UNIVERSITÉ SORBONNE PARIS NORD
LABORATOIRE D'INFORMATIQUE DE PARIS NORD - LIPN
ÉQUIPE: ALGORITHMES ET OPTIMISATION COMBINATOIRE - AOC

Machine Learning for Lagrangian Relaxation

THÈSE DE DOCTORAT
PRÉSENTÉE PAR

Francesco DEMELAS

À L'ÉCOLE DOCTORALE GALILÉE

POUR OBTENIR LE GRADE DE
DOCTEUR EN INFORMATIQUE

SOUTENUE À VILLETANEUSE LE 09/07/2025
DEVANT LE JURY COMPOSÉ DE:

Céline ROUVEIROL	Université Sorbonne Paris Nord	PRÉSIDENTE DE JURY
Bissan GHADDAR	Technical University of Denmark	RAPPORTEUSE
Alberto CESELLI	University of Milan	RAPPORTEUR
Caio Filippo CORRO	INSA de Rennes	EXAMINATEUR
Emiliano TRAVERSI	ESSEC Business School France	EXAMINATEUR
Joseph LE ROUX	Université Sorbonne Paris Nord	DIRECTEUR DE THÈSE
Antonio FRANGIONI	Università di Pisa	DIRECTEUR DE THÈSE
Mathieu LACROIX	Université Sorbonne Paris Nord	ENCADRANT DE THÈSE

Abstract

In this thesis, we study the application of machine learning to Lagrangian relaxation. The latter involves dualizing a subset of constraints that define the feasible region, incorporating them as a penalization term in the objective function. For each choice of penalization vector, called the Lagrangian multipliers vector, we obtain a different bound on the objective value of the original problem. The tightest bound is obtained by solving the Lagrangian Dual problem, typically addressed using iterative algorithms such as the Bundle method.

We first explore an amortized optimization method that directly predicts a Lagrangian multipliers vector. A probabilistic Graph Neural Network encoder maps the instance, represented as a bipartite graph, to a latent space, providing a representation of dualized constraints. From this representation, a deterministic decoder predicts a Lagrangian multiplier for each dualized constraint. We train this neural model in an unsupervised learning way, by directly maximizing the Lagrangian bound obtained by solving the Lagrangian subproblem using the predicted multipliers. We show that our prediction may be used to replace or initialize the Bundle method.

While initialization is important, it is not the only factor affecting the performance of the Bundle method. We present a machine learning–based technique for dynamically tuning, at each iteration, the Bundle method parameter that serves as both step size and regularization term. We still train our model to maximize the Lagrangian bound, and we show how to approximate the gradient of the algorithm's execution using unrolling techniques and analyzing the solution structure of the quadratic problem solved at each iteration. This enables us to train our model using standard back-propagation techniques. However, the gradient approximation currently lacks sufficient expressiveness, which limits performance. Finally, we propose replacing the quadratic problem that must be solved at each iteration to obtain a search direction with a learned surrogate model. The learning model emulates the behaviors of the quadratic problem using a recurrent network combined with an attention mechanism to predict a convex combination of the subgradients stored during the bundle execution. Furthermore, by smoothing the stabilization point updates, we obtain a fully differentiable framework that can be trained using automatic differentiation.

The last approach goes beyond the traditional Lagrangian relaxation framework. Indeed, the final method can be interpreted as a trainable optimizer, opening the way for applications in meta-learning and other advanced optimization tasks.

Keywords: Lagrangian Relaxation, End-to-End Optimization, Amortization, Unrolling, Bundle Method.

Résumé

Dans cette thèse, nous étudions l'application de l'apprentissage automatique à la relaxation lagrangienne. Cette technique consiste à dualiser un sous-ensemble de contraintes définissant la région admissible, en les intégrant sous forme de pénalisation dans la fonction objective. Pour chaque vecteur de pénalisation, appelé vecteur de multiplicateurs de Lagrange, on obtient une borne différente de la valeur objective du problème initial. La meilleur borne est obtenue en résolvant le dual Lagrangien, généralement résolu avec un algorithme itératif tel que la méthode des faisceaux (Bundle method).

Nous proposons d'abord une méthode d'optimisation amortie qui prédit un vecteur de multiplicateurs de Lagrange. Un encodeur probabiliste, basé sur un réseau de neurones graphiques, projette l'instance, représentée comme un graphe biparti, dans un espace latent fournissant une représentation des contraintes dualisées. Un décodeur déterministe prédit ensuite un multiplicateur pour chaque contrainte dualisée. Ce modèle est entraîné de manière non supervisée, en maximisant directement la borne lagrangienne obtenue avec les multiplicateurs prédits. Nous montrons que ces prédictions permettent de remplacer ou d'initialiser la méthode des faisceaux.

Bien que l'initialisation soit un facteur important, elle n'est pas la seule influençant les performances de la méthode des faisceaux. Nous introduisons un modèle d'apprentissage automatique permettant d'ajuster dynamiquement, à chaque itération, le paramètre de la méthode des faisceaux qui définit le pas et le poids de la régularisation. Le modèle est toujours entraîné pour maximiser la borne lagrangienne, en approximant le gradient par une méthode d'unrolling et en analysant la structure de la solution du problème quadratique. Cela permet un entraînement par rétropropagation. Cependant, l'approximation du gradient manque encore d'expressivité. Enfin, nous proposons de remplacer le problème quadratique déterminant la nouvelle direction de recherche par un modèle d'apprentissage automatique. Ce modèle imite le comportement de la méthode des faisceaux: à l'aide d'un réseau récurrent associé à un mécanisme d'attention, il prédit une combinaison convexe des sous-gradients obtenus pendant l'exécution de la méthode. En lissant les mises à jour du point de stabilisation, nous obtenons un modèle entièrement différentiable et entraînable par différentiation automatique.

La dernière approche va au-delà du cadre classique de la relaxation lagrangienne. En particulier, elle peut être interprétée comme un optimiseur entraînable, ouvrant la voie à des applications en méta-apprentissage ainsi qu'à d'autres problèmes d'optimisation avancés.

Mots-clés : Relaxation Lagrangienne, Optimisation End-to-End, Optimisation Amortie, Unrolling, Méthode des Faisceaux.

Acknowledgements

A thesis is undoubtedly a collective effort, and without all of you, this work would not exist.

I wish to express my gratitude to my thesis supervisors, Antonio Frangioni, Joseph Le Roux, and Mathieu Lacroix, for their scientific and human support during these long and demanding years. I consider myself fortunate to have had you as supervisors. Thank you for guiding me through my academic development and for providing me with essential tools for a research career.

I would like to thank Bissan Ghaddar and Alberto Ceselli for their willingness and patience in reviewing this thesis. More broadly, I would also like to thank all the other members of the jury: Caio Filippo Corro, Céline Rouveirol, Emiliano Traversi, and Roberto Wolfler Calvo.

Finally, I would like to thank all my friends, my family, and my colleagues at LIPN.

Remerciements

Une thèse est sans aucun doute un travail collectif, et sans vous tous, ce travail n'existerait pas.

Je souhaite tout d'abord exprimer ma gratitude à mes directeurs de thèse, Antonio Frangioni, Joseph Le Roux et Mathieu Lacroix, pour leur soutien scientifique et humain durant ces années longues et exigeantes. Je me considère chanceuse de vous avoir eus comme encadrants. Merci de m'avoir accompagnée dans mon parcours académique et de m'avoir transmis des outils essentiels pour le métier de chercheuse.

Je remercie Bissan Ghaddar et Alberto Ceselli pour leur disponibilité et leur patience dans la relecture de cette thèse. Plus largement, je tiens aussi à remercier les autres membres du jury : Caio Filippo Corro, Céline Rouveirol, Emiliano Traversi et Roberto Wolfler Calvo.

Enfin, merci à toutes mes amies et tous mes amis, à ma famille, ainsi qu'à mes collègues du LIPN.

Ringraziamenti

Una tesi è senza dubbio un lavoro collettivo e senza tutti voi, questa tesi non esisterebbe.

Innanzitutto desidero esprimere la mia gratitudine ai miei supervisori di tesi Antonio Frangioni, Joseph Le Roux e Mathieu Lacroix per il sostegno scientifico e umano durante questi anni lunghi e impegnativi. Mi ritengo fortuna ad avervi avuto come supervisori di tesi. Grazie per avermi accompagnata nella crescita accademica e avermi fornito degli strumenti essenziali per il mestiere della ricerca.

Ringrazio Bissan Ghaddar e Alberto Ceselli per aver avuto avuto la disponibilità e la pazienza di rileggere questa Tesi. Più largamente voglio ringraziare anche tutti gli altri membri della giuria: Caio Filippo Corro, Céline Rouveirol, Emiliano Traversi e Roberto Wolfler Calvo.

Un ringraziamento a tutte le mie amiche ed i miei amici, alla mia famiglia ed alle colleghe e colleghi del LIPN.

Contents

Introduction

Solving optimization problems lies at the heart of many real-world applications and is a major area of research. However, problems like Mixed Integer Linear Programs (MILPs) are, in general, computationally intractable. To tackle their complexity, relaxation techniques have been widely adopted. These methods aim to simplify the original problem to be sufficiently easy to solve. The optimal objective value of such relaxations provides a bound on the feasible solutions of the *original* problem, which enables an estimation of their quality.

Lagrangian relaxation consists of removing a set of constraints from the feasible region, while still incorporating them into the objective function through a penalty term. The penalty coefficients are known as Lagrangian multipliers. By dualizing constraints that connect otherwise independent components, the original problem can be decomposed into smaller, more manageable subproblems. Each of these subproblems is solved iteratively with different values for the Lagrangian multipliers. The task of determining the optimal Lagrangian multipliers is referred to as the Lagrangian Dual problem.

Despite their effectiveness, traditional methods for solving the Lagrangian Dual problem, such as the Subgradient method or the Bundle method, may suffer from slow convergence and sensitivity of parameters. Recently, machine learning has emerged as a powerful tool, capable of learning heuristics from data to accelerate classical optimization techniques. This has opened a new direction for developing data-driven optimization methods, particularly in scenarios where similar problem instances are solved repeatedly and prior experience can be exploited.

