heuristics: A general review. *Journal of the Operational Research Society*, 39(11):991–1004.

Maqsood, I. and Huang, G. H. (2003). A two-stage interval-stochastic programming model for waste management under uncertainty. *Journal of the Air & Waste Management Association*, 53(5):540–552.

Marufuzzaman, M., Eksioglu, S. D., and Huang, Y. E. (2014). Two-stage stochastic programming supply chain model for biodiesel production via wastewater treatment. *Computers & Operations Research*, 49:1–17.

Masti, D. and Bemporad, A. (2019). Learning binary warm starts for multiparametric mixed-integer quadratic programming. In *2019 18th European Control Conference (ECC)*, pages 1494–1499. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400.

Merzifonluoglu, Y. and Geunes, J. (2021). The risk-averse static stochastic knapsack problem. *INFORMS Journal on Computing*, 33(3):931–948.

Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. (2014). Recurrent models of visual attention. *Advances in Neural Information Processing Systems*, 27:2204–2212.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Möller, A., Römisch, W., and Weber, K. (2008). Airline network revenue management by multistage stochastic programming. *Computational Management Science*, 5(4):355–377.

Monem, M., Alam, M. G. R., Abdullah-Al-Wadud, M., Huda, S., Hassan, M. M., and Fortino, G. (2022). An industry-4.0-compliant sustainable bitcoin model through optimized transaction selection and sustainable block integration. *IEEE Transactions on Industrial Informatics*, 18(12):9162–9172.

Morita, H., Ishii, H., and Nishida, T. (1989). Stochastic linear knapsack programming problem and its application to a portfolio selection problem. *European Journal of Operational Research*, 40(3):329–336.

Mottini, A. and Acuna-Agost, R. (2017). 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.

Müller, R., Kornblith, S., and Hinton, G. E. (2019). When does label smoothing help? *Advances in Neural Information Processing Systems*, 32:4696–4705.

Mulvey, J. M. and Shetty, B. (2004). Financial planning via multi-stage stochastic optimization. *Computers & Operations Research*, 31(1):1–20.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., Addanki, R., Hapuarachchi, T., Keck, T., Keeling, J., Kohli, P., Ktena, I., Li, Y., Vinyals, O., and Zwols, Y. (2020). Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*.

Nazari, M., Oroojlooy, A., Takáč, M., and Snyder, L. V. (2018). 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.

Nguyen, T. T., Nguyen, N. D., and Nahavandi, S. (2020). Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839.

Onal, S., Akhundov, N., Büyüktahtakın, İ. E., Smith, J., and Houseman, G. R. (2020). An integrated simulation-optimization framework to optimize search and treatment path for controlling a biological invader. *International Journal of Production Economics*, 222:107507.

Oroojlooyjadid, A., Snyder, L. V., and Takáč, M. (2019). Applying deep learning to the newsvendor problem. *IISE Transactions*, 52(4):444–463.

Pan, X., Zhao, T., and Chen, M. (2019). Deepopf: Deep neural network for dc optimal power flow. In *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Pan, Z., Tang, J., and Liu, O. (2009). Capacitated dynamic lot sizing problems in closed-loop supply chain. *European Journal of Operational Research*, 198(3):810–821.

Paulus, A., Rolínek, M., Musil, V., Amos, B., and Martius, G. (2021). Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, pages 8443–8453. Cambridge, MA: Proceedings of Machine Learning Research.

Pochet, Y. and Wolsey, L. A. (2006). *Production planning by mixed integer programming*. New York, NY: Springer.

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics*, 56(3):239–249.

Prékopa, A. (2013). *Stochastic programming*. Dordrecht, Netherlands: Springer Science & Business Media.

Quadt, D. and Kuhn, H. (2005). Conceptual framework for lot-sizing and scheduling of flexible flow lines. *International Journal of Production Research*, 43(11):2291–2308.

Quadt, D. and Kuhn, H. (2007). Capacitated lot-sizing with extensions: a review. *4OR*, 6(1):61–83.

Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.

Rockafellar, R. T. and Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Buffalo, NY: Cornell Aeronautical Laboratory.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35(3):309–333.

Ruszczyński, A. and Shapiro, A. (2003). Stochastic programming models. *Handbooks in Operations Research and Management Science*, 10:1–64.

Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76.

Samavati, M., Essam, D., Nehring, M., and Sarker, R. (2017). A methodology for the large-scale multi-period precedence-constrained knapsack problem: an application in the mining industry. *International Journal of Production Economics*, 193:12–20.

Schmidt, G. and Wilhelm, W. E. (2000). Strategic, tactical and operational decisions in multi-national logistics networks: a review and discussion of modelling issues. *International Journal of Production Research*, 38(7):1501–1523.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

Shen, Y., Sun, Y., Eberhard, A., and Li, X. (2021). Learning primal heuristics for mixed integer programs. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Shen, Y., Sun, Y., Li, X., Eberhard, A., and Ernst, A. (2022). Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9926–9934. Palo Alto, CA: Association for the Advancement of Artificial Intelligence.

Shrouf, F. and Miragliotta, G. (2015). Energy management based on internet of things: practices and framework for adoption in production management. *Journal of Cleaner Production*, 100:235–246.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. Cambridge, MA: Proceedings of Machine Learning Research.

Singh, K. J., Philpott, A. B., and Wood, R. K. (2009). Dantzig-Wolfe decomposition for solving multistage stochastic capacity-planning problems. *Operations Research*, 57(5):1271–1286.

Skar, C., Doorman, G., and Tomasgard, A. (2014). Large-scale power system planning using enhanced benders decomposition. In *2014 Power Systems Computation Conference*, pages 1–7. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Smith, K. A. (1999). Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34.

