POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Decision Making Under Uncertainty Using Machine Learning

RAHUL PATEL
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Mathématiques appliquées

Août 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Decision Making Under Uncertainty Using Machine Learning

présenté par **Rahul PATEL**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Michel GENDREAU, président
Andrea LODI, membre et directeur de recherche
Yoshua BENGIO, membre et codirecteur de recherche
Emma FREJINGER, membre

DEDICATION

To my grandparents and parents.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my co-advisors, Andrea Lodi and Yoshua Bengio, for providing me the freedom to explore a fundamental research problem during the course of this thesis. It was because of them I got to work in such an exciting and upcoming research area of combinatorial machine learning. I appreciate Andrea's words of encouragement and support to the members of the chair during these turbulent times of pandemic. I would also like to thank my brilliant collaborators, Sriram Sankaranarayanan and Emma Frejinger, from whom I learned a lot. I am grateful to Sriram for chalking out the exact research idea which is at the heart of this thesis.

I would also like to thank Mehdi Taobane for gracefully managing the logistical aspects during my masters; right from scheduling my interview with Andrea to scholarship and organizing my defense. I would like to thank Khalid for providing the necessary technical support critical for conducting the experiments. I would like to thank all of my colleagues at the chair, especially, Jeff Decary, Didier Chételat, Antoine Prouvost, Elias Khalil and Justin Dumouchelle, for the wonderful discussions on varied topics like useful pointers for my research to exploring things in Montreal.

I am exceptionally grateful to my friends, particularly, Shashwat Sanghavi, Nisarg Patwa, Harsh Prajapati, Dhanraj Vaghela, Eeshan Dhekane and Sujaat Ali for their positivity and support during the course of past two years. I would like to thank all the teachers, mentors and educational institutions I was a part of, right from kindergarten to this date, for shaping my intellect and imbibing in me the quality to ask the right questions.

I will always be in debt of the love and support of my parents. They have motivated me to pursue my dreams and taught me to take responsibility for my actions. I am ever so grateful to my fiancé, who has been with me through the thick and thin since the past 10 years, who knows me better than I do myself. Last but not the least, I would like to thank God for creating this beautiful universe and all its amazing life forms.

RÉSUMÉ

Nous proposons un algorithme basé sur l'apprentissage supervisé pour obtenir de bonnes solutions primales pour les programmes stochastiques en deux étapes en nombres entiers (en anglais, two-stage stochastic integer programs (2SIP)). Le but de l'algorithme est de prédire un scénario représentatif (en anglais, representative scenario (RS)) pour le problème tel qu'en résolvant de manière déterministe le 2SIP avec la réalisation aléatoire égale au scénario représentatif, l'algorithme donne une solution quasi optimale au 2SIP original. Prédire un RS, au lieu de prédire directement une solution, garantit la faisabilité de la solution de première étape. Si le problème possède un recours complet, la réalisabilité de la deuxième étape est également garantie.

Nous effectuons des expériences sur deux problèmes: le problème de localisation d'entrepôts avec capacité stochastique (en anglais, stochastic capacitated facility location problem (S-CFLP)) et problème d'affectation généralisée stochastique (en anglais, stochastic generalized assignment problem (S-GAP)). Les deux problèmes ont des variables entières et des contraintes linéaires dans les première et deuxième étapes. La méthode proposée est capable de produire de bonnes solutions primales pour le S-CFLP lorsqu'elle est testée sur les tailles sur lesquelles elle a été entraînée. De plus, notre temps de calcul est compétitif par rapport à celui pris par Gurobi pour obtenir une qualité de solution similaire. Cependant, nos modèles ne sont pas capables de généraliser et de produire de bonnes solutions primales lorsqu'ils sont testés sur les tailles sur lesquelles ils n'ont pas été entraînés. Dans le cas de S-GAP, jusqu'à maintenant, notre méthode peine à trouver de bonnes solutions primales. Nous discutons des défis et des solutions potentielles que nous pourrions utiliser pour leur faire face.

ABSTRACT

We propose a supervised learning based algorithm to obtain good primal solutions for two-stage stochastic integer programming (2SIP) problems with constraints in the first and second stages. The goal of the algorithm is to predict a representative scenario for the problem such that, deterministically solving a two-stage stochastic integer program with the random realization equal to a representative scenario, gives a near-optimal solution to the original 2SIP. Predicting a representative scenario, instead of directly predicting a solution ensures first-stage feasibility of the solution. If the problem is known to have complete recourse, second-stage feasibility is also guaranteed.

We perform computational tests on two problems, namely, the stochastic capacitated facility location problem (S-CFLP) and stochastic generalized assignment problem (S-GAP). Both the problems have integer variables and linear constraints in the first and second stages. The proposed method is able to produce good primal solutions for the S-CFLP when tested on the sizes on which it was trained. Also, our computing time is competitive to that taken by Gurobi to achieve a similar solution quality. However, our models are not able to generalize and produce good primal solutions when tested on the sizes on which they were not trained. In the case of S-GAP, as of now, our method struggles to find good primal solutions. We discuss the challenges and the potential solutions we would be pursuing to alleviate them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

LP	linear program
MILP	mixed integer linear programming
2SIP	two-stage stochastic integer program
RS	representative scenario
OVDR	Objective Value Difference Ratio
NOS	number of scenarios
S-CFLP	stochastic capacitated facility location problem
S-GAP	stochastic generalized assignment problem
AI	Artificial Intelligence
ML	Machine Learning
i.i.d.	independent and identically distributed
RL	Reinforcement Learning
LR	Linear Regression
FFNN	Feed-forward Neural Network
CNN	Convolutional Neural Network
MSE	mean squared error
R^2	R-squared score
SGD	stochastic gradient descent

CHAPTER 1 INTRODUCTION

A key trait of humans is rationality; this involves taking conscious decisions to minimize some cost. Due to the limited availability of resources (fossil, time, bandwidth, etc.) in the world, more often than not, the decisions that we are allowed to make need to respect certain constraints. Mathematical optimization – more popularly known as mathematical programming – is one such branch of mathematics that formalizes the art of decision making by providing techniques to model and solve decision-making problems. This problem of optimal decision making becomes all the more challenging and complex when the constraints under which one needs to operate are not known in advance. Speaking broadly the area that deals with such problems is called *decision making under uncertainty*. It includes areas such as stochastic programming, stochastic control, bandit problems, active learning, etc. [1]. In this thesis, we focus on stochastic integer programming, which lies at the intersection of mathematical programming and decision making under uncertainty.

The preliminaries, crucial to the work presented in the thesis, are covered in Section 1.1 and Section 1.2.

1.1 Mathematical Programming

The first step to solve a decision-making problem is its mathematical modeling. The key components used for defining the model are as follows:

- Problem data or problem input
- Set of variables over which one wants to optimize
- Set of constraints on variables
- Objective function to be optimized

Supposing that the objective function models cost, we would like to minimize it. Consider a problem P of the form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) && && (1.1)\\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0, && i = 1, 2, \ldots, k && (1.2)\\ & h_j(\mathbf{x}) = 0, && j = 1, 2, \ldots, l && (1.3)\\ & \mathbf{x} \in X, && && (1.4) \end{aligned}$$

where s.t. is a shorthand for subject to. Here, $\mathbf{x} \in \mathbb{R}^n$ is the decision variable vector, $f : \mathbb{R}^n \to \mathbb{R}$ is the objective value function of $\mathbf{x}$ to be minimized, $\mathbf{g} = \{g_1, \ldots, g_k\}$ and $\mathbf{h} = \{h_1, \ldots, h_l\}$ are the set of functions used to define constraints on $\mathbf{x}$ and $X \subseteq \mathbb{R}^n$. The exact definition of $f, \mathbf{g}, \mathbf{h}$ and X depends on the problem definition. The type of optimization problem and their solution techniques are characterized by $f, \mathbf{g}, \mathbf{h}$ and X.

1.1.1 Linear Programming

A mathematical programming problem can be characterized as a linear program (LP), when the objective value function and the constraints are linear. An LP in the standard form can be written as

$$\begin{aligned} \min \quad & \mathbf{c}^\intercal \mathbf{x} && (1.5)\\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} && (1.6)\\ & \mathbf{x} \geq \mathbf{0} && (1.7) \end{aligned}$$

where $\mathbf{c}$ and $\mathbf{x}$ are n-dimensional column vectors, $\mathbf{b}$ is an m-dimensional column vector and A is an $m \times n$ matrix.

George Dantzig, known as the father of LP, used it to solve several military planning problems. One of the key contributions of Dantzig was the Simplex method, which can efficiently solve an LP with a polynomial average-case complexity with respect to the size of the problem, [2–4]. Suppose that the constraint polyhedron is feasible, bounded and non-degenerate. First, the simplex method initializes the solution to one of the vertices of the constraint polyhedron and then improves it iteratively by moving from one vertex to another, such that the objective value is improved. There are theoretically better algorithms to solve an LP with a polynomial worst-case time complexity (ellipsoid and interior-point methods [5]). However, the choice of the algorithm to solve an LP should be made depending upon the context of the problem.

1.1.2 Mixed Integer Linear Programming

Many decision problems in practice involve that decisions are made from a discrete set, which can be modeled by constraining a subset of variables to take integer values. The mixed integer linear programming (MILP) paradigm offers us the tools to model and solve such problems. A general MILP formulation is given as

$$\begin{aligned} \min \quad & \mathbf{c}^\intercal \mathbf{x} && (1.8)\\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} && (1.9)\\ & \mathbf{x} \geq \mathbf{0} && (1.10)\\ & \mathbf{x}_i \in \mathbb{Z} \qquad \forall i \in \mathcal{I} && (1.11) \end{aligned}$$

where $\mathcal{I}$ is the set of indices of variables $\mathbf{x}$ which are required to be integer and $\mathbb{Z}$ is the set of integers. As compared to an LP formulation, we imposed additional constraints given by Equation (1.11) which models integer decisions. However, integrality constraints makes the problem non-convex, leading to MILP falling in the NP-hard complexity class [5].

We know that LP belongs to the P complexity class; there exist polynomial time algorithms to solve any given LP. We leverage this fact and advances in solving LP to iteratively solve a MILP. The basic idea is to start with an LP relaxation of the original MILP and iteratively add more constraints to this LP to prune the LP solution space, such that the eventual LP solution obtained satisfies the integrality constraints of the original MILP. This idea is at the core of many algorithms for solving MILPs like branch-and-bound [6], cutting planes [7], branch-and-cut [8], branch-and-price, etc. We will briefly describe branch-and-bound and cutting plane approaches to solve a MILP, which are the building blocks to other advanced approaches to MILP solving.

Branch-and-bound

Branch-and-bound is a tree-based approach that iteratively prunes and creates smaller partitions of the solution space using relaxation bounds and creating sub-MILP problems adding cuts, respectively. Details of the same, for a maximization problem, are given in Algorithm

1 based on [9].

Algorithm 1: The branch-and-bound algorithm to solve a maximization MILP

Data: MILP instance I

Result: Optimal integer solution to $I, \mathbf{x}^*_{IP}$

Let I_{LP} be the LP relaxation of instance I;
Let the best lower bound $BLB \leftarrow -\infty$;
Let the set of open nodes $\mathcal{O} \leftarrow \{I_{LP}\}$;
Initialize $\mathbf{x}^*_{IP} \leftarrow \emptyset$;
while $|\mathcal{O}| > 0$ **do**
    Select a node I^s_{LP} from the set of open nodes $\mathcal{O}$ and delete it from $\mathcal{O}$;
    Solve I^s_{LP};
    if I^s_{LP} *is feasible* **then**
        Let OBJ^s_{LP} and $\mathbf{x}^s_{LP}$ be the objective value and solution of I^s_{LP}, respectively;
        if $OBJ^s_{LP} > BLB$ **then**
            if $\mathbf{x}^s_{LP}$ *satisfies integrality constraints of* I **then**
                $BLB \leftarrow OBJ^s_{LP}$;
                $\mathbf{x}^*_{IP} \leftarrow \mathbf{x}^s_{LP}$;
            else
                Create two subproblems I^{s1}_{LP} and I^{s2}_{LP} by adding constraints such that none of them contain $\mathbf{x}^s_{LP}$ in their feasible region. However, both of their feasible regions should contain all the feasible integer solutions of I^s_{LP};
                $\mathcal{O} \leftarrow \mathcal{O} \cup \{I^{s1}_{LP}\} \cup \{I^{s2}_{LP}\}$;
            end
        end
    end
end
return $\mathbf{x}^*_{IP}$;

The branch-and-bound solves an LP relaxation of the MILP problem on each node of the tree. Once solved, it checks first if the bound obtained is better than the best bound so far. If no, then update the best bound and create two sub-problems by branching on a variable which is supposed to be an integer, but takes fraction value in the relaxed solution.

Cutting planes

The general idea of algorithms falling in the broad category of cutting planes to solve a MILP starts by optimizing over the polyhedron defined by the LP relaxation and iteratively adds constraints such that the LP optimal solution, containing fractional solution is cut-off, without removing any feasible solution from the original non-convex constraint set of MILP. The present-day cutting plane algorithms are used as sub-routine within the branch-and-bound framework to make it a branch-and-cut. Next, we illustrate through a toy example of how a cutting plane method might function at an iteration. Let $x \in \mathbb{Z}^2$ be an integer decision variable and let $c \in \mathbb{R}$ be a positive constant. Suppose, we need to obey $x_1 + x_2 \leq c$, then we can use the rounding argument to claim that $x_1 + x_2 \leq \lfloor c \rfloor$. Note that all the feasible solutions for $x \in \mathbb{Z}$ are still valid, however, we are successful in chopping of the search space.