In this work, we investigate how machine learning models can be leveraged to improve the resolution of the Lagrangian Dual. Our goal is to design learning-based strategies that either enhance classical solvers or replace some of their components, thereby accelerating convergence or allowing the rapid computation of high-quality bounds.

Beyond improving computational efficiency, the proposed approaches contribute to a broader understanding of how optimization algorithms can be enhanced by learning, and how learning techniques can be embedded into traditional frameworks.

Overview of the document

In Chapter 1, we provide the necessary background in the field of Lagrangian relaxation. In particular, we give a detailed presentation of Lagrangian relaxation and ascent

methods used to solve it. We specifically cover the Subgradient method, the Cutting Plane method, and the Bundle method. Among these, we place greater emphasis on the Bundle method, as it serves as the core resolution technique in our proposed approaches.

In Chapter 2, we present the foundational concepts of machine learning. We start by defining the notion of a learning problem and how it can be addressed. Then, we present the types of models considered in our work.

Chapter 3 offers a review of the literature at the intersection of machine learning and optimization. We emphasize applications of machine learning to decomposition techniques, with particular attention to approaches involving Lagrangian relaxation. A special focus is given to amortized optimization strategies, which aim to replace iterative resolution processes with learned models. Challenges inherent to this domain are also highlighted.

The following two chapters present the contributions of this thesis. Chapter 4 introduces a method that leverages the Continuous relaxation to obtain tighter bounds by predicting Lagrangian multipliers. Specifically, we model the optimization instance using a Bipartite Graph, where each node represents a component of the problem. We then apply a Graph Neural Network (GNN) encoder to iteratively refine the node embeddings through message-passing, in order to obtain a hidden representation for each dualized constraint. This hidden representation of dualized constraints is used to predict a scalar value for each dualized constraint, corresponding to the predicted Lagrangian multiplier. This approach is an unsupervised learning approach as the model is trained to directly maximize the Lagrangian bound based on these predicted multipliers, without requiring knowledge of the optimal Lagrangian multipliers.

However, this method is limited to cases where an informative Lagrangian multiplier vector can be identified easily, even if it remains far from the optimal solution of the Lagrangian Dual. In Chapter 4, we obtain such a vector by leveraging the values associated with the dualized constraints in a dual optimal solution to the Continuous relaxation. This approach assumes that the Continuous relaxation is easy to solve but yields weaker bounds than the Lagrangian Dual. Additionally, the bipartite graph representation used may become intractable as the instance size grows.

To overcome these limitations, we propose alternative methods that incorporate machine learning inside the Bundle method. In Chapter 5, we start by proposing a learning–based mechanism for dynamically tuning, at each iteration, the regularization parameter of the Bundle method. The model is trained to maximize the Lagrangian bound obtained by considering the Lagrangian multiplier vectors obtained during the bundle execution. We demonstrate how to approximate the gradient of the Bundle method using unrolling techniques and a characterization of the solution to the quadratic problem solved at each iteration of the Bundle method. This enables us to train our model considering standard back-propagation techniques. However, the current gradient approximation lacks sufficient expressiveness, which sometimes limits its performance.

We then propose a second method, replacing the quadratic problem that must be solved at each iteration of the Bundle method to obtain a search direction with a learned surrogate model. The learning model emulates the behaviors of the quadratic problem using a recurrent network combined with an attention mechanism to predict a convex combination of the subgradients stored during the bundle execution. Furthermore, by smoothing the stabilization point updates, we obtain a fully differentiable framework

that can be trained using automatic differentiation.

Finally, Chapter 6 concludes the thesis by summarizing our contributions and outlining promising directions for future research.

Main Contributions

The main contributions of this thesis are developed in Chapter 4 and Chapter 5. These two chapters explore different ways of using machine learning to assist or replace the resolution Lagrangian Dual problem. We did not use machine learning as *black-box*, but we integrated domain knowledge into the learning process.

Both chapters present approaches that handle instances of varying sizes. While this property is often neglected in the literature, it is important in real-world optimization problems, where the structure and scale of instances may vary considerably. The majority of the existing approaches predicting Lagrangian multipliers either rely on fixed-size input models or extract problem-specific features, which limits the generalization capability of the resulting models. In contrast, both contributions in this thesis emphasize generalizable architectures capable of operating directly on instances, effectively leveraging their inherent combinatorial structure.

Chapter 4 leverages a structured representation of the optimization instance that can be passed as input to a neural network, while preserving sufficient information to reconstruct a valid output solution. The instance is encoded as a bipartite graph, a representation previously used in the literature, though for different purposes. This structure enables the model to propagate information across related components of the problem, rather than treating them as independent elements. However, this design choice introduces a limitation, as discussed in Section 4.1.2. Indeed, the size of the graph grows with the number of variables and constraints, making it less suitable for large-scale problems due to the computational cost of graph convolutions. Furthermore, we show that even partial leveraging on classical optimization results, such as the solution of the Continuous relaxation, can be useful to provide predictions. This insight opens the door to future generative approaches, where different types of information, obtained through the resolution of an optimization problem, could guide the learning process. Chapter 4 makes another novel contribution, as it casts the prediction of a Lagrangian multiplier vector as an energy-based modeling task, drawing a connection with a well-established body of work in the machine learning literature.

Chapter 5 follows a different idea. Rather than designing a model to predict a solution from scratch, it integrates machine learning within the Bundle method, a resolution algorithm of the Lagrangian Dual. Specifically, the focus is on modifying the structure of the Bundle method, inserting learned components into its iterative execution. This results in a machine learning-based framework that is differentiable end-to-end. This is the first work, to our knowledge, that differentiates through the execution of the Bundle method to enable direct optimization of the final objective value. Some previous studies have explored similar ideas using Subgradient methods, but the Bundle method has a more complex structure that poses further challenges. We also demonstrate how certain elements of the algorithm can be reinterpreted or replaced by neural networks models, yielding a hybrid approach fully trainable using automatic differentiation techniques. By learning the network parameters, we obtain a

learned optimizer that demonstrates promising generalization capabilities, both when performing more iterations and when applied to datasets not seen during training. This approach also addresses a key limitation of the bipartite graph representation introduced in Chapter 4, as it operates in a lower-dimensional feature space. As a result, it scales more effectively to larger instances. Furthermore, it represents a meaningful step toward the development of generative models that compute solutions to optimization problems by following the structure of the algorithms used to solve them. This opens up promising avenues for applying similar strategies to other classes of problems, where algorithm-aware learning can offer both interpretability and efficiency.

Chapter 1

Background on Lagrangian Relaxation

Mathematical optimization is a branch of mathematics that focuses on solving optimization problems and selecting the best element from a set of alternatives based on specific criteria. This chapter provides some fundamental concepts about Mathematical optimization, emphasizing their application in achieving the main results of this thesis. Specifically, the findings discussed in Chapter 4, and partially in Chapter 5, focus on Lagrangian relaxation.

Lagrangian relaxation is one of the most efficient approaches to solving Mixed Integer Linear Programs (MILPs) [263] with complex constraints. Given a penalization vector for these constraints, which are called Lagrangian multipliers (LMs), it yields a bound on the optimal value of the MILP. The Lagrangian Dual problem aims to identify the LMs that offer the tightest bound.

In Section 1.1, we begin by presenting the mathematical formulation of an MILP and the formulation of the corresponding Lagrangian relaxation. We will then show that Lagrangian relaxation can be seen as a Partial Convex relaxation of the problem, and in particular, this enables us to understand better the relationship between Lagrangian relaxation and Continuous relaxation, which serves as the foundation for the model discussed in Chapter 4.

The challenge of determining the best bound provided by the Lagrangian relaxation can be modeled as a concave optimization problem with a piece-wise linear objective function. In Section 1.3, we discuss some algorithms that can be used for concave maximization (and convex minimization) problems.

MILPs are NP-hard problems [245]. The complexity sometimes arises from a subset of constraints that link together otherwise independent subproblems. More formally [54,

Chap. 8], let P be a MILP of the form:

$$(P) \qquad \min_{\boldsymbol{x}} \ \boldsymbol{w}^\top \boldsymbol{x} \tag{1.1a}$$
$$\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{1.1b}$$
$$\boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d} \tag{1.1c}$$
$$\boldsymbol{x} \in \mathbb{R}_+^{n_1} \times \mathbb{N}^{n_2} \tag{1.1d}$$

The Branch&Bound (B&B) method is a key technique in solving MILPs. B&B, initially introduced in [149] in 1960, is a general algorithm that solves a problem by splitting it into several smaller subproblems. This process is repeated recursively on the generated subproblems until they are simplified enough to be solved. The progression of the procedure can be effectively visualized and described using a tree structure.

At each iteration, an unsolved subproblem is selected based on a predefined *node selection* strategy. Each subproblem corresponds to a leaf node in the current execution-state tree. Subsequently, both a lower and an upper bound on the optimal objective function are computed. The lower bound is generally obtained by solving a cheaper relaxation of the associated problem. Instead, the upper bound is given by the best-known feasible solution (incumbent), which may be improved using heuristics.

If the lower bound at the current node is greater than the best incumbent, the sub-tree rooted at that node is pruned, as it cannot lead to an improved solution. When the difference between the global lower bound and the objective value of the best incumbent is smaller than a user-defined threshold, the process terminates with an $\epsilon-$optimal solution, with an approximation quality that depends on the threshold. Otherwise, the search continues by branching on the feasible region of the selected subproblem. If the difference between the bounds at the current node exceeds the threshold, the search continues by branching on the feasible region of the current subproblem. This region is divided into two or more subsets, typically by imposing additional constraints on one or more variables, following a *variable selection* strategy.

A common relaxation used to obtain a lower bound is the Continuous relaxation (CR), which removes the integrity constraints, transforming constraints (1.1d) in $\boldsymbol{x} \in \mathbb{R}_+^{n_1+n_2}$. However, this is just one possibility. Another widely used relaxation is the Lagrangian relaxation, which relaxes selected linear constraints instead of integrality constraints. The Lagrangian relaxation is described in more detail in Section 1.1.