Solveling, G., Solak, S., Clarke, J.-P., and Johnson, E. (2011). Runway operations optimization in the presence of uncertainties. *Journal of Guidance, Control, and Dynamics*, 34(5):1373–1382.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stahlberg, F. (2020). Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27:3104–3112.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge, MA: MIT press.

Tajeddini, M. A., Rahimi-Kian, A., and Soroudi, A. (2014). Risk averse optimal operation of a virtual power plant using two stage stochastic programming. *Energy*, 73:958–967.

Tang, D., Qin, B., and Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432. Red Hook, NY: Curran Associates Inc.

Tang, Y., Agrawal, S., and Faenza, Y. (2020). Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. Cambridge, MA: Proceedings of Machine Learning Research.

Tempelmeier, H. (2013). Stochastic lot sizing problems. In *Handbook of Stochastic Models and Analysis of Manufacturing System Operations*, pages 313–344. New York, NY: Springer.

Thevenin, S., Adulyasak, Y., and Cordeau, J.-F. (2022). Stochastic dual dynamic programming for multiechelon lot sizing with component substitution. *INFORMS Journal on Computing*.

Toledo, C. F. M., da Silva Arantes, M., Hossomi, M. Y. B., França, P. M., and Akartunalı, K. (2015). A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *Journal of Heuristics*, 21(5):687–717.

Uzsoy, R., Lee, C.-Y., and Martin-Vega, L. A. (1992). A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning. *IIE Transactions*, 24(4):47–60.

Van Slyke, R. M. and Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663.

Varnamkhasti, M. J. (2012). Overview of the algorithms for solving the multidimensional knapsack problems. *Advanced Studies in Biology*, 4(1):37–47.

Veliz, F. B., Watson, J.-P., Weintraub, A., Wets, R. J.-B., and Woodruff, D. L. (2015). Stochastic optimization models in forest planning: a progressive hedging solution approach. *Annals of Operations Research*, 232(1):259–274.

Vespucci, M. T., Maggioni, F., Bertocchi, M. I., and Innorta, M. (2012). A stochastic model for the daily coordination of pumped storage hydro plants and wind power plants. *Annals of Operations Research*, 193(1):91–105.

Vinyals, O., Bengio, S., and Kudlur, M. (2015a). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015b). Pointer networks. *Advances in Neural Information Processing Systems*, 28:2692–2700.

Watson, J.-P. and Woodruff, D. L. (2011). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370.

Wichmann, M. G., Johannes, C., and Spengler, T. S. (2019). Energy-oriented lot-sizing and scheduling considering energy storages. *International Journal of Production Economics*, 216:204–214.

Wilbaut, C., Hanafi, S., and Salhi, S. (2008). A survey of effective heuristics and their application to a variety of knapsack problems. *IMA Journal of Management Mathematics*, 19(3):227–244.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

Wu, F. and Sioshansi, R. (2017). 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.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Wu, Y., Song, W., Cao, Z., and Zhang, J. (2021). Learning scenario representation for solving two-stage stochastic integer programs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=06Wy2BtxXrz.

Xavier, Á. S., Qiu, F., and Ahmed, S. (2021). Learning to solve large-scale security-constrained unit commitment problems. *INFORMS Journal on Computing*, 33(2):739–756.

Xiao, J., Yang, H., Zhang, C., Zheng, L., and Gupta, J. N. (2015). A hybrid lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 63:72–82.

Yen, J. W. and Birge, J. R. (2006). A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40(1):3–14.

Yilmaz, D. and Büyüktahtakın, İ. E. (2022a). An expandable learning-optimization framework for sequentially dependent decision-making. *Submitted to European Journal of Operational Research*.

Yilmaz, D. and Büyüktahtakın, İ. E. (2022b). Learning optimal solutions via an LSTM-optimization framework. *Submitted to Operations Research Forum*.

Yin, X. and Büyüktahtakın, İ. E. (2021). A multi-stage stochastic programming approach to epidemic resource allocation with equity considerations. *Health Care Management Science*, 24(3):597–622.

Yin, X. and Büyüktahtakın, İ. E. (2022). Risk-averse multi-stage stochastic programming to optimizing vaccine allocation and treatment logistics for effective epidemic response. *IISE Transactions on Healthcare Systems Engineering*, 12(1):52–74.

Yin, X., Büyüktahtakın, İ. E., and Patel, B. P. (2023). Covid-19: Data-driven optimal allocation of ventilator supply under uncertainty and risk. *European Journal of Operational Research*, 304(1):255–275.

Yu, X. and Shen, S. (2020). Multistage distributionally robust mixed-integer programming with decision-dependent moment-based ambiguity sets. *Mathematical Programming*.

Zakaria, A., Ismail, F. B., Lipu, M. H., and Hannan, M. A. (2020). Uncertainty models for stochastic optimization in renewable energy applications. *Renewable Energy*, 145:1543–1571.

Zamzam, A. S. and Baker, K. (2020). Learning optimal solutions for extremely fast ac optimal power flow. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Zha, D., Lai, K.-H., Zhou, K., and Hu, X. (2019). Experience replay optimization. *arXiv preprint arXiv:1906.08387*.

Zheng, J., Wang, L., Wang, S., Liang, Y., and Pan, J. (2021). Solving two-stage stochastic route-planning problem in milliseconds via end-to-end deep learning. *Complex & Intelligent Systems*, 7(3):1207–1222.

Zhou, Z., Li, X., and Zare, R. N. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science*, 3(12):1337–1344.

Zou, J., Ahmed, S., and Sun, X. A. (2019). Stochastic dual dynamic integer programming. *Mathematical Programming*, 175(1):461–502.