1.2 Machine Learning

The human brain is a marvelous computing machine with about 86 billion neurons [10]. As humans, we would like to emulate this complex computing system using a mix of modern-day hardware and software to solve problems of humanity, and in the process get a better understanding of the governing principles of our intellect. This field of study is broadly known as Artificial Intelligence (AI). Machine Learning (ML) is a sub-field of AI, that leverages data of the past experiences and optimization, rather than explicitly stated rules, to assist decision making in the future in similar situations.

ML can be broadly classified into three main categories of supervised learning, unsupervised learning and reinforcement learning. In supervised learning, we are given a training set containing input-output pairs. The task is to find a mapping from the input space to the output space, such that this mapping is able to correctly predict the output for the unseen inputs. In unsupervised learning, the task is to find the latent representation of the data points. This latent representation can then be used for tasks like grouping similar data points, density estimation, etc. One of the successful applications of this technique can be found in the field of natural language processing for improving language understanding tasks [11]. In reinforcement learning, an agent learns the optimal control mechanism for some task by continuously interacting with the environment and improving the expected reward it gathers during an episode. The agent-environment interaction is modeled as a Markov decision process. The agent starts in some initial state and performs some action. The environment processes this action to return a reward signal and the next state to the agent. This interaction continues until the agent reaches a terminal state and the episode ends. By

interacting with the environment over multiple episodes, the agent learns how good it is to be in some state and what is the best course of action in this state.

1.2.1 Supervised learning

Given a labeled dataset of input-output pairs, supervised learning tries to learn a mapping that can correctly predict an output associated with an input. We later use this mapping to predict output for a previously unseen input. These unseen input-output pairs on which we test are also sampled from the same distribution as the input-output pairs on which it was trained. Supervised learning can be divided into two main classes of problems, namely, classification and regression.

In (binary) classification, the output $\mathbf{y} \in \{0, 1\}$, indicating a discrete class to which the input $\mathbf{x}$ belongs. For example, consider a labeled dataset of images containing oranges and apples. As we know, apples and oranges come in many different forms, colors and sizes. Hence, writing an exhaustive if-then-else program to classify an image as an apple or orange is out of the question. Since we have a labeled dataset, we can use supervised learning to train a model that can correctly classify whether the input image is an apple or orange. The image of the fruit can be treated as an input $\mathbf{x}$ and the corresponding label $\mathbf{y}$ can either be 0 for being an apple or 1 for being orange. Thus, whenever a new unlabeled image is given to the model, it will predict the conditional probability $p(\mathbf{y} = 1|\mathbf{x})$. We can predict the class to which the input belongs by thresholding this probability, i.e., we label it 1 if $p(\mathbf{y} = 1|\mathbf{x}) \geq 0.5$ or 0 otherwise. Note that a given type of fruit might have different properties based on the geographical location where it is produced. Hence, one should not expect the learning algorithm trained on the images of the fruits of particular geography to work on those of another geography, even for the same kind of fruit. This phenomenon is known as the distribution shift. To alleviate this issue, one must strive to make the train set to capture the variations that the model is supposed to encounter during inference for good results. Hence, the system requirements should be clear before endeavoring into the development of such a system.

In regression, the output label $\mathbf{y} \in \mathbb{R}$, mapping the input $\mathbf{x}$ to a real number. For example, consider building a house price prediction system. For simplicity assume that the input $\mathbf{x}$ only contains the information about the number of rooms, number of bathrooms, postal code and area of house in square meters. The corresponding $\mathbf{y}$ label can be the price of that house. We can then use a regression model trained on the data of many such houses.

To put it formally, let $\mathcal{X}$ and $\mathcal{Y}$ be the feature/input space and label/output space, respectively, $P(\mathcal{X}, \mathcal{Y})$ (or P for short) be the non-tractable joint probability distribution defined

over them and $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ a parameterized mapping. The expected discrepancy/loss/cost associated with mapping $f_{\boldsymbol{\theta}}$,

$$L^{emp}(f_{\boldsymbol{\theta}}, \mathcal{X} \times \mathcal{Y}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}\left[L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\right]$$

is called the *empirical risk*. The goal of supervised learning is to find the optimal mapping $f_{\boldsymbol{\theta}^*} \in \mathcal{F}$, where $\mathcal{F}$ is the family of parameterized mappings from $\mathcal{X}$ to $\mathcal{Y}$, that minimizes the empirical risk associated with it, i.e.,

$$f_{\boldsymbol{\theta}^*} = \arg\min_{f_{\boldsymbol{\theta}} \in \mathcal{F}} L^{emp}(f_{\boldsymbol{\theta}}, \mathcal{X} \times \mathcal{Y}).$$

This process of finding $f_{\boldsymbol{\theta}^*}$ from $\mathcal{F}$ is called the *empirical risk minimization*. Note that the exact evaluation of $\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}\left[L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\right]$ is not possible because P is non-tractable, i.e., we do not have access to $\mathcal{X} \times \mathcal{Y}$. Hence, we do the Monte-Carlo approximation to evaluate the same; we create a training dataset $\mathcal{D}^n_{train} = \{(\mathbf{x}^i, \mathbf{y}^i) : (\mathbf{x}^i, \mathbf{y}^i) \sim P,\ 1 \leq i \leq n\}$ by drawing n independent and identically distributed (i.i.d.) samples from P and calculating,

$$L^{emp}(f_{\boldsymbol{\theta}}, \mathcal{D}^n_{train}) = \frac{1}{|\mathcal{D}^n_{train}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}^n_{train}} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) \approx L^{emp}(f_{\boldsymbol{\theta}}, \mathcal{X} \times \mathcal{Y}).$$

The empirical risk can also be viewed from the perspective of the parameters $\boldsymbol{\theta}$, as $f_{\boldsymbol{\theta}}$ is characterized by $\boldsymbol{\theta}$, i.e.,

$$L^{emp}(\boldsymbol{\theta}, \mathcal{D}^n_{train}) \equiv L^{emp}(f_{\boldsymbol{\theta}}, \mathcal{D}^n_{train})$$

and the corresponding optimization process is to find $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}$, instead of $f_{\boldsymbol{\theta}^*} \in \mathcal{F}$. The popular gradient descent based methods like [12–16] are among the preferred choice of the optimization algorithm to find $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}$ as they perform well under practical considerations like computational complexity and time.

Once we have obtained these optimal set of parameters $\boldsymbol{\theta}^*$ for the training data distribution, we check how well do they perform over other samples drawn from P, not present in the training set. As we already know, checking this in its true sense is not feasible as P is non-tractable. Hence, we again do the Monte-Carlo approximation but on a new test dataset; we create a test dataset $\mathcal{D}^m_{test} = \{(\mathbf{x}^i, \mathbf{y}^i) : (\mathbf{x}^i, \mathbf{y}^i) \sim P, (\mathbf{x}^i, \mathbf{y}^i) \notin \mathcal{D}^n_{train},\ 1 \leq i \leq m\}$ by drawing m i.i.d. samples from P, not present in training dataset, and calculating,

$$L^{emp}(\boldsymbol{\theta}^*, \mathcal{D}^m_{test}) = \frac{1}{|\mathcal{D}^m_{test}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}^m_{test}} L(f_{\boldsymbol{\theta}^*}(\mathbf{x}), \mathbf{y}).$$

$L^{emp}(f_{\boldsymbol{\theta}^*}, \mathcal{D}^m_{test})$ is called the *generalization/test error*. To judge the performance of $\boldsymbol{\theta}^*$ and drive the model building process in the right direction, we can leverage the relationship between the train and test errors. Both the train and test errors being low is the ideal scenario. The case in which the train error is high and the test error is low usually does not occur in practice, provided the train and test datasets are created using i.i.d. samples. If we have a high train and test errors, our model is under-fitting the data as it does not have enough capacity. To overcome this, we should increase the model capacity by increasing the number of parameters. If we have a low train error and high test error, our model is over-fitting the data as it has more capacity than required. The high capacity enables it to memorize the training data, resulting in low train error. When this model is subjected to previously unseen data, drawn from the same distribution as training data, it performs miserably. We can take different measures to prevent the model from memorization and driving it to learn a meaningful mapping. First, we can decrease the model capacity by reducing the number of parameters. Second, we can add a regularization term or weight penalty to the empirical loss term during optimization,

$$L^{emp}(f_{\boldsymbol{\theta}^*}, \mathcal{D}^n_{train}) + \lambda R(\boldsymbol{\theta})$$

where $R : \boldsymbol{\Theta} \to \mathbb{R}$ is the regularizer and $\lambda > 0$ is the regularization constant. The two commonly used regularizers are lasso penalty $\mathcal{L}^1(\boldsymbol{\theta}) = \sum_i |\boldsymbol{\theta}^i|$ and ridge penalty $\mathcal{L}^2(\boldsymbol{\theta}) = \sum_i \left\|\boldsymbol{\theta}^i\right\|^2$. Last, we can add more samples in the training dataset. However, this traditional understanding might only be one half of the story as argued in [17], where they continue training beyond the over-fitting regime and achieve improved generalization.

Gradient descent based methods, as stated before, are the workhorse of many modern-day optimization algorithms that help us find the optimal parameters $\boldsymbol{\theta}^*$. Hence, if we evaluate the gradient of $L^{emp}(f_{\boldsymbol{\theta}}, \cdot)$ with respect to $\boldsymbol{\theta}$ and a move a small step α against that direction

$$\boldsymbol{\theta}^{new} \leftarrow \boldsymbol{\theta}^{old} - \alpha \nabla_{\boldsymbol{\theta}} L^{emp}(f_{\boldsymbol{\theta}^{old}}, \cdot)$$

then, $L^{emp}(f_{\boldsymbol{\theta}^{new}}, \cdot) \leq L^{emp}(f_{\boldsymbol{\theta}^{old}}, \cdot)$. This requires us to have a differentiable loss function L. The common choices for L are the cross-entropy loss for classification tasks and the mean squared error for regression tasks. Ideally, we would like to be as close as possible to $\boldsymbol{\theta}^*$ by iteratively updating $\boldsymbol{\theta}$. A nice overview comparing different gradient descent based algorithms is provided in [18]. Next we explain linear regression in Section 1.2.2 and artificial neural network in Section 1.2.3, which are used to solve a regression task in this thesis.

1.2.2 Linear Regression

Linear regression (LR) model, as the name suggests, tries to learn a linear mapping/hypothesis from the input space to output space, i.e., $f_{\boldsymbol{\theta}} : \mathcal{X} \rightarrow \mathcal{Y}$ is linear. Let $\mathbf{X} = [\mathbf{x}^i]_{i=1}^n$ be the inputs and $\mathbf{Y} = [\mathbf{y}^i]_{i=1}^n$ be the corresponding outputs, then the output predictions $\hat{\mathbf{Y}} = \boldsymbol{\theta} \cdot \mathbf{X}$. Note that $\mathbf{x}_0^i = 1, \forall i$ and hence $\boldsymbol{\theta}_0$ acts a bias. This helps in keeping the notation simple. The error $\mathbf{E}$ in the predictions is equal to $\mathbf{E} = \hat{\mathbf{Y}} - \mathbf{Y}$ and we would like to minimize the mean squared error mean squared error (MSE), i.e.,

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum \mathbf{E}^2 = \min_{\boldsymbol{\theta}} \frac{1}{n} \left\| \hat{\mathbf{Y}} - \mathbf{Y} \right\|^2 = \min_{\boldsymbol{\theta}} \frac{1}{n} \left\| \boldsymbol{\theta} \cdot \mathbf{X} - \mathbf{Y} \right\|^2$$

The value of optimal parameters $\boldsymbol{\theta}^*$ is then given by

$$\boldsymbol{\theta}^* = (\mathbf{X^T X})^{-1} \mathbf{X^T Y}.$$

However, for stability and keeping the weights from assuming extreme values (too big or small), it is better to add a lasso or ridge regularizer to the MSE. Note that we can also minimize the $l1$-norm of $\mathbf{E}$, i.e., the absolute value $|\mathbf{E}|$.

1.2.3 Artificial Neural Networks

An artificial neural network (ANN) is a computational model based on an analogy with the *biological* neural network and are composed of interconnected neurons. A neuron is the basic computational unit of an ANN, which takes a linear combination of its input and applies a non-linearity to produce the output, i.e.,

$$\mathbf{y} = \sigma(\boldsymbol{\theta} \cdot \mathbf{x})$$

where $\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}$ and σ are the input, output, neuron parameters and non-linearity respectively. A few of the common non-linearities include sigmoid, tanh, ReLU, leaky-ReLU, etc. An ANN consists of multiple layers of neurons. The predictions of the ANN are compared against the labels or ground truth and the error is computed. This error is used to tune the parameters of the ANN (or the neurons) using backpropagation [19, 20], which uses the chain rule of calculus.

Feed-forward Neural Network

FFNN is a type of an ANN in which each neuron in a given layer is connected to all the neurons in the next layer. The first, intermediate and last layers are called input, hidden and output layers, respectively. Suppose the output at a layer i is $\mathbf{a}^{(i)}$, then we compute the output of the next layer $\mathbf{a}^{(i+1)}$ as

$$\mathbf{a}^{(i+1)} = \sigma(\mathbf{W}^{(i)} \cdot \mathbf{a}^{(i)})$$

where $\mathbf{W}^{(i)}$ is the parameter matrix connecting layer i to $i+1$. $\mathbf{a}^{(0)}$ is set equal to the input. The final output is equal to $\mathbf{a}^{(l)}$ for an $l-$layer neural network. The computations performed to calculate the final layer output $\mathbf{a}^{(l)}$ starting from $\mathbf{a}^{(0)}$, is called the forward-pass. We calculate the loss between the label/ground truth $\mathbf{y}$ and prediction $\mathbf{a}^{(l)}$. Once we have the error, we calculate its gradient with respect to the model parameters sequentially, starting from the last layer to the first layer.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a special type of an ANN, designed specifically to cater to image data. First, image data is high dimensional, and if we try to process it using a FFNN, then the number of parameters can easily explode. For example, a grayscale image of size 100x100 would lead to 10,000 dimensional input. CNN, on the other hand, exploits local connectivity, i.e., they only operate on a small chunk of the data at a given point in time. Dealing with only a small portion of the input helps in reducing the number of parameters as compared to the FFNN. Second, images contain spatial data, however, FFNN are not designed to leverage this. On the other hand, CNN uses parameter sharing which helps to extract exactly similar features on the entire input. Finally, images contain information that is usually invariant to certain translations. Consider performing the classification task on images of apples. These apples might be captured under different lighting, rotation, scaling, etc. However, all of these images contain the same information that an apple is present in that image. CNN uses a technique called max-pooling, which helps it achieve translation invariance. We refer the reader to [21, 22] for further understanding of the CNN.