1.1 Lagrangian Relaxation

The *Lagrangian subproblem* is obtained by dualizing a family of constraints. Here we relax the constraints defined by the inequalities (1.1b), and penalize their violation with Lagrangian multipliers (LMs) $\boldsymbol{\pi} \in \mathbb{R}_+^m$:

$$\big(LR(\boldsymbol{\pi})\big) \qquad \min_{\boldsymbol{x}} \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{\pi}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})$$
$$\boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d}$$
$$\boldsymbol{x} \in \mathbb{R}_+^{n_1} \times \mathbb{N}^{n_2}$$

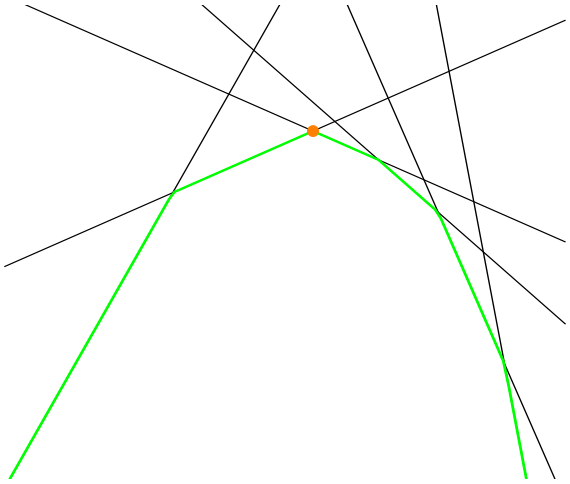

Figure 1.1: Example of Lagrangian function for a one-dimensional Lagrangian multipliers space. Each black line represents a cut, which can also be seen as the objective function of the Lagrangian subproblem for a fixed primal solution $\boldsymbol{x}^*$ as a function of the Lagrangian multiplier. The green curve represents the Lagrangian function, obtained by considering the minimum of the former function (in black) for each fixed Lagrangian multiplier. The orange point marks the optimal solution of the Lagrangian Dual.

Weak Lagrangian duality ensures that $LR(\boldsymbol{\pi})$ provides a lower bound for P. The goal of the Lagrangian Dual problem is to determine the optimal bound.

$$(LD) \qquad \max_{\boldsymbol{\pi} \geq \mathbf{0}} LR(\boldsymbol{\pi}).$$

Lagrangian relaxation becomes particularly valuable when removing certain inequalities results in a Lagrangian subproblem $LR(\boldsymbol{\pi})$ that can be solved efficiently. It is crucial to select the set of constraints to dualize carefully. Indeed, this choice involves a trade-off between the speed and the quality of the resulting bound, as will be discussed later.

In general, there is a *No Free Lunch Theorem*, as discussed in the paragraph on the integrality property, saying that to achieve better bounds than the Continuous relaxation, the Lagrangian subproblem should be "harder" than a Linear problem. We will see that if the Lagrangian subproblem is equivalent to its Continuous relaxation for all the Lagrangian multipliers vectors, then the Lagrangian Dual provides the same bound as that of the Continuous relaxation.

1.1.1 Lagrangian Relaxation as (Partial) Convex Relaxation

It is well established that the Lagrangian Dual(LD) is equivalent to the Convex relaxationof (1.1a)-(1.1d) obtained considering the following convexification of the *easy* constraints

$$(\tilde{P}) \quad \min \left\{ \boldsymbol{c}^\top \boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\,,\ \boldsymbol{x} \in \mathrm{conv}(X) \right\} \tag{1.2}$$

where $X = \{\boldsymbol{x} \in \mathbb{R}^{n_1}_+ \times \mathbb{N}^{n_2} \mid \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d}\}$ and $\mathrm{conv}(X)$ denotes the convex hull of X.

Geoffrion's theorem [94] ensures that the Lagrangian Dual provides valuable information as much as a Continuous relaxation and possibly more. In practice, the bound provided by the Lagrangian relaxation is strictly better than that of the Continuous relaxation on various applications.

Theorem 1.1.1. *The Lagrangian Dual is equivalent to the linear dual of the Partial Convex relaxation $(\tilde{P})$, that is*

$$LD = \tilde{D}$$

Proof. It is a well-known fact that, if all the entries of $\boldsymbol{C}$ and $\boldsymbol{d}$ are rational, then $\mathrm{conv}(X)$ is a convex polytope. This is because, in this case, all the vertices of the polytope defined by the linear system $\boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d}$ will be rational, as they can be described as a solution to a linear system obtained by considering only a subset of rows satisfying the equality constraints.

Consequently, the points in $\mathrm{conv}(X)$ can be expressed as a convex combination of these rational vertices. This means that we can find a matrix $\tilde{\boldsymbol{C}}$ and a vector $\tilde{\boldsymbol{d}}$ such that $\mathrm{conv}(X) = \{\boldsymbol{x} \in \mathbb{R}^{n_1+n_2}, \tilde{\boldsymbol{C}}\boldsymbol{x} \geq \tilde{\boldsymbol{d}}\}$.

Hence, we can rewrite $(\tilde{P})$ as

$$\min_{\boldsymbol{\pi},\boldsymbol{\eta}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} \,|\, \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b},\, \tilde{\boldsymbol{C}}\boldsymbol{x} \geq \tilde{\boldsymbol{d}},\, \boldsymbol{x} \in \mathbb{R}^{n_1+n_2} \right\}.$$

Passing to the linear dual, we obtain

$$\max \left\{ \boldsymbol{b}^\top \boldsymbol{\pi} + \tilde{\boldsymbol{d}}^\top \boldsymbol{\mu} \,|\, \boldsymbol{A}^\top \boldsymbol{\pi} + \tilde{\boldsymbol{C}}^\top \boldsymbol{\mu} = \boldsymbol{c},\, \boldsymbol{\pi} \geq \boldsymbol{0},\, \boldsymbol{\mu} \geq \boldsymbol{0} \right\},$$

that can be equivalently written by separating the maximum on the two variables $\boldsymbol{\pi}$ and $\boldsymbol{\eta}$, as

$$\max_{\boldsymbol{\pi}} \left\{ \boldsymbol{b}^\top \boldsymbol{\pi} + \max_{\boldsymbol{\eta}} \left\{ \tilde{\boldsymbol{d}}^\top \boldsymbol{\mu} \,|\, \boldsymbol{A}^\top \boldsymbol{\pi} + \tilde{\boldsymbol{C}}^\top \boldsymbol{\mu} = \boldsymbol{c},\, \boldsymbol{\mu} \geq \boldsymbol{0} \right\} \,|\, \boldsymbol{\pi} \geq \boldsymbol{0} \right\}.$$

By dualizing the inner problem, we obtain:

$$\max_{\boldsymbol{\pi}} \left\{ \boldsymbol{b}^\top \boldsymbol{\pi} + \min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{\pi}^\top \boldsymbol{A}\boldsymbol{x} \,|\, \tilde{\boldsymbol{C}}^\top \boldsymbol{x} \geq \tilde{\boldsymbol{d}},\, \boldsymbol{x} \in \mathbb{R}^{n_1+n_2} \right\} \,|\, \boldsymbol{\pi} \geq \boldsymbol{0} \right\}.$$

This can be equivalently rewritten by replacing the feasible region with the convex hull as follows:

$$\max_{\boldsymbol{\pi}} \left\{ \boldsymbol{b}^\top \boldsymbol{\pi} + \min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{\pi}^\top \boldsymbol{A}\boldsymbol{x} \,|\, \boldsymbol{x} \in \mathrm{conv}(X),\, \boldsymbol{x} \in \mathbb{R}^{m+p} \right\} \,|\, \boldsymbol{\pi} \geq \boldsymbol{0} \right\}.$$

Thus, we conclude that the formulation of the Lagrangian Dual is equivalent to the Partial Convex relaxation achieved by dualizing the constraints.

□

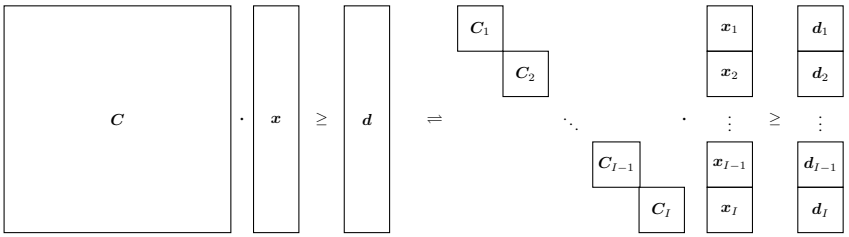

Figure 1.2: Representation of the block decomposition that can be associated with a Lagrangian relaxation.

One word on Integrality Property Some cases exist in which the Lagrangian relaxation coincides with the Continuous relaxation. This equivalence is related to the *integrality property*.

Definition 1.1.2. We say that the constraints $\boldsymbol{Cx} \geq \boldsymbol{d}$ satisfy the *integrality property* if, for each vector of Lagrangian multipliers, $\boldsymbol{\pi}$, the polytope that defines the feasible region of the associated subproblem has only integer vertices.

This implies that the subproblem objective value is equivalent to that of its Continuous relaxation for each fixed vector of Lagrangian multipliers. It is immediate to see that if the problem has only continuous variables, then every possible set of constraints satisfies the integrality property. So, the subproblem can be solved as straightforwardly as a Linear problem. A consequence is that if all the variables are continuous, the Lagrangian and Linear Dual coincide.

In general, the Lagrangian relaxation coincides with the convexification $(\tilde{P})$ of (P), and so $v(LD) \geq v(\bar{P})$, that is the value of the Lagrangian Dual cannot be worse than the one of the Continuous relaxation. This follows from the fact that the two relaxations have the same objective function, but the feasible region of $(\tilde{P})$ is contained in that of $(\bar{P})$.

This finding emphasizes that the subproblem must be *more challenging* than a Linear problem to achieve a better bound than the Continuous relaxation. Understanding that the integrality property is determined by the constraints explicitly included in the formulation is essential. Consequently, the same problem can have different Lagrangian relaxations, some providing the same bound as the Continuous relaxation, while others providing better bounds. If the problems have only continuous variables, any Lagrangian relaxation will satisfy the integrality property.