1.3 Outline

This thesis is primarily based on our work "A learning based algorithm to quickly compute good primal solutions for Stochastic Integer Programs", which appeared at the conference

Constraint Programming, Artificial Intelligence, and Operations Research 2020 as a short paper [23]. Especially, the introduction, motivation, problem definition and methodology given in Section 2.1, Section 2.2, Section 2.3 and Section 2.4, respectively are taken from [23]. Also, [23] provides the preliminary results on the stochastic capacitated facility location problem (S-CFLP) of a fixed size. Section 2.5 details the computational study to further test the scalablity of our approach to handle variable sized instances of the S-CFLP. We also report the performance of our method on the stochastic generalized assignment problem (S-GAP). Finally, Chapter 3 details the conclusion, the challenges faced during the experimentation and the potential next steps to alleviate these challenges.

CHAPTER 2 A LEARNING BASED ALGORITHM TO QUICKLY COMPUTE GOOD PRIMAL SOLUTIONS FOR STOCHASTIC INTEGER PROGRAMS – AN EXTENSION

2.1 Introduction

Two-stage stochastic integer programming (2SIP) is a standard framework to model decision making under uncertainty. In this framework, first the so-called *first-stage* decisions are made. Then, the values of uncertain parameters in the problem are determined, as if sampled from a known distribution. Finally, the second set of decisions is made depending upon the realized value of the uncertain parameters, the so-called *second-stage* or *recourse* of the problem. The decision maker, in this setting, minimizes the sum of (i) a linear function of the first-stage decision variables and (ii) the expected value of the second-stage optimization problem.

2SIP is studied extensively in the literature [24–34] owing to its applicability in various decision making situations with uncertainty, like the stochastic unit-commitment problems for electricity generation [33, 34], stochastic facility location problems [26], stochastic supply chain network design [35], among others. With the overwhelming importance of 2SIP a wide range of solution algorithms have been proposed, for example, [25, 36–40].

2.2 Motivation and Literature Review

This body of work broadly falls in the category of combinatorial ML (CML), where ML is leveraged to solve problems in combinatorial/discrete optimization. AlexNet [41] winning the 2012 ImageNet competition was a monumental moment for the ML community, as researchers from the other scientific communities started believing in the promise of ML. Since then the field has made remarkable progress in areas like computer vision, natural language processing and reinforcement learning owing to a mix of novel algorithms and increased computing capabilities. Thus, researchers from different domains started asking the pertinent question of how can they can use ML to advance the state-of-the-art in their fields.

The field of discrete optimization has many *computationally hard* problems, for example, MILP is NP-hard. As a result, many state-of-the-art algorithms in this domain heavily rely on heuristics which are based on the knowledge accumulated over the years by the experts and can be problem specific. There can be better techniques to solve these problems, which are not yet discovered. Reinforcement learning, a specific type of ML, can be used in such cases, to discover novel control strategies, just like in the case of AlphaGo [42]. In some

cases, the heuristic might be time consuming to compute and one would like to have a fast approximation for the same. We can use supervised learning to tackle this issue, where we go through an expensive training phase, but have an extremely fast inference phase which approximates the heuristics during test time. For example, [43] uses ML to approximate a strong branching strategy used in the branch-and-bound search tree. Last but not the least, optimization algorithms assume no knowledge of the previously solved problems. One might likely be solving many similar optimization problems, originating from a specific non-tractable probability distribution, without benefiting from the insights of previously solved problems. ML algorithms are inherently data-driven and hence can prove useful for solving problems coming from some specific, non-tractable, probability distribution. All of these factors combined acted as a catalyst for the field of CML. For a snapshot of the recent developments in this area, we point the reader to these surveys [44, 45].

Stochastic Programming and ML [46] proposes a hybrid strategy by combining scenario trees and ML to solve multi-stage stochastic programming problems. They specifically use supervised learning and scenario-tree approximations to construct a decision policy. They test this policy to quickly evaluate an approximation and selecting good scenario trees. [47] proposes a General Policy Function Approximation, based on Policy Function Approximation introduced in [48], to evaluate electricity portfolio policies. [49] introduces an end-to-end learning approach for stochastic programming problems, such that the predictions perform well on the downstream optimization task, rather than just excelling at the learning task. The crucial component of their algorithm is the ability to differentiate through an argmin operator, with some assumptions of strong convexity. [50] proposes to solve the Constrained Stochastic Program as an RL task, by combining the scalability of the RL with the modeling and constraint filtering techniques of constraint programming. One of the solution techniques to solve two-stage stochastic integer program (2SIP) is Benders Decomposition. Benders decomposition starts with a relaxation of the original problem, by only considering a few of the original constraints. It iteratively tries to find an intermediate solution and check if it satisfies all the constraints. We selectively add only those cuts which are violated by the intermediate solution. [51] uses supervised learning to improve Benders Decomposition, by selecting valuable cuts. A cut is deemed valuable if it significantly reduces the feasible region at a given iteration of the algorithm, or is active for the final solution. This helps in faster convergence of the algorithm.

Our work is mainly inspired by [52]. They proposed a reinforcement learning-based heuristic solver to quickly find solutions to 2SIP. Given that the agent can be trained offline, the algorithm provided solutions much faster for some classes of problems compared to an open-

source general-purpose mixed-integer programming (MIP) solver, in their case, SCIP [53,54]. However, their method is based on the following restrictive assumptions:

(i) All first-stage variables are required to be binary. General integer variables or continuous variables in the first stage cannot be handled.

(ii) Any assignment of the binary variables is required to be feasible for the first stage of the problem, i.e., no constraints are allowed in the first stage.

Assumption (i) above is intrinsic to the method in [52], as both the *initialization policy* and the *local move policy* of the method involves flipping the bits of the first-stage decision vector. Hence, one cannot easily have general integer variables or continuous variables. Assumption (ii) is again crucial to the algorithm in [52], as flipping a bit in the first stage could potentially violate constraints present in the first-stage and make the new decision infeasible. To deal with this issue, it might require a more complicated feedback mechanism to check and discard infeasible solutions. In fact, if there are constraints, it is NP-hard to decide if there exists a flip that keeps the decision feasible. Alternatively, one could empirically penalize the infeasible solutions, but tuning the value of penalty might be a hard problem in itself.

In contrast, our method does not require either of these two restrictive assumptions. We allow binary, general integer as well as continuous variables in both the first and second stages of the problem. We also allow constraints in both stages of the problem. Furthermore, we have a simple and direct approach to handle the first-stage constraints, without requiring any empirical penalties.

We make the following common assumption to exclude pathological cases, where an uncertain realization can turn a feasible first-stage decision infeasible.

Assumption 1 *The 2SIP has* complete recourse*, i.e., if a first-stage decision is feasible given the first-stage constraints, then it is feasible for all the second-stage problems as well.*

We make another assumption of uncertainty with finite support, so we can have a proper benchmark to compare our solution against. However, this assumption can be readily removed, without affecting the proposed algorithm.

Assumption 2 *The uncertainty distribution in the 2SIP has a finite support.*

2.3 Problem Definiton

We formally define a 2SIP as follows:

$$\min_{x \in \mathbb{R}^{n_1}} \quad c^\intercal x + \mathbb{E}_\xi \left[Q(x, \xi) \right] \tag{2.1a}$$

$$\text{subject to} \quad Ax \leq b \tag{2.1b}$$

$$x_i \in \mathbb{Z}, \quad \forall\, i \in \mathcal{I}_1 \tag{2.1c}$$

where,

$$Q(x, \xi) \quad = \quad \min_{y \in \mathbb{R}^{n_2}} \left\{ q_\xi^\intercal y_\xi : W y_\xi \leq h_\xi - T_\xi x, y_\xi \geq 0; y_i \in \mathbb{Z}\ \forall\, i \in \mathcal{I}_2 \right\}$$

where $x \in \mathbb{R}^{n_1}$ and $y \in \mathbb{R}^{n_2}$ are the first and second-stage decisions respectively, $c \in \mathbb{R}^{n_1}$, $A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, $y_\xi \in \mathbb{R}^{n_2}$, $q_\xi \in \mathbb{R}^{n_2}$, $W \in \mathbb{R}^{m_2 \times n_2}$, $T_\xi \in \mathbb{R}^{m_2 \times n_1}$, $h_\xi \in \mathbb{R}^{m_2}$, $\mathcal{I}_1 \subseteq \{1, \ldots, n_1\}, \mathcal{I}_2 \subseteq \{1, \ldots, n_2\}$, $\xi \in \Xi$ and where $(\Xi, \mathscr{F}_\Xi, p)$ defines a probability space.

When Assumption 2 holds, the 2SIP described above can also be expressed as a single deterministic MIP as follows:

$$\min_{x,y} \quad c^\intercal x + \sum_{\forall \xi \in \Xi} p_\xi q_\xi^\intercal y_\xi \tag{2.2a}$$

$$\text{subject to} \quad Ax \leq b \tag{2.2b}$$

$$W y_\xi \leq h_\xi - T_\xi x \qquad \forall \xi \in \Xi \tag{2.2c}$$

$$x_i \in \mathbb{Z}, \quad \forall\, i \in \mathcal{I}_1 \tag{2.2d}$$

$$y_{\xi i} \in \mathbb{Z}, \quad \forall \xi \in \Xi, \forall\, i \in \mathcal{I}_2. \tag{2.2e}$$

where, Ξ is the set of random scenarios and p_ξ is the probability of a random scenario $\xi \in \Xi$.

When Assumption 2 does not hold, the formulation Equation (2.2) could be a finite-sample approximation of Equation (2.1), which is extensively studied in the stochastic programming literature. Imitating [52], we compare our algorithm against solving Equation (2.2) with a general-purpose MIP solver.

2.4 Methodology

This section details the methodological contribution to solve an SIP quickly using machine learning.

2.4.1 Surrogate Formulation

We first define the objective value function (OVF) $\Phi : \mathbb{R}^{n_1} \to \mathbb{R}$, mapping $x \mapsto c^\intercal x + \mathbb{E}_\xi [Q(x, \xi)]$ - the function we are trying to optimize over the mixed-integer set defined in Equation (2.1).

Given Equation (2.2), we define *the surrogate problem associated with* $\xi^\dagger = (q^\dagger, h^\dagger, T^\dagger)$, as follows:

$$\min_{x,y} \quad c^\intercal x + (q^\dagger)^\intercal y \tag{2.3a}$$

$$\text{subject to} \quad Ax \leq b \tag{2.3b}$$

$$Wy \leq h^\dagger - T^\dagger x \tag{2.3c}$$

$$x_i, y_j \in \mathbb{Z}, \quad \forall\, i \in \mathcal{I}_1;\, j \in \mathcal{I}_2 \tag{2.3d}$$

In other words, a surrogate formulation is a special case of the extensive formulation with the set of scenarios $\Xi = \{\xi^\dagger\}$. Note that $\xi^\dagger$ need not be equal to one of the scenarios used to formulate the extensive formulation given by Equation (2.2). Now, the idea behind the algorithm proposed in the thesis is captured by Conjecture 1.

Conjecture 1 *Let Equation* (2.2) *(and hence Equation* (2.1)*) have an optimal objective value of* f^**. There exists* $\xi^\star = (q^\star, h^\star, T^\star)$ *such that if* $(x^\dagger, y^\dagger)$ *solves the (much smaller) surrogate problem defined by* $\xi^\dagger = \xi^\star$*, then,* $f^* = \Phi(x^\dagger)$*.*

Observe that by construction, $x^\dagger$ is feasible to the original problem in Equation (2.1). Also, Conjecture 1 asserts that, there exists a realization of the uncertainty $\xi^\star$ such that if one deterministically optimizes for that realization $\xi^\star$, then its solutions are optimal for the original 2SIP. Each such $\xi^\star$ is called a *representative scenario* (RS) for the given 2SIP.

Now, given adequate computing resources, one could solve the following bilevel program to

obtain an RS.

$$\min_{\substack{U,v,w \\ x,y}} \quad c^\intercal x + \sum_{\forall \xi \in \Xi} p_\xi q_\xi^\intercal y_\xi \tag{2.4a}$$

$$\text{subject to} \quad (x,w) \in \arg\min_{x,w} \left\{ c^\intercal x + v^\intercal w : \begin{array}{ll} Ax \leq b; & \\ Ww \leq v - Ux; & \\ x_i \in \mathbb{Z} & \forall\, i \in \mathcal{I}_1 \\ w_i \in \mathbb{Z} & \forall\, i \in \mathcal{I}_2 \end{array} \right\} \tag{2.4b}$$

$$Wy_\xi \leq h_\xi - T_\xi x \qquad \forall \xi \in \Xi \tag{2.4c}$$

$$y_{\xi i} \in \mathbb{Z}, \quad \forall \xi \in \Xi, \forall\, i \in \mathcal{I}_2 \tag{2.4d}$$

If the optimal value of this problem matches the optimal value of the original 2SIP, then the corresponding values for (U, v, w) form the RS. Note that if T_ξ is the same for all $\xi \in \Xi$, then Equation (2.4) is a mixed-integer bilevel *linear* program (MIBLP) and can hopefully be solved faster than the general case.