1.1.2 Block Decomposition

A particular case of interest arises when the matrix $\boldsymbol{C}$ exhibits a block-disjointed structure, in the sense that there exist sub-matrices $\boldsymbol{C}_i$ for $i = 1, \cdots, I$ of $\boldsymbol{C}$ such that the constraints $\boldsymbol{Cx} \geq \boldsymbol{d}$ can be rewritten as

$$\boldsymbol{C}_i \boldsymbol{x}_i \geq \boldsymbol{d}_i,$$

where we divide the variables as $x = (x_1, \cdots, x_I)$

This allows us to split the Lagrangian subproblem into *independent* subproblems of the form:

$$\begin{aligned}\big(LR_i(\boldsymbol{\pi})\big) \qquad \min_{\boldsymbol{x}_i} \; & \boldsymbol{w}_i^\top \boldsymbol{x}_i + \boldsymbol{\pi}^\top(\boldsymbol{b}_i - \boldsymbol{A}_i\boldsymbol{x}_i) \\ & \boldsymbol{C}_i\boldsymbol{x}_i \geq \boldsymbol{d}_i, \;\; i = 1, \cdots, I \\ & \boldsymbol{x}_i \in \mathbb{R}_+^{n_{1,i}} \times \mathbb{N}^{n_{2,i}}, \;\; i = 1, \cdots, I\end{aligned}$$

Thus, the Lagrangian Dual can be rewritten as:

$$\max_{\boldsymbol{\pi} \geq \boldsymbol{0}} \sum_{i=1}^{I} LR_i(\boldsymbol{\pi}).$$

The main advantage is that we can solve I independent subproblems with a smaller size (and easier to solve) than the original problem.

1.2 Karush Kuhn Tucker Conditions

Consider an optimization problem of the form:

$$\min_{\boldsymbol{x} \in X} f(\boldsymbol{x}) \tag{1.3a}$$

$$\text{s.t. } g_i(\boldsymbol{x}) \leq 0 \qquad i = 1, \cdots, m \tag{1.3b}$$

$$h_j(\boldsymbol{x}) = 0 \qquad j = 1, \cdots, p \tag{1.3c}$$

$$\tag{1.3d}$$

where X is a convex subset of $\mathbb{R}^m$, f is a convex function.

The Karush–Kuhn–Tucker (KKT) conditions are first-order necessary conditions for a solution of an optimization problem to be optimal, provided that certain regularity conditions hold.

We begin by introducing some preliminary definitions.

Definition 1.2.1. Given a point $\boldsymbol{x} \in \mathbb{R}^m$ and $\epsilon > 0$ we define the *ball* centered at $\boldsymbol{x}$ with radius ϵ as

$$B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^m \,|\, ||\boldsymbol{x} - \boldsymbol{y}|| \leq \epsilon\}.$$

Definition 1.2.2. Let $\boldsymbol{x} \in \mathbb{R}^n$ and $f : X \to \mathbb{R}$ be a continuously differentiable function. We say that $\boldsymbol{x}$ is a *saddle point* for the function f if it is a stationary point (i.e., the orthogonal derivatives in the point are zero) but it is not a maximum or minimum of the function (neither global nor local), i.e. $\forall \epsilon > 0, \;\; \exists \boldsymbol{x}_m, \boldsymbol{x}_M \in B_\epsilon(\boldsymbol{x})$ such that $f(\boldsymbol{x}_m) < f(\boldsymbol{x}) < f(\boldsymbol{x}_M)$.

Definition 1.2.3. Let $X \subseteq \mathbb{R}^m$. We define the *affine set generated by X* as

$$\text{aff}(X) := \{\boldsymbol{y} \in \mathbb{R}^m \,|\exists n, \exists \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in X, \; \theta_1, \cdots, \theta_n \in \mathbb{R} \text{ s.t. } \boldsymbol{y} = \sum_{i=1}^{n} \theta_i \boldsymbol{x}_i \,\}$$

The **Lagrangian Function** associated with the optimization problem is defined as:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \equiv f(\boldsymbol{x}) + \boldsymbol{\mu}^\top g(\boldsymbol{x}) + \boldsymbol{\lambda}^\top h(\boldsymbol{x})$$

The Karush–Kuhn–Tucker theorem then states that:

- Sufficient Conditions: If $(\boldsymbol{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ is a saddle point of $\mathcal{L}$ in $\boldsymbol{x} \in X$ and $\boldsymbol{\mu} \geq 0$, then $\boldsymbol{x}^*$ is an optimal vector for Problem (1.3).

- Necessary Conditions: Suppose that f and each g_i are convex in X for $i = 1, \cdots, m$ and that there exists $\boldsymbol{x}_0 \in relint(X) := \{\boldsymbol{x} \in X \mid \exists \epsilon > 0 \text{ s.t. } B_\epsilon(\boldsymbol{x}) \cap \text{aff}(X) \subseteq X\}$ such that $g(\boldsymbol{x}_0) < 0$. Given an optimal vector $\boldsymbol{x}^*$ for the above optimization problem there is associated a vector $(\boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ satisfying $\boldsymbol{\mu}^* \geq 0$ such that $(\boldsymbol{x}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*)$ is a saddle point of $\mathcal{L}$.

In particular, supposing that the objective function f and all the constraint functions g_i, h_j admit a sub-derivative in the optimal solution $\boldsymbol{x}^*$, the necessary conditions allow us to rewrite the solution of the optimization problem as a solution of the system composed of the following conditions:

Stationarity Condition:

$$\mathbf{0} \in \partial f(\boldsymbol{x}) + \sum_{i=1}^{m} \mu_i \partial g_i(\boldsymbol{x}) + \sum_{j=1}^{p} \lambda_j \partial h_j(\boldsymbol{x})$$

Primal Feasibility Condition:

$$g_i(\boldsymbol{x}) \leq 0 \qquad \forall i = 1, \cdots, m$$

and

$$h_j(\boldsymbol{x}) = 0 \qquad \forall j = 1, \cdots, p$$

Dual Feasibility Condition:

$$\mu_i \geq 0 \qquad \forall i = 1, \cdots, m$$

Complementary Slackness Condition:

$$\mu_i g_i(\boldsymbol{x}^*) = 0 \qquad \forall i = 1, \cdots, m$$

The system of KKT conditions is typically not directly solved, and many optimization algorithms can be seen as methods for numerically solving the KKT system of conditions [35, page 244].

1.3 Ascent Type Methods

Definition 1.3.1. Given a convex subset Π of a real vector space, we say that the function $\phi : \Pi \to \mathbb{R}$ is *concave* if for all $0 \leq t \leq 1$ and $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2 \in \Pi$:

$$\phi(t\boldsymbol{\pi}_1 + (1-t)\boldsymbol{\pi}_2) \geq t\phi(\boldsymbol{\pi}_1) + (1-t)\phi(\boldsymbol{\pi}_2).$$

We focus on algorithms designed to tackle problems of the form:

$$\max_{\boldsymbol{\pi} \geq 0} \phi(\boldsymbol{\pi}) \tag{1.4}$$

where $\phi : \mathbb{R}^m \to \mathbb{R}$ is a concave function taking finite values in $\mathbb{R}$. If Problem (1.4) corresponds to the Lagrangian Dual obtained by relaxing equality constraints, instead of inequality constraints, the feasible region becomes $\boldsymbol{\pi} \in \mathbb{R}^m$.

While many works focus on convex minimization problems, concave maximization is fundamentally equivalent, as $\max_x(-f(x)) = -\min_x f(x)$, and if f is convex, then $-f$ is concave. We adopt the concave maximization formulation, as it aligns with the approaches developed in Chapter 4 and Chapter 5. For example, looking at the Lagrangian relaxation, we can find the Lagrangian Dual formulation by simply substituting $\phi(\cdot)$ with the Lagrangian subproblem value function $LR(\cdot)$, that is a concave piecewise linear function.

Many algorithms for convex/concave problems are based on **subgradients**, which generalize derivative functions for smooth functions to convex/concave functions.

Definition 1.3.2. We say that $\boldsymbol{g}$ is a *subgradient* of a concave function ϕ in $\boldsymbol{\pi}_0$ if $\forall \boldsymbol{\pi} \in \Pi$:

$$\phi(\boldsymbol{\pi}) - \phi(\boldsymbol{\pi}_0) \leq \boldsymbol{g}^\top(\boldsymbol{\pi} - \boldsymbol{\pi}_0).$$

To be exact, the previous definition refers to *supergradient*, which is used in the case of a concave function. The *subgradient* is defined similarly, but reversing the inequality. Henceforth, in the rest of this Thesis, we will refer to *subgradient* with a slight abuse of notation to talk about what is generally referred to as the supergradient in the literature.

In the case of the Lagrangian relaxation, we can find a subgradient in $\boldsymbol{\pi}_0$ by solving the associated Lagrangian subproblem $LR(\boldsymbol{\pi}_0)$, taking the associated primal solution $\boldsymbol{x}_0^*$, and then the subgradient will correspond to the constraint violation for that primal solution, that is simply

$$(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0^*).$$

This follows from the fact that

$$\partial_{\boldsymbol{\pi}} \left(\min_{\boldsymbol{x} \in conv(x)} \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\pi}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \right) \supseteq \partial_{\boldsymbol{\pi}} \left(\boldsymbol{c}^\top \boldsymbol{x}_0^* + \boldsymbol{\pi}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0^*) \right) = (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0^*).$$

To solve problems such as (1.4), a well-known approach is to use Subgradient methods [207, Chap 5.3]. Specifically referred to as Subgradient Ascent methods for concave maximization, as opposed to the Subgradient Descent methods, typically employed for convex minimization problems.

Some Details on Clarke Subgradient To formally clarify some details in Chapter 4 and Chapter 5, we have to consider a definition that is a bit more general than the subgradient, that is, the Clarke subgradient. Informally, we need a definition similar to that of subgradient, acting *locally* and applicable even when the function is not mandatory convex/concave. The Clarke subgradient is defined for *K-Lipschitz functions in Banach spaces*

Definition 1.3.3. Let X be a Banach space and $Y \subseteq X$. A function $\phi : X \to \mathbb{R}$ is said to be a *Lipschitz* function on Y if exists $K > 0$ such that

$$|\phi(\boldsymbol{x}) - \phi(\boldsymbol{y})| \leq K||\boldsymbol{x} - \boldsymbol{y}|| \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in Y$$

This definition considers a function such that the absolute value of the difference between the function computed at two points is bounded by the distance of the two points, in the Banach space, rescaled to some constant K.

Given a direction v with respect to which we want to approximate the derivative, we start considering a notion of *generalized directional derivative* through this direction as:

Definition 1.3.4. Given a Banach space X and a *Lipschitz* function $\phi : X \to \mathbb{R}$ we define the *generalized directional derivative* of f in $\boldsymbol{x}$ through the direction $\boldsymbol{v}$ as

$$\phi^{\circ}(\boldsymbol{x}; \boldsymbol{v}) = \limsup_{\boldsymbol{y} \to \boldsymbol{x},\, t \downarrow 0} \frac{\phi(\boldsymbol{y} + t\boldsymbol{v}) - \phi(\boldsymbol{y})}{t}$$

where $\boldsymbol{y} \in X$ and $t \in \mathbb{R}$.

If ϕ is a $K-$Lipschitz function there exists at least one vector ξ such that $\phi^{\circ}(\boldsymbol{x}; \boldsymbol{v}) \geq \langle \xi, \boldsymbol{v} \rangle$ [53, Chapter 2]. This justifies considering the following definition for generalized subgradient.

Definition 1.3.5. If ϕ is a $K-$Lipschitz function near $\boldsymbol{x} \in X$, we define the *Clarke subgradient* of ϕ at $\boldsymbol{x}$ as

$$\partial\phi(x) = \{\xi \in X^* \mid \phi^{\circ}(\boldsymbol{x}; \boldsymbol{v}) \geq \langle \xi, \boldsymbol{v} \rangle \ \forall \boldsymbol{v} \in X\}$$

where X^* is the dual space of X.

We refer the reader to [53] for more details. Here, we emphasize that the Clarke subgradient reduces to the gradient of ϕ when the function is differentiable, and coincides with the subgradient when ϕ is convex (or concave using an analogous definition of supergradient). Furthermore, this definition enables the derivation of a chain rule analogue in this setting.

1.3.1 Subgradient Method

In the scenario where $\Pi = \mathbb{R}^m$, given a starting point, $\boldsymbol{\pi}_0 \in \mathbb{R}^m$. The Subgradient Ascent method involves iterative updates of the current point in the following form:

$$\boldsymbol{\pi}_{t+1} = \boldsymbol{\pi}_t + \eta_t \mathbf{g}_t \qquad \forall t = 0, 1, \cdots \tag{1.5}$$

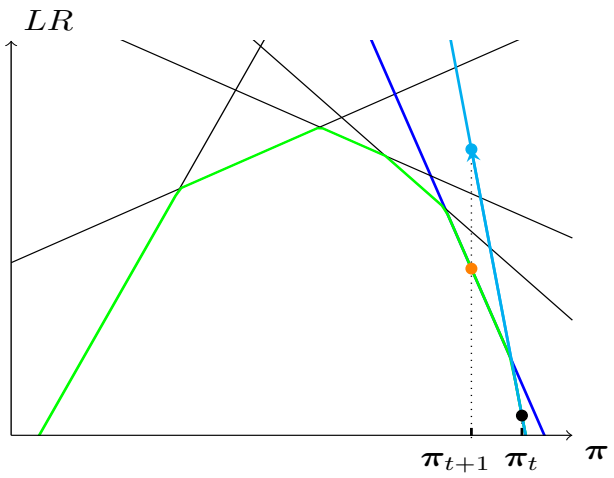

Figure 1.3: Starting from the black point, we use the direction provided by a subgradient at that point (cyan line) to make a step. Computing the value of the true function at that point (orange point), we can obtain a new subgradient (blue line) for the next iteration.

Here, $\mathbf{g}_t \in \partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_t)$ is a subgradient of ϕ at $\boldsymbol{\pi}_t$. So, the approach tries to improve the current point by taking a step of size η_t, known as the *step-size*, in the direction provided by the subgradient of ϕ at the current point.

Choosing an appropriate step size is crucial for converging Subgradient methods, especially when the objective function is non-differentiable. For more information on the step size conditions required to ensure convergence, refer to [32].

Subgradient methods can exhibit instability, as there is no guarantee that the next trial point will yield an improvement over the current one. Therefore, successful implementations must keep track of the best point found so far, which provides the maximum value of the function ϕ.

When the function to be maximized is also differentiable, the Subgradient method consists of exactly one element, corresponding to the function's gradient [238, Theorem 25.1].

Projected Subgradient Descent If additional constraints define Π, as the non-negativity constraints $\boldsymbol{\pi} \geq 0$, we can employ an extension of the previous algorithm to handle these constraints. This extension is called *Projected Subgradient method*, as it involves projecting the point obtained by the Subgradient method's update rule into Π. Therefore, one iteration can be expressed as:

$$\boldsymbol{\pi}_{t+1} = \mathcal{P}_{\Pi}(\boldsymbol{\pi}_t + \eta_t \boldsymbol{g}_t) \tag{1.6}$$

where $\mathcal{P}_{\Pi}$ is the projection into Π and, as previously, $\boldsymbol{g}_t \in \partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_t)$ is a subgradient of ϕ in $\boldsymbol{\pi}_t$.

ADAM A particular case of the Subgradient method is when the function ϕ that we want to maximize is smooth. In this case, the subgradient set will contain only one element: the function gradient.

Gradient-type methods are also commonly employed to solve machine learning tasks. To mitigate the instability issues associated with standard Gradient method, many variants incorporate the concept of *momentum* [218]. This involves aggregating the gradients of the points visited during the previous iterations. *ADAptive Moment estimation* (ADAM) [134] is one widely used variant of the Subgradient method that is nowadays standalone in machine learning.

At any given iteration $t = 0, 1, \cdots, n$, this method first computes the *momentum* as a convex combination of the previous momentum and the gradient at the current trial point, i.e.

$$\boldsymbol{m}_{t+1} \leftarrow \beta_1 \boldsymbol{m}_t + (1 - \beta_1)\boldsymbol{g}_t$$

where $\boldsymbol{g}_t \in \partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_t)$ and $\beta_1 \in [0, 1[$ is a fixed parameter. For the first iteration we consider $\boldsymbol{m}_0 = \mathbf{0}$. Similarly, it also computes the second moments of the gradient as

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2) \left\| \boldsymbol{g}_t \right\|_2^2 .$$

Also in this case, we consider, at the first iteration, $v_0 = 0$

These two values represent an aggregation of the first and second-order information about the function being optimized, which has been collected at all the points so far visited during the algorithm's execution.

Kingma and Ba [134] note that m_t and v_t are biased towards zero, especially in the first iterations and especially when the decay rates are low. For this reason, they are first unbiased using

$$\bar{\boldsymbol{m}}_{t+1} = \frac{\boldsymbol{m}_{t+1}}{1 - \beta_1^t}$$

and

$$\bar{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^t},$$

then, the next trial point is chosen as

$$\boldsymbol{\pi}_{t+1} \leftarrow \boldsymbol{\pi}_t + \eta_t \frac{\bar{\boldsymbol{m}}_{t+1}}{\sqrt{\bar{v}_{t+1}} + \epsilon}.$$

where $\epsilon > 0$ is a (small) scalar and $\eta_t > 0$ is the step size. The step-size rule is originally presented as $\eta_t = \eta$ with $\eta > 0$ fixed.

1.3.2 Cutting Plane Method

The Cutting Plane method [225] is an optimization technique often used to solve Linear Programming problems in formulations with a large number of constraints as well as to solve general concave (and convex), possibly non-differentiable, optimization problems. This technique is combined with branching frameworks to solve MILPs, resulting in the so-called Branch & Cut [65, 167].

Cutting Plane involves refining a feasible set or objective function through the use of linear inequalities, known as cuts. We will focus here on the application to convex/concave problems.

The function ϕ is concave, hence, given a subgradient $\boldsymbol{g}_{\bar{\boldsymbol{\pi}}}$ of ϕ in $\bar{\boldsymbol{\pi}}$ and its linearization error at zero $\alpha_{\bar{\boldsymbol{\pi}}} = \phi(\bar{\boldsymbol{\pi}}) - \phi(\mathbf{0}) - \boldsymbol{g}_{\bar{\boldsymbol{\pi}}}^\top(\bar{\boldsymbol{\pi}} - \mathbf{0}) = \phi(\bar{\boldsymbol{\pi}}) - \phi(\mathbf{0}) - \boldsymbol{g}_{\bar{\boldsymbol{\pi}}}^\top\bar{\boldsymbol{\pi}}$ we have that $\phi(\boldsymbol{\pi}) \leq \boldsymbol{g}_{\bar{\boldsymbol{\pi}}}^\top\boldsymbol{\pi} + \alpha_{\bar{\boldsymbol{\pi}}}$. Let us observe that we can always rescale the function ϕ in such a way that $\phi(\mathbf{0}) = 0$. In the rest of this section, we will use this observation, and so we simplify the linearization errors as $\alpha_{\bar{\boldsymbol{\pi}}} = \phi(\bar{\boldsymbol{\pi}}) - \boldsymbol{g}_{\bar{\boldsymbol{\pi}}}^\top\bar{\boldsymbol{\pi}}$.

Let $\beta^* = \{(\alpha_{\boldsymbol{\pi}}, \boldsymbol{g}_{\boldsymbol{\pi}}) \mid \boldsymbol{g}_{\boldsymbol{\pi}} \in \partial\phi(\boldsymbol{\pi}),\ \alpha_{\boldsymbol{\pi}} = \phi(\boldsymbol{\pi}) - \boldsymbol{g}_{\boldsymbol{\pi}}^\top\boldsymbol{\pi}\}_{\boldsymbol{\pi}\in\Pi}$. Problem (1.4) can be rewritten as:

$$\max_{v,\boldsymbol{\pi}} \ v \tag{1.7a}$$

$$\text{s.t. } v \leq \boldsymbol{g}^\top\boldsymbol{\pi} + \alpha \qquad \forall(\boldsymbol{g},\alpha) \in \beta^* \tag{1.7b}$$

$$\boldsymbol{\pi} \geq 0 \tag{1.7c}$$

$$v \in \mathbb{R}. \tag{1.7d}$$

The issue with this formulation is that the number of constraints of the type (1.7b) can be potentially infinite, making it intractable. Let us observe that in the case of the Lagrangian relaxation (but more generally when the function is piecewise linear), we can restrict it to a finite set of subgradients. We can consider a function $LR_{\boldsymbol{x}}(\boldsymbol{\pi}) = \boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{\pi}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})$ for each vertex of the polytope defined by the inequalities $\boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d}$. For a piecewise linear function, this approach reduces to taking one point in each distinct linear segment. By doing this, we can reduce the problem to a finite bundle size, even though it may still be exponentially large, as a function of the number of variables.