2.4.2 Learning Algorithm

The goal of ML algorithms is to predict an optimal (U, v, w) to Equation (2.4), given the data for the 2SIP. On the one hand, we are expecting the ML algorithms to predict the solutions of a seemingly much harder optimization problem than the original 2SIP. On the other hand, this is easier for ML since there are no constraints on the predicted variables – U, v, w. Supervised learning is a natural tool to achieve this goal.

Supervised learning can be used if there is a training dataset of problem instances and their corresponding RS. The task of predicting RS can be formulated as a regression task as RS is real valued. The algorithm tries to minimize the mean squared error (MSE) between the true and predicted RS. The prediction can also be evaluated on the merits of optimization metrics, comparing the solution and objective value of a true and predicted RS.

2.5 Computational Study

This section presents the computational results that can help assess the performance of our methodology in different problem settings. For the same, we adopt a three-phase strategy.

Phase 1: We check the performance of our methodology on the stochastic capacitated facility location problem (S-CFLP) for a fixed size of instances, details of which are given in the experiment 1 of Section 2.5.1.

Phase 2: We check the performance of our methodology on multiple sizes of the S-CFLP, details of which are given in the experiments 2 and 3 of Section 2.5.1. This will help assess the scalability of our approach to handle multiple sizes, after some positive results with fixed size instances of the S-CFLP.

Phase 3: We check the performance of our methodology on the stochastic generealized assignment problem (S-GAP) to assess its scalability to a new problem type, details of which are given in the experiment 1 of Section 2.5.2.

Note that the results presented in [23] are also under *similar* settings as in *Phase 1*. The similarity lies in terms of the problem type and size of instances that are tested. In both cases, we work with the S-CFLP with 10 facilities and 50 scenarios. The differences are in terms of the features, the computational platform to perform the experiments and the optimizer used by the ML algorithms. In [23], the dimension of the features used is 190, whereas in *Phase 1* the dimension of the features used is 140. We talk about the modified feature engineering in Section 2.5.1. Also, the experiments presented in [23] are performed on a local server, whereas the experiments of *Phase 1* are performed on Beluga cluster, Compute Canada. We used stochastic gradient descent (SGD) with momentum as optimizer for experiments in [23], whereas we use Adam in *Phase 1* experiments. Apart from these, *Phase 1* contains additional results about the performance of the ML algorithms on the instances for which we were not able to find an representative scenario (RS), i.e., unlabeled set. However, the key conclusion of the ML models being able to obtain good primal solutions as compared to Gurobi, in significantly less time, still holds.

Comparisons In order to evaluate the ML-based prediction of $\xi^\star$, which we refer to as $\hat{\xi^\star}$, we compare the solution obtained by solving the surrogate problem associated with $\xi^\dagger = \hat{\xi^\star}$ against solutions obtained by various algorithms.

We use different supervised learning models like Linear Regression (LR), FFNN and CNN to predict an RS. We compare these predictions against

1. Solutions obtained using Gurobi by solving Equation (2.2) (GRB). For the S-CFLP, the percentage gap allowed was 2% and for the S-GAP, we solved it to optimality.

2. Solutions obtained by solving the surrogate associated with the average scenario, namely $\sum_{k=1}^{N} \xi_k/N$ (AVG), where $N = |\Xi|$ and ξ_k is the k^{th} scenario in Ξ.

3. Solutions obtained by solving the surrogate associated with a randomly chosen scenario from the N choices (RND).

4. Solutions obtained by solving the surrogate associated with a randomly chosen scenario from the distribution of the scenario predicted by LR (DIST).

Note that GRB produces better solutions (in most cases) than the ML methods, however, taking a significantly longer time. We, therefore, assess the time it takes GRB to get a solution of comparable quality to the ML methods. This time is referred to as GRB-X, which is the time taken by Gurobi to beat an ML method X.

Performance metrics

1. **Objective Value Difference Ratio**: We define the objective value difference ratio (OVDR) as a relative measure of how well a given method B performs with respect to a reference method A, in terms of the objective value obtained by them in the optimization process. Let $\texttt{OBJ}^A$ and $\texttt{OBJ}^B$ be the objective value of the method A and method B, respectively.

$$\text{OVDR} \equiv \frac{\texttt{OBJ}^B - \texttt{OBJ}^A}{\texttt{OBJ}^A}$$

   In the context of this thesis, we compare the objective value of different methods ($\texttt{OBJ}^M$) with respect to to the objective value obtained by solving the extensive form of 2SIP using Gurobi [55] ($\texttt{OBJ}^{GRB}$). Thus, OVDR can also be stated as,

$$\text{OVDR} \equiv \frac{\texttt{OBJ}^M - \texttt{OBJ}^{GRB}}{\texttt{OBJ}^{GRB}}$$

   Also, for the problems defined in this thesis, $\texttt{OBJ}^{GRB}$ is always positive. Hence, when OVDR is negative, the objective value of the method is better than that obtained by the extensive form using Gurobi (assuming we are solving a minimization problem) and vice versa. To make the comparison easier, we evaluate the percentage OVDR (OVDR%) by multiplying the original OVDR with 100.

$$\text{OVDR\%} \equiv \text{OVDR} \times 100$$

   For the remainder of this thesis, OVDR refers to OVDR% unless otherwise stated explicitly.

2. **Mean Squared Error**: The mean squared error (MSE) between two $m > 0$ dimensional quantities, $\mathbf{y}$ and $\hat{\mathbf{y}}_{\boldsymbol{\theta}}$, can be defined as:

$$\text{MSE} \equiv \frac{\sum_m (\hat{\mathbf{y}}_{\boldsymbol{\theta}} - \mathbf{y})^2}{m}$$

In our case, we use MSE to evaluate the empirical risk $L(\boldsymbol{\theta})$ of an ML algorithm (estimator) for a multi-output regression task. In the same context, $\hat{\mathbf{y}}_{\boldsymbol{\theta}}$ refers to the prediction of the estimator for some input $\mathbf{x}$ and $\mathbf{y}$ is its true label. Hence,

$$\text{MSE}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) = \frac{\sum_m (\hat{\mathbf{y}}_{\boldsymbol{\theta}} - \mathbf{y})^2}{m} = \frac{\sum_m (h_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y})^2}{m}$$

where $h_\theta : \mathcal{X} \to \mathcal{Y}, x \in \mathcal{X}, y \in \mathcal{Y}$. Also, for performing empirical risk minimization using SGD, we evaluate the batch loss. Let the number of samples in the batch be n, $\hat{\mathbf{Y}}_{\boldsymbol{\theta}} = [\hat{\mathbf{y}}_{\boldsymbol{\theta}}^{(1)}, \hat{\mathbf{y}}_{\boldsymbol{\theta}}^{(2)}, \ldots, \hat{\mathbf{y}}_{\boldsymbol{\theta}}^{(n)}]$ be the vector of predictions and $\mathbf{Y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n)}]$ be the vector of labels, then, the empirical risk is equal to

$$\text{MSE}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) = \frac{\sum_n \sum_m (\hat{\mathbf{Y}}_{\boldsymbol{\theta}} - \mathbf{Y})^2}{n \cdot m}$$

3. **R-squared Score**: The R-squared (R^2) score is a metric to gauge the ability of an estimator $h_{\boldsymbol{\theta}}$ to capture the variance in the labels $\mathbf{y}^{(i)}$s by the corresponding predictions $\hat{\mathbf{y}}_{\boldsymbol{\theta}}^{(i)}$s, respectively. Semantically, it tries to capture the following relation:

$$R^2 \equiv \frac{\text{Explained variation}}{\text{Total variation}} \equiv 1 - \frac{\text{Unexplained variation}}{\text{Total variation}}$$

Let $\bar{\mathbf{y}}$ be the mean of n labels,

$$\bar{\mathbf{y}} = \frac{\sum_{i=1}^{n} \mathbf{y}^{(i)}}{n}$$

then, we can define total variation, explained variation and unexplained variation as

$$\text{Total variation} \equiv \sum_{i=1}^{n} \left(\bar{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\right)^2$$

$$\text{Explained variation} \equiv \sum_{i=1}^{n} \left(\bar{\mathbf{y}}^{(i)} - \hat{\mathbf{y}}_{\boldsymbol{\theta}}^{(i)}\right)^2$$

$$\text{Unexplained variation} \equiv \sum_{i=1}^{n} \left(\hat{\mathbf{y}}_{\theta}^{(i)} - \mathbf{y}^{(i)}\right)^2$$

Usually, R^2 lies between 0 and 1; 0 indicates that the estimator is not able to capture any variation in the labels and 1 indicates that the estimator is fully able to explain

the variation in the labels. There are cases in which it can also be negative. A negative value implies that the mean of the labels is better able to capture the variation than the predictions of the estimator.

However, solely considering the R^2 to judge an estimator might be misleading; there are cases in which low and high R^2 are reported for good and poorly performing models, respectively [56].

2.5.1 Stochastic Capacitated Facility Location Problem

Problem Formulation

We consider a stochastic capacitated facility location problem (S-CFLP), with binary and continuous variables both in the first and second stages. Let F be the set of potential facilities that can be opened. We assume that the number of clients is equal to the number of facilities. Hence we can also use the set F to refer the clients. The demand of the clients are random. To ensure complete recourse, i.e., every client's demand is satisfied, we add a hub facility (or hub) that can satisfy infinite demand in the second-stage. Let F' be the set of facilities including the hub. The first-stage decision is to select a subset of facilities from F and decide what capacity to install in each of them. Once the client demands are realized, we are allowed to make the second-stage decision which assigns the open facilities to clients to satisfy their demand. Table 2.1 contains the notations used to formulate the S-CFLP.

Table 2.1 Notations used to define an instance of the S-CFLP.

Data	Definition
c_i^f	Fixed cost of opening facility i
c_i^v	Variable cost of opening facility i
c_{ij}^{tf}	Fixed cost of transporting from facility i to client j
c_{ij}^{tv}	Variable cost of transporting from facility i to client j
ξ_k^j	Demand of client j in scenario k
Variable	**Definition**
b_i	$\begin{cases} 1, & \text{if facility } i \text{ is open} \\ 0, & \text{otherwise} \end{cases}$
v_i	Capacity installed at facility i
u_{ij}	$\begin{cases} 1, & \text{if facility } i \text{ satisfies demand of client } j \\ 0, & \text{otherwise} \end{cases}$
y_{ij}	Supply from facility i to client j

Using the definitions given in Table 2.1, we formulate the S-CFLP as follows:

$$\min_{b,v} \quad \sum_{i\in F}\left(c_i^f b_i + c_i^v v_i\right) + \frac{1}{N}\sum_{k=1}^{N} Q(b,v,\xi_k) \tag{2.5}$$

$$\frac{|F|}{10} \quad \leq \quad \sum_{i\in F} b_i \quad \leq \quad \frac{3\ |F|}{4} \tag{2.6}$$

$$v_i \quad \leq \quad Mb_i \qquad \forall i \in F \tag{2.7}$$

$$b_i \quad \in \quad \{0,1\} \qquad \forall i \in F \tag{2.8}$$

$$v_i \quad \geq \quad 0 \qquad \forall i \in F \tag{2.9}$$

Here $Q(b,v,\xi_k)$ is defined as the optimal value of the problem:

$$\min_{u,y} \quad \sum_{i\in F'}\sum_{j\in F}\left(c_{ij}^{tf}u_{ij} + c_{ij}^{tv}y_{ij}\right) \tag{2.10}$$

$$\sum_{j\in F} y_{ij} \quad \leq \quad v_i \qquad \forall i \in F \tag{2.11}$$

$$\sum_{i\in F'} y_{ij} \quad = \quad \xi_k^j \qquad \forall j \in F \tag{2.12}$$

$$y_{ij} \quad \leq \quad Mu_{ij} \qquad \forall i \in F', \forall j \in F \tag{2.13}$$

$$u_{ij} \quad \leq \quad b_i \qquad \forall i \in F, \forall j \in F \tag{2.14}$$

$$u_{ij} \quad \in \quad \{0,1\} \qquad \forall i \in F', \forall j \in F \tag{2.15}$$

$$y_{ij} \quad \geq \quad 0 \qquad \forall i \in F', \forall j \in F \tag{2.16}$$

We minimize the fixed and variable costs of opening a facility along with the fixed and variable costs of transportation between the facilities and clients in Equation (2.5). A fixed cost of c_i^f is incurred, if a facility is opened in location i, and a variable cost of c_i^v is incurred per-unit capacity of the facility opened in location i. The binary variable, b_i tracks if a facility is opened in location i and the continuous variable v_i indicates the size of the facility at location i. The constraints in Equation (2.7), Equation (2.8) and Equation (2.9) ensure that the costs are incurred in the right way. Finally, Equation (2.6) is a complicating constraint, which says that at least a tenth of the locations must have a facility open and not more than three-quarters of the locations must have a facility open.

The second-stage objective in Equation (2.10) minimizes the transportation cost incurred under a random demand scenario ξ_k. Then, Equation (2.11) ensures that the total quantity transported out of a facility is not greater than the capacity of the facility, while Equation (2.12) ensure that the total quantity supplied to a location j equals the (random) demand

at j. Finally, constraints Equation (2.13) link u and y variables appropriately.

Data generation

1. **Instance generation**: Table 2.2 gives the data sampling details for generating an instance of the S-CFLP.

   Table 2.2 Data sampling details for generating an instance of the S-CFLP

Variable	Sampling method
c^f	`DiscreteUniform[15, 20)`
c^v	`DiscreteUniform[5, 10)`
ξ_k	`Poisson(`λ`)` $\lambda = \lfloor (c^f + 10 * c^v)/\sqrt{\lvert F \rvert} \rfloor$

   Note that c^{tf} and c^{tv} remain fixed across instances of the same size.