Even if, in this case, β^* has a finite dimension, it may still be excessively large to be solved using the Formulation 1.7.

Instead of considering all the constraints in the formulation, the Cutting Plane method focuses only on a subset $\beta \subset \beta^*$ and iteratively finds the constraint that can be added to the problem to improve the best incumbent value.

This means that we consider instead a Reduced Master problem:

$$\max_{v,\boldsymbol{\pi}\in\mathbb{R}\times\mathbb{R}^m} \ v \tag{1.8a}$$

$$\text{s.t. } v \leq \boldsymbol{g}^\top\boldsymbol{\pi} + \alpha \qquad \forall(\boldsymbol{g},\alpha) \in \beta \tag{1.8b}$$

$$\boldsymbol{\pi} \geq 0 \tag{1.8c}$$

Given the current objective value and solution of the Reduced Master problem $v^*, \boldsymbol{\pi}^*$, we can update the set β by computing the value of the function ϕ in the current optimal solution $\boldsymbol{\pi}^*$.

If v^* is equal to the optimal objective value of the Reduced Master problem, then $\boldsymbol{\pi}^*$ is the optimal solution of the Problem (1.4) as it respects all the constraints (1.7b). Otherwise, we can add the subgradient and the linearization error associated with $\boldsymbol{\pi}^*$ to β and solve the new Reduced Master problem.

Cutting Planes and Column Generation Cutting Planesare an essential feature of MILP solvers, which must balance strengthened linear relaxation with increased

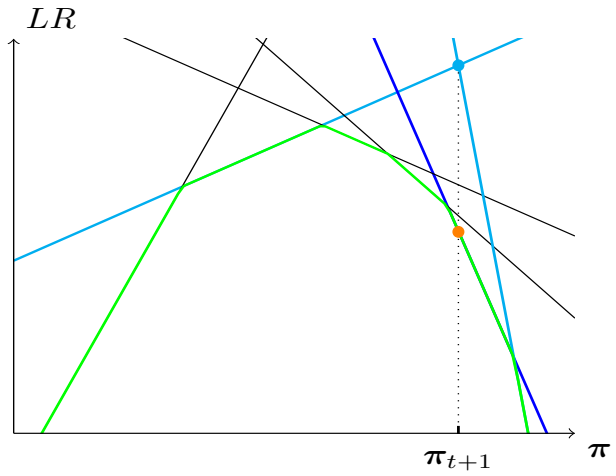

Figure 1.4: This figure illustrates an example of cuts, depicted in cyan, alongside the corresponding optimum of the Cutting Plane method for that iteration. At this stage, we can compute the value for the *real* function (orange point) and an associated subgradient (blue line). This subgradient can be used in the next iteration to improve the model.

computations due to added cuts [66]. This method is strictly related to the Column Generationapproach, as generating cuts (constraints) in the primal problem is equivalent to generating variables in the dual problem. For simplicity in notation, we assume that $\beta = \{(\boldsymbol{g}_1, \alpha_1), \cdots, (\boldsymbol{g}_{|\beta|}, \alpha_{|\beta|})\}$.If we consider the dual of the Cutting Plane Reduced Master problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{|\beta|}} \sum_{i=1}^{|\beta|} \alpha_i \theta_i \tag{1.9a}$$

$$\text{s.t.} \sum_{i=1}^{|\beta|} \theta_i = 1 \tag{1.9b}$$

$$\sum_{i=1}^{|\beta|} \boldsymbol{g}_i \theta_i \geq \mathbf{0} \tag{1.9c}$$

we can see that this problem is one variable for each point $\boldsymbol{\pi}$ with the associated subgradients and linearization errors. If we consider $\Pi = \mathbb{R}^m$, that is, we remove the non-negativity constraints, then Constraint 1.9c should be respected with equality.

We present this relationship here for two main reasons. First, it is necessary to correctly situate the literature of machine learning for Column Generation and machine learning or Cutting Plane, presented in Chapter 3. These two approaches should not be seen as completely independent, but related by the duality theory. The other reason is that, in our implementation of the Bundle method (presented in Section 1.3.3), we will solve the Dual formulation of a regularized version of the Cutting Plane Master

problem, which leads to a regularized version of the Problem (1.9).

Application to Lagrangian Relaxation In the context of the Lagrangian relaxation $\phi \equiv LR$, so (1.4) becomes the Lagrangian Dual

$$\max_{\boldsymbol{\pi} \in \Pi} \left(\min_{\boldsymbol{x} \in \chi} (\boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\pi}^\top A\boldsymbol{x}) - \boldsymbol{\pi}^\top b \right). \tag{1.10}$$

That problem can be rewritten as:

$$\max_{v, \boldsymbol{\pi}} \left(v - \boldsymbol{\pi}^\top b \right) \tag{1.11}$$

$$\text{s.t. } v \leq \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\pi}^\top A\boldsymbol{x} \qquad \forall \boldsymbol{x} \in conv(\chi) \tag{1.12}$$

$$\boldsymbol{\pi} \in \Pi \tag{1.13}$$

$$v \in \mathbb{R}, \tag{1.14}$$

and the Reduced Master problem considers a subset of points in $conv(\chi)$.

Then, given $v^*, \boldsymbol{\pi}^*$, the current solution of the Master problem, iteratively update the set β by solving the subproblem:

$$\min_{\boldsymbol{x} \in \chi} \boldsymbol{c}^\top \boldsymbol{x} + (\boldsymbol{\pi}^*)^\top A\boldsymbol{x} \tag{1.15}$$

If v^* is less than or equal to the optimal objective value of (1.15), then $\boldsymbol{\pi}^*$ is the optimal solution of the Problem (1.10) as it respects all the constraints (1.12). Otherwise, denoting by $\boldsymbol{x}^*$ the optimal solution of 1.15, we can add the associated subgradient and linearization error to β, then solve the new Reduced Master problem.

1.3.3 Bundle Method

The Bundle method (BM) [155, 154] is an iterative method that solves problems of type (1.4). From a general point of view, Bundle method aims to solve the Problem (1.4) by iteratively optimizing a model ϕ_β of the function ϕ plus some regularization, in many cases, the Euclidean norm of the distance between the best point found so far and the new trial point. The model ϕ_β is iteratively updated based on the information from the trial points. A common choice for the model ϕ_β is the Cutting Plane Model presented so far. In the following, we will use *only* that model. In [84], theoretical conditions are posed on the convergence of the Bundle in a general setting.

Note that the optimal solution $\boldsymbol{\pi}^*$ of the problem

$$\max_{\boldsymbol{\pi} \geq 0} \phi(\boldsymbol{\pi})$$

and the optimal solution d^* of the problem

$$\max_{\boldsymbol{d}: \boldsymbol{d} + \bar{\boldsymbol{\pi}} \geq 0} \phi(\boldsymbol{d} + \bar{\boldsymbol{\pi}}) - \phi(\bar{\boldsymbol{\pi}})$$

are related by the relationship $\boldsymbol{\pi}^* = \bar{\boldsymbol{\pi}} + \boldsymbol{d}^*$, for any given vector $\bar{\boldsymbol{\pi}}$.

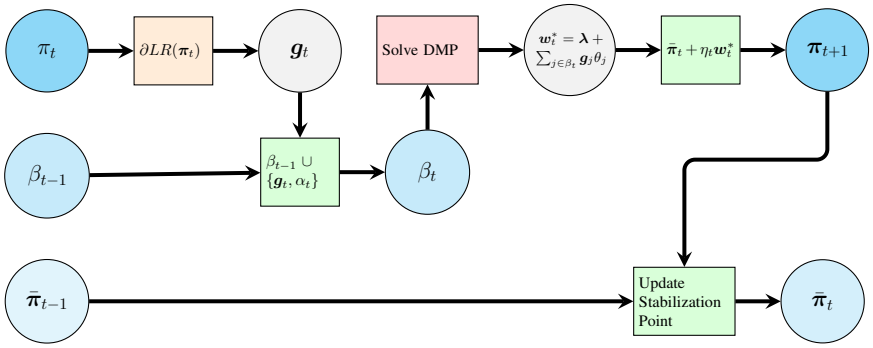

Figure 1.5: Bundle execution. Schematic representation of one iteration. Starting from the trial point $\boldsymbol{\pi}_{t-1}$, the bundle β_{t-1} and the stabilization point $\bar{\boldsymbol{\pi}}_{t-1}$ at iteration $t-1$ we compute the same values for the next iteration t.

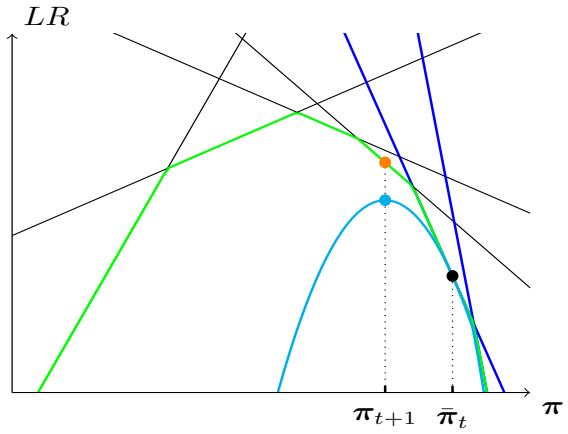

Figure 1.6: Bundle iteration for $\phi = LR$, the *blue* lines correspond to the planes for which we have the gradient information in the bundle. The coordinate in the x-axis of the *black* point represents the current stabilization point. The *cyan* lines represent the quadratic approximation at the current iteration obtained considering the Cutting Plane Model provided by the **blue** lines and adding a quadratic regularization as the Euclidean distance the stabilization point. The coordinate in the x-axis of the *orange* point indicates the new trial point found as a maximum of the current quadratic approximation.