2. **Optimal solution generation**: The generated instances are solved within 2% gap and 10 minutes time limit using Gurobi. We are able to solve all the instances to the specified gap within the specified time limit.

3. **Representative scenario generation**: We employ an iterative method to construct an RS. Let $\xi^\dagger$ be initialized with the average scenario $\sum_{k=1}^{N} \xi_k/N$, $x^\dagger \equiv (b^\dagger, v^\dagger)$ and $\texttt{OBJ}^\dagger \equiv \Phi(x^\dagger)$ be the solution and objective value of the surrogate problem constructed using $\xi^\dagger$, respectively, and $x^* \equiv (b^*, v^*)$ and $\texttt{OBJ}^*$ be the near-optimal optimal solution and objective value, respectively. Whenever the optimization problem is posed as a minimization task, we try to find an RS from the set $\Xi^\star \equiv \left\{\xi^\dagger : \texttt{OBJ}^\dagger \leq c * \texttt{OBJ}^*\right\}$, where c is a positive constant (1.01 in our case). We use a mix of three heuristics to update $\xi^\dagger$.

   Heuristic 1: If a facility $i \in F$ is closed in the near-optimal solution and open in the surrogate solution, then we zero out the demand on that location in surrogate, i.e.,

   $$b_i^* = 0 \wedge b_i^\dagger = 1 \implies \xi_i^\dagger = 0 \tag{2.17}$$

   Heuristic 2: Update the demand on location $i^{max} \equiv \underset{i}{\text{argmax}} \ |v_i^* - v_i^{\bar{\xi}}|$ by p percentage of the current demand. The direction of the update is determined by the difference of

optimal and surrogate capacities installed on those locations:

$$\xi^{\dagger}_{i^{max}} = \xi^{\dagger}_{i^{max}} + \frac{v^{*}_{i^{max}} - v^{\dagger}_{i^{max}}}{|v^{*}_{i^{max}} - v^{\dagger}_{i^{max}}|}\ p\ \xi^{\dagger}_{i^{max}} \tag{2.18}$$

Heuristic 3: Update the demand on location i^{max} by a fraction f of the difference of capacities in the optimal solution and surrogate solution.

$$\xi^{\dagger}_{i^{max}} = \xi^{\dagger}_{i^{max}} + (v^{*}_{i^{max}} - v^{\dagger}_{i^{max}})\ f\ \xi^{\dagger}_{i^{max}} \tag{2.19}$$

We quantify the aggressiveness of a heuristic by how smooth an RS we get using it. Here, heuristic 1 is more aggressive in the sense that there will be more zeros induced in an RS generated using it. Whereas heuristic 2 and heuristic 3 are milder. Also, during our experiments, we were able to find more RS using heuristic 1 as compared to other heuristics. To generate an RS, we start with the milder heuristics. If we are not able to find an RS using them, then we use the aggressive one.

Feature Engineering

It is well known that features describing the connection between variables, constraints and other interactions help ML to perform well rather than just providing plain data matrices [43, 44, 57, 58]. In this spirit, along with the fixed and variable costs to open facilities at different locations, we also provide aggregated features on the set of scenarios. These features give information about each of the potential locations for facilities in S-CFLP as well as the way different locations interact through the demands in adjacent nodes.

Let Ξ be an $N \times |F|$ matrix, where N is the number of scenarios and $|F|$ is the number of clients. We calculate the minimum, maximum, average, standard deviation, median, 75^{th} quantile, and 25^{th} quantile of $\Xi_{[:,i]}$ (i^{th} column of Ξ) for $i = 1, \ldots, |F|$, resulting in a total of $7 \times |F|$ features.

We also find the percentage of scenarios in which some fraction c of the demand for a client is greater than than the demand on all the other nodes

$$\frac{\sum_m \mathbb{1}_{c\Xi_{[:,i]} \geq \Xi_{[:,\neq i]}}}{N}$$

The dimension of the indicator function is equal to the number of scenarios N. We assign c to different values from the set $\mathcal{C} = \{0.9, 1, 1.1, 1.2, 1.5\}$ and end up with $|\mathcal{C}| \times |F|$ features. Note

that $|\mathcal{C}| = 5$ as $\mathcal{C}$ remains fixed across experiments. Along with these, we also have the c^f and c^v adding another $2 \times |F|$ features. Thus, in total we have $(7 \times |F|) + (|\mathcal{C}| \times |F|) + (2 \times |F|) = 14 \times |F|$ features.

Results

We conduct three experiments to evaluate our approach. The three experiments performed were increasingly difficult for the learning algorithm to achieve good generalization. Also, we started using Adam optimizer instead of SGD with momentum, in PyTorch. We set the learning rate and weight penalty of the Adam optimizer to 10^{-3} and 10^{-5}, respectively.

Experiment 1 We start with a minimal setup to evaluate the claim made in Conjecture 1, where the size and number of scenarios (NOS) for all the instances in the dataset were fixed. We generate 50K instances of S-CFLP with 10 facilities, 10 clients and 50 scenarios. Using heuristics, we were able to find an RS for 49,290 instances. The remaining 710 instances, for which we are not able to find an RS, are referred to as an unlabeled set. We use 45K instances in the train set and the remaining 4290 instances in the test set. We create a 140 dimensional input feature vector for the ML models. The output dimension of the ML models is equal to 10, as we predict a representative demand at each of the 10 locations.

Table 2.3 reports the statistics of OVDR and Table 2.4 statistics on computing times, for different methods to estimate an RS. We report our results on the test and unlabeled sets. FFNN^1 and FFNN^2 in Table 2.3 refer to two different configurations of an FFNN; both of them have the same input and output dimensions, however, the former has a single hidden layer of dimension 256 as compared to the latter with two similar hidden layers. In the context of experiment 1, we jointly refer to FFNN^1 and FFNN^2 as FFNN, instead of explicitly making a distinction of their configuration, if the observation holds for both of them. GRB-L, GRB-F^1 and GRB-F^2 in Table 2.4 refer to the time taken by Gurobi to produce a solution as good as an LR, FFNN^1 and FFNN^2, respectively.

On the test set, we observe from Table 2.3 that LR and FFNN produce decisions that are as good as GRB ones on an average (and by the median value), and in some cases, the ML-based methods even beat GRB, i.e., produce solutions whose objective value is better than that of GRB. This is possible because GRB does not necessarily solve the problem to optimality, but only up to a gap of 2%. Further, even in the worst of the 4,290 test cases, LR is at most 2.67% away from GRB. To show that this is not easily achieved, we also compare GRB against baselines like AVG, RND and DIST. We observe from Table 2.3 that these methods perform much poorer than GRB, unlike LR and FFNN.

On the unlabeled set, we observe an increase in the average, min and median of the OVDR for all methods, as compared to the test set, from Table 2.3. ML based methods are still able to produce good decisions, with minimum average OVDR of 0.83% for LR and a maximum average OVDR of 1.14% for FFNN^2. The relative performance of the ML based methods in terms of average OVDR on the test set also holds on the unlabeled set, i.e., LR $<$ FFNN^1 $<$ FFNN^2 in terms of average OVDR. Also, the maximum OVDR of the ML based methods on the unlabeled set is similar to that on the test set.

Table 2.3 Experiment 1 OVDR statistics (in %) of different methods.

GRB vs.	Avg. ± Stdev.	Min	Median	Max
Test set				
AVG	8.12 ± 1.61	3.21	8.08	14.00
RND	12.15 ± 12.13	0.60	8.11	89.05
DIST	5.13 ± 4.60	-0.28	3.68	43.25
LR	**0.65 ± 0.42**	-0.55	0.61	2.67
FFNN^1	0.78 ± 0.50	-0.45	0.71	3.36
FFNN^2	0.87 ± 0.59	-0.46	0.77	4.64
Unlabeled set				
AVG	9.43 ± 1.87	4.08	9.45	14.57
RND	13.37 ± 12.96	0.15	9.31	81.10
DIST	6.83 ± 4.66	0.15	6.08	25.56
LR	**0.83 ± 0.41**	-0.28	0.81	2.69
FFNN^1	1.00 ± 0.50	-0.14	0.93	3.48
FFNN^2	1.14 ± 0.62	-0.07	1.06	4.27

Analyzing the time improvement in using LR and FFNN, we observe from Table 2.4 that these methods solve the S-CFLP *orders of magnitude* faster than GRB, both on the test and unlabeled sets. On the one hand, GRB takes over 6 and 12 seconds, on average, on the test and unlabeled sets, respectively. On the other hand, LR takes a maximum of 0.02 seconds and an average of 0.003 seconds, on both the test and unlabeled sets. The time taken to solve the S-CFLP using the ML methods *includes* is the time in computing the features, predicting an RS and solving the surrogate problem associated with the predicted RS. This underscores the fact that our method is able to produce good solutions for the S-CFLP, which could be useful in itself or used in tandem with exact solvers like Gurobi for improved efficiency. In fact, LR and FFNN are still orders of magnitude faster than GRB.

We observe that the standard ML objective of improving generalization, i.e., the performance on the test set, as measured by MSE, is achieved by choosing a model with more parameters.

The MSE achieved by LR is 71.11, $FFNN^1$ is 44.64 and $FFNN^2$ is 38.50. In particular, the generalization of $FFNN^1$ is better than that of LR and the performance of $FFNN^2$ is better than that of $FFNN^1$. In contrast to the above observation, when we measure the performance using average OVDR, we observe that LR has a better performance compared to both $FFNN^1$ and $FFNN^2$. But, given that OVDR is known to be a discontinuous function, learning by trying to minimize the empirical OVDR is a harder task, forcing us to use a proxy like that of MSE, as we have done above. We do not fully understand the reliability of differentiable metrics like the MSE when our ultimate goal is to minimize OVDR, and we leave this analysis to future work.

Table 2.4 Experiment 1 computing time statistics (in seconds) of different methods.

Method	Avg. ± Stdev.	Min	Median	Max
Test set				
GRB	6.2592 ± 14.3423	0.2544	1.3688	468.9182
AVG	0.0025 ± 0.0005	0.0023	0.0024	0.0226
RND	0.0025 ± 0.0005	0.0023	0.0024	0.0145
DIST	0.0025 ± 0.0006	0.0023	0.0024	0.0194
GRB-L	3.0190 ± 9.0502	0.2220	0.8522	468.9182
$GRB\text{-}F^1$	2.6158 ± 8.6584	0.2220	0.8190	468.9182
$GRB\text{-}F^2$	2.4687 ± 5.9890	0.2173	0.8041	110.0594
LR	0.0028 ± 0.0005	0.0026	0.0027	0.0160
$FFNN^1$	0.0028 ± 0.0005	0.0026	0.0027	0.0180
$FFNN^2$	0.0028 ± 0.0004	0.0024	0.0027	0.0136
Unlabeled set				
GRB	12.2431 ± 18.8957	0.4950	11.9533	216.8782
AVG	0.0026 ± 0.0011	0.0023	0.0024	0.0161
RND	0.0025 ± 0.0004	0.0023	0.0024	0.0089
DIST	0.0025 ± 0.0005	0.0023	0.0024	0.0131
GRB-L	4.4137 ± 9.0979	0.3042	0.9319	94.1244
$GRB\text{-}F^1$	3.6492 ± 8.6216	0.2918	0.8811	94.1244
$GRB\text{-}F^2$	2.9721 ± 7.1220	0.2918	0.8451	94.1244
LR	0.0029 ± 0.0008	0.0026	0.0027	0.0155
$FFNN^1$	0.0028 ± 0.0009	0.0026	0.0027	0.0155
$FFNN^2$	0.0029 ± 0.0012	0.0025	0.0027	0.0201

Experiment 2 After some initial validation of the idea with fixed size and NOS equal to 10 and 50, respectively, we focus on creating a model that can predict an RS for multiple sizes and NOS. Hence, we try to train a model that can handle sizes between 15 and 25, and

NOS between 25 and 50. The key step to create such a model is to generate feature vectors of the same dimension for instances of different size-NOS combinations. If we consider an instance with the maximum possible size of 25 and maximum possible NOS of 50, for this experiment, we will end up with a 350 dimensional feature vector. The process of creating features is described in point 4 of Section 2.5.1. For any instance with sizes or NOS less than the maximum possible, we add relevant padding to convert it to a 350-dimensional feature vector. We normalize the features between -1 and 1, and use -2 for padding to indicate null values in the feature vector. We generate 50 instances for a given size-NOS combination, giving a total of 27,300 instances.

The upper bound on size and NOS is decided based on the following criteria:

- Time to solve the extensive form of S-CFLP to 2% gap,
- Time to find an RS using heuristics,
- Number of labeled instances for the learning algorithm.

Out of these 27,300 instances, we were able to label 23,539 instances, and are left with 3761 instances being unlabeled. We use 85% of the labeled instances to create the train set and the remaining 15% for the test set.

Table 2.5 Experiment 2 OVDR statistics (in %) of different methods.

GRB vs.	Avg. ± Stdev.	Min	Median	Max
Test set				
AVG	12.29 ± 6.75	0.00	12.09	32.88
RND	17.42 ± 20.53	-0.38	9.59	216.38
DIST	20.84 ± 14.65	-0.11	19.49	70.46
LR	**1.00 ± 0.57**	-0.35	0.94	4.88
FFNN	2.17 ± 2.40	-0.32	1.42	28.08
CNN	1.82 ± 1.63	-0.46	1.34	17.38
Unlabeled set				
AVG	19.47 ± 5.10	1.42	19.94	37.12
RND	27.85 ± 26.92	-0.17	20.19	206.63
DIST	34.65 ± 12.87	0.40	36.62	70.04
LR	**1.27 ± 0.64**	-0.22	1.16	6.31
FFNN	4.34 ± 3.63	0.03	3.08	24.92
CNN	3.20 ± 2.19	0.08	2.66	16.75

The FFNN considered for these experiments contains two hidden layers. The input layer contains 350 neurons, the hidden layers contain 256 neurons each and the output layer contains 25 neurons. We use ReLU activation in the hidden and output layers. The CNN consists of 3 convolutional layers, each followed by a max-pooling layer. Every convolutional layer has a kernal size 3, stride 1 and no-padding. The number of kernels in the three convolutional layers are 16, 32 and 64, respectively. Each of the max-pooling layers has a kernel size 3 and stride 3. Consequently, the 1×350 shaped input is successively transformed into features of shape $16 \times 116, 32 \times 38$ and 64×12 by the three layers. The last 64×12 shaped feature is flattened to get a 768 dimensional feature, which is then passed through a FFNN to produce a 25-dimensional output.

Table 2.5 contains the OVDR statistics and Table 2.6 contains the time statistics for experiment 2, both on train and unlabeled sets. The key result from experiment 1 still holds, i.e., the ML methods, especially LR, are able to achieve good average OVDR in a fraction of the time taken by Gurobi. We observe that average OVDR has gone up for all the methods as compared to experiment 1, however, this increase is more drastic for the baseline methods. AVG achieves the best average OVDR of 12.29% on the test set and 19.47% on the unlabeled set, among the baselines. LR, on the other hand, is able to achieve an average OVDR of 1% and 1.27% on the train set and the unlabeled set, respectively.

In terms of time, on average, GRB takes around 9 seconds and 23 seconds on the train set and the unlabeled set, respectively. Whereas, LR takes 0.006 seconds and 0.01 seconds, on an average, on the train set and the unlabeled set, respectively. Also, GRB-L suggests that it is still time consuming for Gurobi to produce a solution of the same quality as an LR.

Experiment 3 In this experiment, we further scrutinize the generalization ability of our model, by training on some size-NOS combinations and testing on, partially or completely, unseen size-NOS combinations. We generate 5000 instances for each of the (size-NOS) combination from the set {(10, 25), (15, 25), (15, 50), (25, 50)}. For the size-NOS combinations (10, 25), (15, 25), (15, 50), (25, 25) and (25, 50), we were able to find an RS in 4663, 4477, 4654, 3005 and 3273 instances, respectively. The increased number of labeled instances with more NOS can be attributed to the fact that for any given size, solving their extensive form with more scenarios makes the problem harder and increases the objective value. This increases the 1% gap barrier value for classifying a scenario as an RS, eventually increasing the space of scenarios that can be classified as an RS. We sample 5000 labeled instances of (10, 50) from the first experiment. We use 85% of the labeled instances for training, giving us a train set of size 21,309 and a test set of size 3763. We refer to this sub-class of instances in the test set as seen-size-seen-NOS (SSSS), as we have already seen these (size,

Table 2.6 Experiment 2 computing time statistics (in seconds) of different methods.

Method vs.	Avg. ± Stdev.	Min	Median	Max
Test set				
GRB	9.4537 ± 26.5455	0.0220	2.2679	411.2703
AVG	0.0059 ± 0.0042	0.0010	0.0045	0.0271
RND	0.0058 ± 0.0042	0.0009	0.0044	0.0329
DIST	0.0057 ± 0.0041	0.0010	0.0044	0.0264
GRB-L	2.2493 ± 6.6935	0.0220	1.1659	266.2290
GRB-F	1.4666 ± 4.2472	0.0220	0.8848	201.1213
GRB-C	1.5715 ± 5.3128	0.0220	0.8997	201.1213
LR	0.0061 ± 0.0040	0.0014	0.0048	0.0161
FFNN	0.0062 ± 0.0040	0.0015	0.0049	0.0213
CNN	0.0062 ± 0.0040	0.0014	0.0049	0.0201
Unlabeled set				
GRB	23.4069 ± 49.0136	0.0914	5.1477	599.0264
AVG	0.0101 ± 0.0037	0.0014	0.0104	0.0377
RND	0.0101 ± 0.0037	0.0010	0.0104	0.0302
DIST	0.0098 ± 0.0036	0.0010	0.0102	0.0266
GRB-L	3.3761 ± 9.0150	0.0483	2.1077	229.5173
GRB-F	1.8142 ± 2.7233	0.0483	1.4327	70.9094
GRB-C	1.9464 ± 3.4336	0.0483	1.4741	115.3235
LR	0.0103 ± 0.0035	0.0019	0.0107	0.0353
FFNN	0.0105 ± 0.0036	0.0020	0.0108	0.0475
CNN	0.0105 ± 0.0037	0.0019	0.0109	0.0377

Table 2.7 Description of the test set detailing the sub-classes, size-NOS combinations and the corresponding number of instances.

Sub-class	(size-NOS)	# instances	Total
SSSS	(10, 25)	700	3763
	(10, 50)	750	
	(15, 25)	672	
	(15, 50)	699	
	(25, 25)	451	
	(25, 50)	491	
Unlabeled-SSSS	(10, 25)	337	4928
	(10, 50)	0	
	(15, 25)	523	
	(15, 50)	346	
	(25, 25)	1995	
	(25, 50)	1727	
SSUS	(10, 10)	100	1200
	(10, 40)	100	
	(10, 65)	100	
	(10, 100)	100	
	(15, 10)	100	
	(15, 40)	100	
	(15, 65)	100	
	(15, 100)	100	
	(25, 10)	100	
	(25, 40)	100	
	(25, 65)	100	
	(25, 100)	100	
USSS	(5, 25)	100	400
	(5, 50)	100	
	(20, 25)	100	
	(20, 50)	100	
USUS	(5, 40)	100	400
	(5, 100)	100	
	(20, 40)	100	
	(20, 100)	100	
Total			10691

NOS) combinations in the train set. We add more sub-classes to the test set like unlabeled seen-size-seen-NOS (Unlabeled SSSS) referring to the instances for which we were not able to find an RS, seen-size-unseen-NOS (SSUS) referring to instances whose sizes we have seen in the train set but not the NOS, unseen-size-seen-NOS (USSS) referring to instances whose sizes are not present in the train set but the NOS are, and unseen-size-unseen-NOS (USUS) referring to instances whose neither size and nor NOS are present in the train set. A detailed description of the test set is given in Table 2.7. The statistics of the OVDR of the trained models on the sub-classes SSSS and unlabeled-SSSS are given in Table 2.8. While, statistics

of OVDR for the sub-classes SSUS, USSS and USUS are given in Table 2.9. The model architectures for the FFNN and CNN are the same as in experiment 2.

From Table 2.8, we can observe that the trained models indeed generalize well on SSSS and unlabeled-SSSS. This adheres to the generalization ability of the ML models, observed during the previous two experiments, where the model is tested on instances having the same (size, NOS) combinations as present in the training set. LR achieves the best average OVDR among all the methods, with CNN and FFNN being the second and third, respectively. On the one hand, the baselines' performance drops with an increase in the size of the instances. On the other hand, the performance of ML based models is quite robust to such an increase in the size, i.e., the average OVDR increases only slightly with the increase in size from 10 to 25.

Table 2.8 Experiment 3 OVDR statistics (in %), on the sub-classes SSSS and unlabeled-SSSS.

(size-NOS)	AVG	RND	DIST	LR	FNN	CNN
			Sub-class SSSS			
(10, 25)	8.21±1.81	11.32±11.65	5.17±4.06	0.86±0.55	1.05±0.6	0.97±0.57
(10, 50)	8.23±1.63	11.57±11.7	5.97±5.61	0.7±0.43	0.87±0.51	0.79±0.46
(15, 25)	13.27±2.2	18.2±17.42	18.12±6.35	1.0±0.56	1.42±0.77	1.25±0.73
(15, 50)	13.01±2	19.08±18.98	16.96±6.47	0.68±0.38	1.19±0.65	0.91±0.51
(25, 25)	23.33±3.39	33.19±31.99	45.05±6.06	1.25±0.6	3.55±3.97	1.77±0.97
(25, 50)	22.99±2.86	33.84±31.25	45.22±6.08	0.86±0.42	2.53±2.44	1.31±0.79
Min	0.72	-0.34	0.04	-0.56	-0.31	-0.55
Max	35.42	192.91	67.98	5.79	40.99	8.05
Avg.	13.75	19.60	19.84	0.87	1.60	1.11
Median	12.09	12.12	15.01	0.81	1.18	0.98
Std. dev.	6.27	22.31	16.56	0.52	1.93	0.73
			Sub-class unlabeled-SSSS			
(10, 25)	9.63±2.46	13.48±13.08	6.91±4.93	1.17±0.6	1.43±0.67	1.33±0.65
(15, 25)	14.31±2.46	19.56±19.92	21±7.23	1.19±0.56	1.68±0.82	1.51±0.73
(15, 50)	14±2.08	20.24±19.62	18.77±6.77	0.85±0.43	1.45±0.74	1.15±0.57
(25, 25)	24.23±3.44	33.66±31.66	47.14±6.43	1.45±0.64	4.22±4.73	1.97±0.99
(25, 50)	23.94±2.81	33.46±30.17	45.76±5.99	0.97±0.43	3.37±3.49	1.47±0.74
Min	3.42	-0.82	0.27	-0.36	-0.02	-0.11
Max	35.78	214.03	69.30	4.83	68.77	8.25
Avg.	21.36	29.77	39.14	1.19	3.27	1.65
Median	22.75	20.75	43.89	1.11	2.18	1.48
Std. dev.	5.74	29.18	14.70	0.59	3.82	0.88

From Table 2.9, we can observe that the average OVDR increases on the SSUS sub-class as compared to the sub-classes SSSS and unlabeled SSSS. The average OVDR of LR, FFNN and CNN on the sub-class unlabeled SSSS increases from 1.19, 3.27 and 1.65 to 3.57, 7.08 and 5.81 on the sub-class SSUS, respectively. This is contrary to the expected behavior as we

Table 2.9 Experiment 3 OVDR statistics (in %), on the sub-classes SSUS, USSS and USUS.

(size-NOS)	AVG	RND	DIST	LR	FNN	CNN
			Sub-class SSUS			
(10, 10)	8.68±2.24	11.19±10.25	5.44±3.29	3.21±1.48	4.03±1.66	3.89±1.57
(10, 40)	8.22±1.72	12.62±12.38	5.27±3.53	2.71±1.1	3.75±1.41	3.5±1.29
(10, 65)	8.01±1.58	11.97±12.04	3.97±3.6	2.48±0.89	3.51±1.23	3.28±1.04
(10, 100)	7.99±1.59	12.53±12.73	4.07±3.52	2.55±0.84	3.62±1.13	3.4±0.95
(15, 10)	13.49±3.07	18.45±16.16	19.48±5.68	3.84±1.68	5.81±2.12	5.08±2.15
(15, 40)	13.04±1.92	18.46±20.28	16.74±5.98	3.42±1.2	5.4±1.46	5.22±1.29
(15, 65)	12.88±1.68	20.7±19.14	18.3±5.32	3.29±1.03	5.33±1.27	5.25±1.11
(15, 100)	12.73±1.62	17.19±16.82	15.25±5.73	3.3±0.92	5.38±1.14	5.26±1.08
(25, 10)	24.29±5.14	30.58±26.59	48.49±6.67	5.35±1.5	11.26±3.34	9.1±2.28
(25, 40)	23.67±3.14	25.69±23.33	45.55±5.96	4.22±1.22	12.01±2.96	8.5±1.84
(25, 65)	23.3±2.71	34.89±32.02	44.06±6.03	4.28±1.13	12.4±2.69	8.59±1.6
(25, 100)	23.19±2.34	31.55±27.83	45.23±5.57	4.26±1.16	12.43±2.57	8.65±1.48
Min	3.11	-0.24	0.15	-0.32	0.50	0.29
Max	40.01	118.87	6.60	8.64	20.09	14.67
Avg.	14.96	20.48	22.65	3.57	7.08	5.81
Median	13.06	13.25	17.53	3.41	5.74	5.35
Std. dev.	6.94	21.71	18.02	1.45	4.13	2.65
			Sub-class USSS			
(5, 25)	1.59±0.79	3.29±3.35	21.37±10.95	24.09±5.6	26.56±6.39	57.07±10.41
(5, 50)	1.47±0.7	3.46±3.37	22.28±10.19	23.13±5.57	26.97±6.32	56.05±10.69
(20, 25)	19.22±2.78	26.7±24.79	33.72±6.3	16.5±2.76	31.72±33.55	5.92±3.69
(20, 50)	18.76±2.4	28.6±27.36	31.22±5.55	15.89±2.52	26.53±29.59	4.85±2.67
Min	-0.06	-0.22	1.63	10.79	1.14	0.81
Max	24.69	155.11	60.15	47.26	164.47	81.01
Avg.	10.26	15.51	27.15	19.90	28.45	30.97
Median	8.24	6.16	28.63	18.61	25.82	26.25
Std. dev.	8.95	22.18	10.11	5.74	22.82	26.78
			Sub-class USUS			
(5, 40)	1.49±0.69	3.05±3.27	23.48±10.32	23.65±5.72	26.75±6.33	56.43±10.52
(5, 100)	1.38±0.67	3.49±3.61	22.17±11.36	23.59±5.59	26.68±6.41	56.08±10.9
(20, 40)	18.91±2.58	24.35±20.07	32.18±5.77	15.93±2.65	30.12±32.43	5.06±3.12
(20, 100)	18.47±1.93	24.33±23.48	33.69±6.05	15.63±2.44	27.04±29.11	4.42±2.22
Min	-0.14	-0.71	0.33	10.35	1.07	0.64
Max	25.63	107.41	50.92	48.14	161.98	85.77
Avg.	10.06	13.81	27.88	19.70	27.65	30.50
Median	8.41	5.03	28.89	18.19	25.65	24.25
Std. dev.	8.80	18.81	10.09	5.87	22.21	26.94

have an aggregation over the NOS in the feature space and the sizes are already seen during the training. However, we are still 50% better than the best baseline of 14.96 achieved by AVG.