The term $\phi(\bar{\boldsymbol{\pi}})$ is added to rescale the function, representing the difference between the new trial point and the previous stabilization point. We can then use the Cutting Plane approach to this new problem. The main difference is that in this case, we have as linearization errors $\alpha_i = \phi(\boldsymbol{\pi}_i) - \phi(\bar{\boldsymbol{\pi}}_t) - \boldsymbol{g}_t^\top(\boldsymbol{\pi}_i - \bar{\boldsymbol{\pi}}_t) \geq 0$ and the new point is chosen as $\boldsymbol{\pi}_{t+1} = \bar{\boldsymbol{\pi}}_t + \boldsymbol{d}_t^*$.

Many implementations of the Bundle method consider the function $\phi_{\bar{\boldsymbol{\pi}}}(\boldsymbol{d}) = \phi(\bar{\boldsymbol{\pi}} + \boldsymbol{d}) - \phi(\bar{\boldsymbol{\pi}})$ for a fixed $\bar{\boldsymbol{\pi}}$ the Problem (1.4) is equivalent to:

$$\max_{\boldsymbol{d} \in \mathbb{R}^n \,:\, \bar{\boldsymbol{\pi}} + d \geq 0} \phi_{\bar{\boldsymbol{\pi}}}(\boldsymbol{d}) \tag{1.16}$$

in the sense that we can find the solution $\boldsymbol{\pi}^*$ of (1.4) as $\boldsymbol{\pi}^* = \bar{\boldsymbol{\pi}} + \boldsymbol{d}^*$, where $\boldsymbol{d}^*$ is the optimal solution of (1.16). Note that the optimal values do not coincide as $\phi_{\bar{\boldsymbol{\pi}}}(\boldsymbol{d}^*) = \phi(\boldsymbol{\pi}^*) - \phi(\bar{\boldsymbol{\pi}})$. We adopt this formulation to facilitate a clearer interpretation of the results presented in Chapter 5.

At a given iteration t, BM starts with a *trial point* $\boldsymbol{\pi}_t$ and uses a *bundle* β_t (a set containing a collection of gradients of the objective function $\phi_{\bar{\boldsymbol{\pi}}}(\boldsymbol{d})$) to estimate the real objective function. More formally, let $\beta_t = \{(\boldsymbol{g}_1, \alpha_1) \dots (\boldsymbol{g}_{|\beta_t|}, \alpha_{|\beta_t|})\}$ be the set of subgradients g_i of $\phi_{\bar{\boldsymbol{\pi}}_t}(\boldsymbol{d})$ in $\boldsymbol{d}_i = \boldsymbol{\pi}_i - \bar{\boldsymbol{\pi}}_t$ (that is equivalent to the gradient of $\phi(\boldsymbol{\pi})$ in $\boldsymbol{\pi}_t$) and $\alpha_i = (\phi(\boldsymbol{\pi}_i) - \phi(\bar{\boldsymbol{\pi}}_t)) - \boldsymbol{g}_i^\top(\boldsymbol{\pi}_i - \bar{\boldsymbol{\pi}}_t)$ be the corresponding linearization errors, collected up to iteration t. Let us observe that, by concavity and by the definition of subgradient, the linearization errors are always non-negative, that is, $\alpha_i \geq 0$. We select these linearization errors because they allow for a more accurate representation of the Bundle iteration, aligning better with the results developed in Chapter 5.

BM aims at finding the optimal solution of (1.16) by using the information contained in the bundle β_t to construct an approximation of $\phi_{\bar{\boldsymbol{\pi}}}(\boldsymbol{d})$ defined as follows:

$$\phi_{\bar{\boldsymbol{\pi}}_t, \beta_t}(\boldsymbol{d}) = \min_i \{\boldsymbol{g}_i^\top \boldsymbol{d} + \alpha_i \,|\, i \in \{1, \cdots, |\beta_t|\}\}. \tag{1.17}$$

The idea of BM is that, if the set β_i and the parameter t_i are chosen appropriately, the optimal solution of the following problem

$$(MP_t) \quad \max_{\boldsymbol{d}} \{\phi_{\bar{\boldsymbol{\pi}}_t, \beta_t}(\boldsymbol{d}) - \frac{1}{2\eta_t} ||\boldsymbol{d}||_2^2 \,|\, \boldsymbol{d} \in \mathbb{R}^n \,:\, \bar{\boldsymbol{\pi}}_t + \boldsymbol{d} \geq 0\} \tag{1.18}$$

is sufficiently close to the optimal solution of the original problem (1.16), see [84].

Problem (1.18) consists of a concave piecewise linear term and a *normalization term* weighted by the parameter η_t.

Other Regularization Terms In the definition of the Master problem (1.18), we consider the Euclidean norm as a regularization term. However, this is not the only possibility. For example, Frangioni [84] discusses sufficient theoretical properties that the regularization term should possess to ensure convergence.

Bundle Pseudocode The method begins at iteration $t = 0$ by taking as input an initial point $\boldsymbol{\pi}_0$, the stabilization point equal to the initial point ($\bar{\boldsymbol{\pi}}_0 = \boldsymbol{\pi}_0$) and the initial bundle containing only the information of $\phi(\boldsymbol{\pi}_0)$ (i.e. $\beta_0 = \{\boldsymbol{g}_0, \alpha_0\}$). Figure 1.5 represents an iteration of the BM.

Algorithm 1 Bundle methodpseudo-code.

Choose $\boldsymbol{\pi}_0, \epsilon, \eta^*, \eta_0, m$
$\bar{\boldsymbol{\pi}}_0 \leftarrow \boldsymbol{\pi}_0$
▷ At the beginning, the stabilization point coincides with the initial point
$(\boldsymbol{g}_0, \boldsymbol{\alpha}_0) \leftarrow (\partial\phi(\boldsymbol{\pi}_0), 0)$
$\beta_0 \leftarrow \{(\boldsymbol{g}_0, \boldsymbol{\alpha}_0)\}$
$t \leftarrow 0$
while true do
    $(v_t, \boldsymbol{\theta}^{(t)}, \boldsymbol{\lambda}) \leftarrow$ Solve $DQP(\eta_t, \beta_t)$
    ▷ Solve the Dual Master problem (1.19)
    $\boldsymbol{w}^{(t)} \leftarrow \sum_{i=1}^{|\beta_t|} \boldsymbol{g}_i \theta_i^{(t)}$ ▷ Compute the new trial direction
    $\boldsymbol{\pi}_{t+1} \leftarrow \bar{\boldsymbol{\pi}}_t + \eta_t \left(\boldsymbol{w}^{(t)} + \boldsymbol{\lambda}\right)$ ▷ Compute the new trial point
    $(\boldsymbol{g}_{t+1}, \boldsymbol{\alpha}_{t+1}) \leftarrow (\partial\phi(\boldsymbol{\pi}_{t+1}), \phi(\bar{\boldsymbol{\pi}}_t) - \phi(\boldsymbol{\pi}_{t+1}) + \boldsymbol{g}_{t+1}^\top(\boldsymbol{\pi}_{t+1} - \bar{\boldsymbol{\pi}}_t))$
    ▷ Compute the gradient and the linearization error
    $\beta_{t+1} \leftarrow \beta_t \cup (\boldsymbol{g}_{t+1}, \boldsymbol{\alpha}_{t+1})$ ▷ Update the Bundle
    if $\phi(\boldsymbol{\pi}_{t+1}) - \phi(\bar{\boldsymbol{\pi}}_t) > m(\eta_t || \sum_{i\in\beta_t} \theta_i^{(t)} \boldsymbol{g}_i||^2 + \sum_{i\in\beta_t} \alpha_i \theta_i^{(t)})$ **then**
        $\bar{\boldsymbol{\pi}}_{t+1} \leftarrow \boldsymbol{\pi}_{t+1}$ ▷ Serious Step
        Update $\alpha_i \ \forall i = 1, \cdots, |\beta_t|$
    else
        $\bar{\boldsymbol{\pi}}_{t+1} \leftarrow \bar{\boldsymbol{\pi}}_t$ ▷ Null Step
    end if
    if Stopping Scriteria **then** ▷ End the Algorithm
        break
    end if
    $\beta_{t+1} \leftarrow$ Remove outdated components β_{t+1}
    $t \leftarrow t + 1$
end while

In line 7, using the information contained in β_t, we solve the Dual Master problem (1.18):

$$\min_{\boldsymbol{\theta},\boldsymbol{\lambda}\in\mathbb{R}^{|\beta_t|}\times\mathbb{R}^m} \frac{\eta_t}{2}\left\|\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i+\boldsymbol{\lambda}\right\|_2^2+\sum_{i=1}^{|\beta_t|}\alpha_i\theta_i+\boldsymbol{\lambda}^\top\bar{\boldsymbol{\pi}}_t \tag{1.19a}$$

$$\text{s.t.}\sum_{i=1}^{|\beta_t|}\theta_i=1 \tag{1.19b}$$

$$\bar{\boldsymbol{\pi}}_t+\eta_t\left(\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i+\boldsymbol{\lambda}\right)\geq 0 \tag{1.19c}$$

$$\boldsymbol{\lambda}\geq 0 \tag{1.19d}$$

$$1\geq\boldsymbol{\theta}\geq 0 \tag{1.19e}$$

obtained by passing through the KKT conditions. We refer to [153, Chapter 4] for further details.

Let us observe that, removing the non-negativity constraints, we obtain the following formulation for the Dual Master problem:

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{|\beta_t|}_{\geq 0}}\left(\frac{\eta_t}{2}\left\|\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i\right\|_2^2+\sum_{i=1}^{|\beta_t|}\alpha_i\theta_i:\ \sum_{i=1}^{|\beta_t|}\theta_i=1\right). \tag{1.20}$$

It is important to note that the optimal solution $\boldsymbol{d}_t^*$ of MP_t and the optimal solution $\boldsymbol{\theta}^{(t)}$ of (1.19) are linked by the relationship $\boldsymbol{d}^{(t)}=\eta_t\left(\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i^{(t)}+\boldsymbol{\lambda}\right)$, and so for the Problem (1.20) multipliers we have $\boldsymbol{d}^{(t)}=\eta_t\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i^{(t)}$. For our purposes, it is convenient to introduce a vector $\boldsymbol{w}_t=\sum_{i=1}^{|\beta_t|}\boldsymbol{g}_i\theta_i^{(t)}$, as in line 8, to represent the trial direction from iteration t to iteration $t+1$. In the following, we will denote by v_t the optimal objective value of the Problem (1.19). Ad-hoc methods to solve the Dual Master problem (1.19) exist, see for example [82].