Also, we can note that the performance of the ML models deteriorates significantly as com-

pared to the results on the previous sub-classes, indicating the fact that they are not able to generalize well across unseen sizes. The ML models find it more difficult to generalize on the instances with size 5 as compared to the instances with size 20. However, baseline methods perform comparably well across smaller sized instances, helping them bring down the overall OVDR, across sizes 5 and 20. The baseline AVG performs best among all the methods on the sub-class USSS and USUS. One interesting observation is the performance of the CNN. On the one hand, it performs the worst for size 5, with an average OVDR around 56. On the other hand, it performs the best for size 20 instances, with an average OVDR around 5. This observation also suggests that we should have more unseen sizes instead of just two in these two sub-classes to better judge a method. We also, in addition to adding more sizes, need to think of fundamentally new approaches that can generalize to new sizes.

2.5.2 Stochastic Generalized Assignment Problem

Problem Formulation

We consider the stochastic generalized assignment problem S-GAP, with binary and discrete variables in the first and second stages, respectively. Let O be the set of orders that needs to be fulfilled and V be the set of vehicles. The first-stage decision is to assign orders on the selected vehicles. In the second-stage, once the time taken to complete an order o on a vehicle v is realized, the decision is to assign overtime to the selected vehicles such that all the orders are satisfied. Table 2.10 contains the notations used to formulate the S-GAP.

Table 2.10 Notations used to define an instance of the S-GAP

Data	Definition
c_{ov}	Cost of completing order o on vehicle v
o_{max}	Maximum orders satisfied by a vehicle
u_{max}	Maximum vehicles used to complete all orders
ξ_k^{ov}	Time taken to complete order o on vehicle v in scenario k
p_v	Penalty incurred for unit overtime on vehicle v
$t_v^{'}$	Time available on vehicle v
Variable	**Definition**
x_{ov}	$\begin{cases} 1, & \text{if order } o \text{ is satisfied by vehicle } v \\ 0, & \text{otherwise} \end{cases}$
u_v	$\begin{cases} 1, & \text{if vehicle } v \text{ is used} \\ 0, & \text{otherwise} \end{cases}$
y_v	Overtime on vehicle v

Finally, the S-GAP is formulated as follows:

$$\min_{x} \quad \sum_{v\in V}\sum_{o\in O} c_{ov}x_{ov} + \frac{1}{N}\sum_{k=1}^{N} Q(x_{ov}, \xi_k^{ov}) \tag{2.20}$$

$$\text{s.t.} \quad \sum_{v\in V} x_{ov} = 1 \qquad \forall o \in O \tag{2.21}$$

$$\sum_{o\in O} x_{ov} \leq o_{max} \qquad \forall v \in V \tag{2.22}$$

$$x_{ov} \leq u_v \qquad \forall v \in V, \forall o \in O \tag{2.23}$$

$$\sum_{v\in V} u_v \leq u_{max} \tag{2.24}$$

$$x_{ov}, u_v \in \{0,1\} \qquad \forall v \in V, \forall o \in O \tag{2.25}$$

Here, $Q(x_{ov}, \xi_k^{ov})$ is defined as the optimal value of the problem

$$\min_{y} \quad \sum_{v\in V} p_v y_v \tag{2.26}$$

$$\text{s.t.} \quad y_v \geq \sum_{o\in O} \xi_k^{ov} x_{ov} - t'_v \qquad \forall v \in V \tag{2.27}$$

$$y_v \geq 0 \qquad \forall v \in V \tag{2.28}$$

$$y_v = 10 z_v \qquad \forall v \in V \tag{2.29}$$

$$y_v, z_v \in \mathbb{Z}_+ \qquad \forall v \in V \tag{2.30}$$

We minimize the assignment cost of orders to vehicles and the expected overtime cost in Equation (2.20). The constraint in Equation (2.21) ensure that a given order is satisfied by only one vehicle. Equation (2.22) limits the maximum number of orders that can be satisfied by a vehicle to o_{max}. The constraint in Equation (2.23) ensures that only a selected vehicle can be used to satisfy an order. The maximum number of vehicles to be used to satisfy all orders is governed by Equation (2.24). The constraint in Equation (2.28) ensures that the total of time available and overtime on a vehicle should be greater than or equal to the time required to satisfy all orders by that vehicle. The constraints in Equation (2.29) and Equation (2.30) ensures that the overtime on a vehicle is an integer multiple of 10.

Data Generation

1. **Instance generation**: We focus on a single size for S-GAP, generating 15K unique instances with 10 vehicles, 15 jobs and 20 scenarios. We generate 15K tuples of c_{ov}, ξ^{ov}, p_v and $t^{'}$ as listed in Table 2.11 and keep the value of $k = 4$ and $u_{max} = 9$

fixed across all of them.

Table 2.11 Data sampling details for generating an instance of the S-GAP

Variable	Method
c_{ov}	$\lceil \texttt{Uniform}[0,1) * 100 \rceil$
ξ_k^{ov}	$\texttt{max}(1, a)$, where $a = \lceil b + \texttt{Uniform}[0,1) * 8 \rceil$ $b = \texttt{Uniform}[0,1) * 50 - 10$
p_v	$\texttt{Poisson}(25)$
$t^{'}$	$\texttt{Poisson}(30)$

2. **Optimal solution generation**: We set a time limit of 15 minutes and a gap limit of 0% in Gurobi to solve the 15K instances generated in the previous step; we are able to close the gap to 0% for all the instances within the time budget of 15 minutes.

3. **Representative scenario generation**: Once we know the optimal solution x_{ov}, it is trivial to construct a *naive* representative scenario, where

$$\xi^{\star}_{naive} \equiv \begin{cases} c, & \text{if } x_{ov} = 0, o \in O, v \in V \\ 0, & \text{otherwise} \end{cases}$$

for $c > c_{min}, c_{min} \in \mathbb{R}$. However, using this *naive* representative scenario as the label will make the supervised learning task challenging. In particular, because of Equation (2.21) ensuring that an order is handled by only one vehicle, we will have only $|O|$ ones in x_{ov}. This leads to an RS with $|O|$ zeros and $(|O| * |V|) - |O|$ c's. Hence, when we train a model to minimize the MSE with a naive RS as the label, the model will learn to predict values very close to c. Such predictions might not even qualify as an RS.

To make the learning task easier for the model, it is necessary to have smoother labels. An RS with many unique values from the set $\mathbb{R}$ is better instead of the one having just two, i.e., 0 and c for the learning task. To achieve this, we generate an RS $\xi^{\star}$ by taking the convex combination of the *naive* RS $\xi^{\star}_{naive}$ and average scenario $\xi_{avg} = \frac{1}{N}\sum_{k=1}^{N} \xi_k$

$$\xi^{\star} = w * \xi^{\star}_{naive} + (1 - w) * \xi_{avg}. \tag{2.31}$$

Note that $\exists w \in [0, 1]$ such that Equation (2.31) holds. A trivial choice of $w = 1$ gives us $\xi^{\star} = \xi^{\star}_{naive}$. Since we are interested in a smoother RS, we systematically lower the

value of w up to the point w_{min}, such that for $w < w_{min}$ will violate Equation (2.31), i.e., the convex combination no longer gives an RS. We do binary search to find w_{min}.

Feature Engineering

The features used to train ML models to predict an RS are as follows:

Type	Data	Size
F1	$(c_{ov}, p_v, t^{'})$	170
F2	$(c_{ov}, p_v, t^{'}, \xi_A^{ov})$	320
F3	$(c_{ov}, p_v, t^{'}, \xi_A^{ov}, \xi_{SD}^{ov})$	470
F4	$(c_{ov}, p_v, t^{'}, \xi_A^{ov}, \xi_{SD}^{ov}, \xi_{MM}^{ov})$	770

Here, ξ_A^{ov}, ξ_{SD}^{ov} and ξ_{MM}^{ov} is the average, standard deviation and minimum-maximum of ξ^{ov} across the scenario axis, respectively. Note that we do not exploit any problem specific property to construct these features. They are composed of static parameters and statistical properties of random parameters of an instance.

Results

We use a common value of $w_{min}^{\star}$ for all the instances to generate an RS. The idea behind this was to keep the learning task as simple as possible for the model. We randomly sample 1000 instances out of 15K and find w_{min}^{i} for each of the sampled instances i. We select $w_{min}^{\star}$ to be equal to the 95th percentile from the sorted list of $w_{min}^{i}, i \in [1, \ldots, 1000]$.

We also try to make the labels more robust to prediction errors. Let (x^i, y^i) be the i^{th} data point in dataset $\mathcal{D}$, $\hat{y}^i$ be the corresponding prediction and δ a positive constant. We quantify the robustness of a label y^i by the value of δ, where higher the value of δ means more robust the label, under the constraint that all the predictions $\hat{y}^i \in [y^i - \delta, y^i + \delta]$ perform equally well as the label y^i for the downstream optimization task. In other words, we want to generate labels such that the errors in predicting them will have minimum impact on the optimization metric. We also need to keep in mind that this additional robustness should not come at the cost of a harder prediction task for the model. A trivial choice of robust label is to consider the *naive* representative scenario containing only c and 0. Here, even if we predict values much less than c or greater than 0, up to some threshold, the label will still be an RS. However, the trivial RS is a bad choice for the label as already discussed. What we do instead is to add a positive offset to $w_{min}^{\star}$; the positive offset moves the current RS towards trivial-RS, making it more robust. It was interesting to observe that after the addition of an

offset to the RS, a few did not hold the condition for being an RS.

Table 2.14 contains the results of different model-feature combinations. Any row of the table gives the positive offset Δ added to $w^{\star}_{min}$, the methods name, the learning metrics (mean square error and R2) and the statistical properties like minimum, maximum, average, median and standard deviation of the OVDR across all instances in the test set. A method name is of the form XX YY, where XX and YY are the acronyms of the ML model and feature type, respectively. The model acronyms LR and FN stand for Linear Regression and Feed Forward Neural Network, respectively. We use 80% of the labeled instances in the train set and the remaining in the test set. The average OVDR corresponding to AVG and RND, is around 17% and 62%, respectively. As expected, these values are pretty consistent as the surrogate corresponding to them will not change across different offsets. Thus, the goal for the ML model should be to generate predictions which have an average OVDR of at least 17% or less.

We do hyperparameter tuning to achieve the best possible results using a FFNN. We pick the dataset which contains the F2 features and the labels have an offset Δ of 0. We fixed the batch size to 256 and used Adam optimizer with a learning rate of 10^{-3}. Let λ be the weight penalty used during optimization. After a variety of tries, we found that having a single hidden layer, with 256 neurons, weight penalty λ equal to 10^{-5}, dropout equal to 0.2 in the output layer and a batch norm in the hidden layer seems to perform the best.

The detailed results of the various configurations of the FFNN are given in Table 2.13. Apart from the λ, dropout and batch-norm, we tried changing the batch size, increasing the model capacity by adding more layers and applying the dropout in the input layer. However, all of these attempts resulted in no significant improvement over the current best setting.

Table 2.13 Hyperparameter tuning for the FFNN used in the S-GAP experiment.

λ	Dropout	Batch-Norm	Loss		
			Train	Val	Test
10^{-4}	0	No	21.76	28.45	28.32
10^{-5}	0	No	14.59	24.96	24.95
10^{-5}	0.1	No	16.1	23.9	23.92
10^{-5}	0.2	No	17.75	23.81	23.8
10^{-5}	0.3	No	19.79	24.61	24.57
10^{-5}	0	Yes	13.86	25.7	25.96
10^{-5}	0.2	Yes	17.7	23.5	**23.32**

As we already mentioned, Table 2.14 contains the performance of the different methods in

terms of the learning metrics (MSE and R^2) and the optimization metric (OVDR). Across different offset Δ values, we see (1) LR outperforms FN, for a given feature type, on both learning and optimization metrics, respectively (2) across different feature types, F3 is able to achieve the best scores on all the learning and optimization metrics, apart from the exception of method FN F3 with offset Δ equal 0.1. Here, FN with feature type F2 achieves the best average OVDR (3) LR with feature type F3 is able to achieve the best learning and optimization metrics, respectively.

We also note that the MSE is a clear indicator of the performance of the model in terms of average OVDR, i.e., for any two models, if one of them performs better than the other in terms of MSE, then its average OVDR will also be better than the other. Indeed, we see this trend among LR and FN with different feature types, and also between LR and FN, for a given offset of Δ.

The best test loss achieved by an LR for Δ equal to 0, 0.05, 0.1 and 0.2 is 20.06, 25.4, 30.88 and 42.75, respectively. The best average OVDR achieved by the same LR and feature combinations for Δ equal to 0, 0.05, 0.1 and 0.2 is 13.84%, 13.81%, 13.82% and 14.37%. We also note that the average OVDR improves with the increase in Δ from 0 to 0.05. However, increasing Δ beyond 0.05 starts deteriorating the average OVDR.

Table 2.14 Learning metrics and OVDR statistics (in %) for the S-GAP for different models

Δ	**GRB vs .**	**Loss**			R^2			**Min**	**Max**	**Avg.**	**Med.**	**Std.**
		Train	**Val**	**Test**	**Train**	**Val**	**Test**					
0	AVG							0	54.84	18.67	17.97	7.23
	RND							15.85	174.94	62.39	59.81	20.39
	LR F1	32.76	-	32.86	0.087	-	0.052	10.29	88.79	40.16	39.32	11.45
	LR F2	19.78	-	20.118	0.45	-	0.42	0	41.25	14.60	14.11	5.94
	LR F3	19.22	-	20.06	0.46	-	0.42	0	35.76	13.84	13.32	5.63
	LR F4	18.71	-	20.5	0.48	-	0.40	0.17	41.11	14.14	13.69	5.74
	FN F1	31.72	36.5	36.84	0.12	-0.04	-0.07	9.04	96.82	42.72	41.92	11.95
	FN F2	17.7	23.5	23.32	0.5	0.33	0.32	0	47.80	16.70	16.22	6.48
	FN F3	16.97	23.64	23.66	0.53	0.33	0.31	0.4	46.28	16.54	16.13	6.46
	FN F4	16.04	25.12	24.88	0.55	0.29	0.28	0	49.03	18.73	18.16	7.01
0.05	AVG							0	52.68	18.85	18.25	07.25
	RND							9.14	165.19	62.90	60.62	20.50
	LR F1	37.56	-	38.96	0.09	-	0.06	9.28	90.32	40.58	39.71	11.60
	LR F2	23.9	-	25.5	0.42	-	0.38	0	43.27	14.76	14.42	06.02
	LR F3	23.17	-	25.4	0.42	-	0.38	0	41.89	13.81	13.48	05.62
	LR F4	22.05	-	26.12	0.45	-	0.37	0	38.54	14.15	13.71	05.70
	FN F1	32.3	42.23	42.69	0.2	-0.03	-0.04	12.64	91.70	43.65	42.69	12.08
	FN F2	21.58	28.08	28.87	0.46	0.3	0.29	0	44.41	17.46	16.80	6.67
	FN F3	20.5	28.08	28.9	0.49	0.31	0.29	1.24	46.93	17.10	16.58	6.70
	FN F4	19.48	29.39	30.59	0.51	0.28	0.25	0	52.05	18.74	18.23	7.07
0.1	AVG							0	58.34	18.73	18.10	7.22
	RND							12.6	173.26	62.16	60.05	19.87
	LR F1	43.07	-	45.21	0.09	-	0.06	11	90.63	40.75	39.77	11.57
	LR F2	28.73	-	31.14	0.39	-	0.35	0.7	38.58	14.97	14.67	05.77
	LR F3	27.81	-	30.88	0.41	-	0.36	0	40.50	13.82	13.49	05.57
	LR F4	27.05	-	31.7	0.43	-	0.34	0	43.15	14.20	13.76	05.73
	FN F1	42.18	47.86	50	0.1	-0.02	-0.05	9.41	96.86	44.35	43.19	12.36
	FN F2	25	32.82	33.4	0.47	0.29	0.3	0.44	50.62	16.75	16.16	6.69
	FN F3	24	33.16	34.45	0.49	0.28	0.28	0	45.18	17.29	16.87	6.70
	FN F4	21.74	35.51	37.83	0.53	0.23	0.2	0.82	59.32	19.97	19.33	7.42
0.2	AVG							1.00	58.34	18.6	17.97	7.05
	RND							16.15	173.54	62.17	59.44	20.38
	LR F1	56.33	-	58.35	0.09	-	0.06	10.65	94.37	42.29	41.64	12.16
	LR F2	40.47	-	43.1	0.35	-	0.30	0	45.71	15.67	15.14	6.20
	LR F3	39.06	-	42.75	0.37	-	0.31	0	41.12	14.37	13.95	5.79
	LR F4	38	-	43.87	0.385	-	0.29	0.23	43.25	14.90	14.49	5.97
	FN F1	52.67	62.65	63.76	0.12	-0.02	-0.03	7.09	99.04	45.58	44.69	12.67
	FN F2	35.17	44.58	46.1	0.4	0.27	0.25	1.28	46.63	18.35	17.95	6.96
	FN F3	32.79	44.6	45.54	0.43	0.27	0.26	0.05	56.25	17.75	17.23	6.89
	FN F4	31.16	47.1	48.78	0.46	0.23	0.2	0.00	53.53	19.80	19.36	7.24

CHAPTER 3 CONCLUSION AND RECOMMENDATIONS

We presented an ML based heuristic to quickly compute good primal solutions to two-stage stochastic integer programs. We tested our approach on two problems, namely, stochastic capacitated facility location problem (S-CFLP) and stochastic generalized assignment problem (S-GAP). We extracted statistical and customized features for both S-CFLP and S-GAP. For the S-CFLP, we did three experiments; first, we trained on single size and tested on the same size; second, we trained on multiple sizes and tested on those same set of sizes; finally, we trained on some specific sizes and tested on previously seen sizes and unseen sizes. The results of the first two experiments were positive, with LR achieving solutions at par with Gurobi while taking only a fraction of its time. For the third experiment, we observe positive results for seen sizes. However, for unseen sizes, the results were not good. This highlights the fact that we may need better parameter sharing to achieve good generalization on previously unseen sizes during training. The experiments conducted also gave us a first-hand experience of how the learning metric like the MSE might not be directly proportional to the optimization metric like the average OVDR, i.e., achieving a good performance in terms of the learning metric does not guaranty an improvement in the average OVDR. During the initial experimentation, we spent quite some time trying to improve the MSE. It was only when we observed the average OVDR, we became aware of this discrepancy.

For the S-GAP problem, we reported an ablation study with four different types of features and two different types of models (LR and FFNN). We also tried to generate robust labels, such that the error in the predictions does not affect the performance in the downstream optimization task. However, we were not able to achieve good performance in terms of the optimization metric of the average OVDR. We are still in the phase of deciding the right complexity that the ML models can handle given the compute limitation. At this stage, we believe that either the problem complexity is too high or we do not have the right set of features that can help us achieve good performance in terms of the average OVDR.

We list two potential causes that might be making the prediction task difficult for the ML. First, the second-stage costs were different for each instance in the S-GAP experiments, as opposed to the second-stage costs being fixed for each instance, of the same size, in the S-CFLP experiments. Second, the S-GAP had an output dimension of 150, whereas in the S-CFLP the maximum output dimension was 25. Such differences can increase the problem complexity and need further investigation.

Interestingly enough, and in contrast to the observation made during the S-CFLP experi-

mentation, for the S-GAP, we observed that achieving a good MSE was a clear indicator of the good performance in terms of the average OVDR. LR, in this problem, achieved the best MSE and the best average OVDR. Also, a similar trend is observed among the FFNN based models. This is contrary to the observation of S-CFLP, where obtaining a good performance on the learning metric had no bearing on the performance of the optimization metric. This underscores the need for a better loss function for training the ML models that can give a good estimate of the model performance on the downstream optimization task.

We would also like to focus on the problem classes for which the value of the stochastic solution is larger as compared to the first-stage solution of the surrogate with average scenario. Such problem classes would be an ideal place to apply our methodology as finding a good first-stage decision traditionally requires solving a big formulation. On the other hand, we can find a good first-stage decision quickly by solving a surrogate formulation, which is significantly smaller, formed using an RS.

REFERENCES

[1] W. B. Powell, "Clearing the jungle of stochastic optimization," in *Bridging data and decisions*. Informs, 2014, pp. 109–137.

[2] G. B. Dantzig, "Programming in a linear structure," in *Bulletin of the American Mathematical Society*, vol. 54, no. 11. AMER MATHEMATICAL SOC 201 CHARLES ST, PROVIDENCE, RI 02940-2213, 1948, pp. 1074–1074.

[3] ——, "Origins of the simplex method," in *A history of scientific computing*, 1990, pp. 141–151.

[4] ——, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[5] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[6] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. pp. 497–520, 1960.

[7] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, vol. 64, no. 5, p. 275–279, 1958. [Online]. Available: https://www.ams.org/journals/bull/1958-64-05/S0002-9904-1958-10224-4/home.html

[8] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Operations Research Letters*, vol. 6, no. 1, pp. 1–7, 1987.

[9] M. Conforti, G. Cornuéjols, G. Zambelli *et al.*, *Integer programming*. Springer, 2014, vol. 271.

[10] S. Herculano-Houzel, "The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost," *Proceedings of the National Academy of Sciences*, vol. 109, no. Supplement 1, pp. 10 661–10 668, 2012.

[11] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady an ussr*, vol. 269, 1983, pp. 543–547.

[14] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[16] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[17] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15 849–15 854, 2019.

[18] S. Ruder, "An overview of gradient descent optimization algorithms," Mar 2020. [Online]. Available: https://ruder.io/optimizing-gradient-descent/

[19] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization*. Springer, 1982, pp. 762–770.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] Y. Bengio, E. Frejinger, A. Lodi, R. Patel, and S. Sankaranarayanan, "A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, E. Hebrard and N. Musliu, Eds. Cham: Springer International Publishing, 2020, pp. 99–111.

[24] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[25] G. Lulli and S. Sen, "A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems," *Management Science*, vol. 50, no. 6, pp. 786–796, 2004.

[26] F. V. Louveaux and D. Peeters, "A dual-based procedure for stochastic facility location," *Operations research*, vol. 40, no. 3, pp. 564–573, 1992.

[27] P. Kall, S. W. Wallace, and P. Kall, *Stochastic programming*. Springer, 1994.

[28] A. Prékopa, *Stochastic programming*. Springer Science & Business Media, 2013, vol. 324.

[29] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on stochastic programming: modeling and theory*. SIAM, 2009.

[30] J. Dupačová, N. Gröwe-Kuska, and W. Römisch, "Scenario reduction in stochastic programming," *Mathematical programming*, vol. 95, no. 3, pp. 493–511, 2003.

[31] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[32] J. Linderoth, A. Shapiro, and S. Wright, "The empirical behavior of sampling methods for stochastic programming," *Annals of Operations Research*, vol. 142, no. 1, pp. 215–241, 2006.

[33] W. B. Powell and S. Meisel, "Tutorial on stochastic optimization in energy—part i: Modeling and policies," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 1459–1467, 2015.

[34] ——, "Tutorial on stochastic optimization in energy—part ii: An energy storage illustration," *IEEE Transactions on Power Systems*, vol. 31, no. 2, pp. 1468–1475, 2015.

[35] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro, "A stochastic programming approach for supply chain network design under uncertainty," *European Journal of Operational Research*, vol. 167, no. 1, pp. 96–115, 2005.

[36] S. Ahmed, M. Tawarmalani, and N. V. Sahinidis, "A finite branch-and-bound algorithm for two-stage stochastic integer programs," *Mathematical Programming*, vol. 100, no. 2, pp. 355–377, 2004.

[37] S. Sen and J. L. Higle, "The c 3 theorem and a d 2 algorithm for large scale stochastic mixed-integer programming: set convexification," *Mathematical Programming*, vol. 104, no. 1, pp. 1–20, 2005.

[38] S. Ahmed, "A scenario decomposition algorithm for 0–1 stochastic programs," *Operations Research Letters*, vol. 41, no. 6, pp. 565–569, 2013.

[39] S. Sen, "Stochastic mixed-integer programming algorithms: Beyond benders' decomposition," *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[40] C. C. Carøe and J. Tind, "L-shaped decomposition of two-stage stochastic programs with integer recourse," *Mathematical Programming*, vol. 83, no. 1-3, pp. 451–464, 1998.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[42] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[43] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[44] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *arXiv preprint arXiv:1811.06128*, 2018.

[45] A. Lodi and G. Zarpellon, "On learning and branching: a survey," *Top*, vol. 25, no. 2, pp. 207–236, 2017.

[46] B. Defourny, D. Ernst, and L. Wehenkel, "Scenario trees and policy selection for multistage stochastic programming using machine learning," *INFORMS Journal on Computing*, vol. 25, no. 3, pp. 488–501, 2013.

[47] G. Murgia and S. Sbrilli, "Integrating multi-stage stochastic programming and machine learning for the evaluation of policies in the electricity portfolio problem," *IMA Journal of Management Mathematics*, vol. 28, no. 1, pp. 109–130, 2017.

[48] B. Defourny, D. Ernst, and L. Wehenkel, "Multistage stochastic programming: A scenario tree based approach to planning under uncertainty," in *Decision theory models for applications in artificial intelligence: concepts and solutions*. IGI Global, 2012, pp. 97–143.

[49] P. Donti, B. Amos, and J. Z. Kolter, "Task-based end-to-end model learning in stochastic optimization," in *Advances in Neural Information Processing Systems*, 2017, pp. 5484–5494.

[50] S. Prestwich, R. Rossi, and A. Tarim, "Stochastic constraint programming as reinforcement learning," *arXiv preprint arXiv:1704.07183*, 2017.

[51] H. Jia and S. Shen, "Benders cut classification via support vector machines for solving two-stage stochastic programs," *arXiv preprint arXiv:1906.05994*, 2019.

[52] V. Nair, D. Dvijotham, I. Dunning, and O. Vinyals, "Learning fast optimizers for contextual stochastic integer programs," in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, A. Globerson and R. Silva, Eds. AUAI Press, 2018, pp. 591–600. [Online]. Available: http://auai.org/uai2018/proceedings/papers/217.pdf

[53] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig, "The SCIP Optimization Suite 6.0," Optimization Online, Technical Report, July 2018. [Online]. Available: http://www.optimization-online.org/DB_HTML/2018/07/6692.html

[54] ——, "The SCIP Optimization Suite 6.0," Zuse Institute Berlin, ZIB-Report 18-26, July 2018. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0297-zib-69361

[55] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[56] M. B. Editor, "Regression analysis: How do i interpret r-squared and assess the goodness-of-fit?" [Online]. Available: https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

[57] P. Bonami, A. Lodi, and G. Zarpellon, "Learning a classification of mixed-integer quadratic programming problems," in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2018, pp. 595–604.

[58] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *arXiv preprint arXiv:1906.01629*, 2019.