In line 9, we consider the trial point $\boldsymbol{\pi}_{t+1}=\bar{\boldsymbol{\pi}}_t+\eta_t\boldsymbol{w}_t$. We compute $\phi(\boldsymbol{\pi}_{t+1})$ and its subgradient $\boldsymbol{g}_{t+1}\in\partial\phi(\boldsymbol{\pi}_{t+1})$. Then, in line 11, we add the information about the gradient and the linearization error of the new trial point to the bundle $\beta_{t+1}=\beta_t\cup(\boldsymbol{g}_{t+1},\alpha_{t+1})$.

If the condition in line 12 is satisfied, we update the stabilization point (i.e. $\bar{\boldsymbol{\pi}}_{t+1}=\boldsymbol{\pi}_{t+1}$), otherwise we keep it unchanged ($\bar{\boldsymbol{\pi}}_{t+1}=\bar{\boldsymbol{\pi}}_t$). Here $m\in[0,1)$ is a hyperparameter. This condition is not simply $\phi(\boldsymbol{\pi}_{t+1})>\phi(\bar{\boldsymbol{\pi}}_t)$ as, in some cases, it may be beneficial to *sufficiently* improve the information contained in the bundle before changing the stabilization point. The parameter m is typically chosen to be strictly positive (even if relatively small) to ensure theoretical convergence properties.

A step in which the stabilization point is updated is called *Serious Step* (SS). Similarly, a step in which the stabilization point is not changed is referred to as a *Null Step* (NS). Making a SS, in a certain iteration, means a *significant* progress toward improving

the solution, while a NS means no such progress was made. In a Serious Step, we need to update the linearization errors α_i of the elements i in the bundle β_{t+1} (line 14).

In line 18, the stopping criterion is checked to decide whether to quit the loop and stop the algorithm. A possible stopping criterion is

$$\eta^* || \sum_{i=1}^{|\beta_t|} \boldsymbol{g}_i \theta_i^{(t)} || + \sum_{i=1}^{|\beta_t|} \alpha_i \theta_i^{(t)} \leq \epsilon \max(0, \phi(\bar{\boldsymbol{\pi}}_{t+1}))$$

At the end of one iteration, line 21, the efficient implementations of the Bundle delete some old component from β. A common strategy is removing components that are not used in the optimal solution over the last K iterations (a *quite* conservative strategy takes $K \sim 20$). This is a part of the $\beta-$strategy, which determines how we handle the size of the bundle. Other components can consider multiple trial points at each iteration or fix a maximum size for the bundle.

Bundle as Compromise between Cutting Planes and Subgradient Interestingly, the Bundle method can be interpreted as a combination of the Cutting Plane method and the Subgradient method. For simplicity, we restrict to the case in which $\Pi = \mathbb{R}^m$. The key observation is that when the step size is zero, the objective function in the Dual Master problem (1.19) reduces to its linear part $\sum_{i=1}^{|\beta_t|} \alpha_i \theta_i$. In this case, the algorithm selects only the component with the smallest linearization error, which corresponds to the stabilization point. As a result, the update follows the direction given by the subgradient of ϕ at the stabilization point. Conversely, when $\eta \to \infty$, the quadratic part provided by the norm in Problem (1.19) dominates, enforcing the constraints $\left|\left|\sum_{i=1}^{|\beta_t|} \boldsymbol{g}_i \theta_i - \boldsymbol{\lambda}\right|\right|_2^2 = 0$. By definition of norm, this implies $\sum_{i=1}^{|\beta_t|} \boldsymbol{g}_i \theta_i \geq \mathbf{0}$.These inequalities can then be explicitly incorporated into the formulation, allowing the removal of the quadratic term from the objective function. This reformulation leads to the dual representation of the Cutting Plane method, namely the Column Generation Master problem (1.9), seeing that in the Column Generation case $\bar{\boldsymbol{\pi}}_t = \mathbf{0}$.

Proximality Term η-Strategy The η parameter controls the weight of the regularization term, balancing between Cutting Plane and Subgradient method steps. The proper selection of the proximal term η, which controls the regularization term in the Dual Master problem, is both non-trivial and essential. As illustrated in Figure 1.7, this choice depends on the current stabilization point and the information accumulated in the bundle up to the current iteration. In particular, on the right-hand side of Figure 1.7, we observe that when all gradients at the optimum are available, solving a Cutting Plane model is preferable, as it leads to the optimal solution of the problem. Hence, in this case, a higher value of η may be preferred.

In [84], theoretical conditions are established for the η-strategy to ensure convergence. Constant strategies can be effective if appropriately fine-tuned. Many state-of-the-art strategies are self-adjusting rules for choosing η_t.

The main state-of-the-art η-strategies are composed of three *levels*, as the ones developed in [83] and also presented in [57]. These levels includes: *Long-term*, *Middle-term*, and *Short-term η-strategies*.

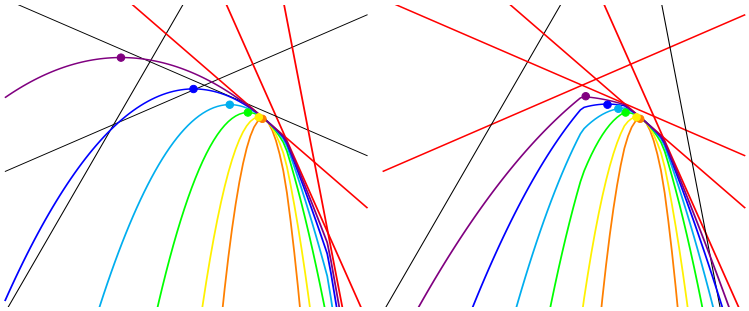

Figure 1.7: A comparison of different quadratic approximations of the Bundle method's Master problem, using varying step sizes and gradient information. The key difference between the two figures is that the one on the left includes less gradient information than the one on the right.

The *Long-term η-strategies* aim to capture the stage of the resolution process. In the Bundle method, this is important because it represents a balance between the Subgradient method and the Cutting Plane method. In particular, we want to prioritize Subgradient method iterations when the solution is still far from optimal, and shift towards Cutting Plane in the final iterations. This approach helps collect the subgradient information necessary to establish the optimality of the solution.

We define the value

$$v_t^* = \eta_t || \sum_{i \in \beta_t} \theta_i^{(t)} \boldsymbol{g}_i ||^2 + \sum_{i \in \beta_t} \alpha_i \theta_i^{(t)}$$

where $\boldsymbol{\theta}^{(i)}$ is the solution of the Dual Master problem at iteration t. The value v^* represents the maximum improvement estimation of the Dual Master problem. We also define

$$\epsilon_t^* = \sum_{i \in \beta_t} \alpha_i \theta_i^{(t)} + \eta^* || \sum_{i \in \beta_t} \theta_i^{(t)} \boldsymbol{g}_i ||^2$$

where η^* is a hyperparameter. The value ϵ_t^* estimates the maximum improvement obtained using a variant of the Dual Master problem in which η^* acts as a regularization parameter. The hyperparameter η^* is generally chosen to be greater than all the possible η_t, thereby emulating the expected behavior by following *more* the decision of the model. It is different from explicitly considering this parameter in the Dual Master problem, as this would alter its solution and consequently the new trial direction, as previously explained. Additionally, we define a hyperparameter $\tilde{m} \in [0, 1)$, which is employed in the η-strategies and is usually fixed to 0.01.

Some examples of Long-term η-strategies are:

- no Long-term η-strategy;
- the *soft* Long-term η-strategy. After a null step, decreases in η_t are inhibited whenever $v_t^* < \tilde{m}\epsilon_t^*$. In other words, we inhibit the decrements are prevented if

η_t is already sufficiently small, such that the maximum improvement is estimated to be less than $\tilde{m}$ times the improvement suggested by a Cutting-Planes model.

- the *hard* Long-term η-strategy. Increases of η_t are allowed, even after null steps, whenever $v_t^* < \tilde{m}\epsilon_t^*$.

- the *balancing* Long-term η-strategy. This strategy aims to keep the two terms $\eta^*||\frac{1}{2}\sum_{i\in\beta_t}\theta_i^{(t)}\boldsymbol{g}_i||_2^2$ *roughly the same size*.

  Increases of η_t, after a serious step, are inhibited if

$$\frac{\eta^*}{2}||\sum_{i\in\beta_t}\theta_i^{(t)}\boldsymbol{g}_i||_2^2 \leq \tilde{m}\sum_{i=1}^{|\beta_t|}\alpha_i\theta_i^{(t)}.$$

  In this case, increasing η_t causes a reduction of $\frac{\eta^*}{2}||\sum_{i\in\beta_t}\theta_i^{(t)}\boldsymbol{g}_i||_2^2$ which is already small.

  On the other hand, decreases in η_t are inhibited if

$$\tilde{m}\frac{\eta^*}{2}||\sum_{i\in\beta_t}\theta_i^{(t)}\boldsymbol{g}_i||_2^2 \geq \sum_{i=1}^{|\beta_t|}\alpha_i\theta_i^{(t)}.$$

  The rational been that decreasing η_t causes an increase in $||\sum_{i\in\beta_t}\theta_i^{(t)}\boldsymbol{g}_i||_2^2$ which is already big.

The *Middle-term η-strategies* are multi-step strategies that consider the outcomes from the *last few iterations*. These strategies include the following conditions:

- A minimum number of consecutive successful steps (SS) with the same η_t must be performed before η_t is allowed to increase.

- A minimum number of consecutive null steps (NS) with the same η_t must be performed before η_t is allowed to decrease.

At the bottom, heuristic *Short-term η-strategies* only rely on information from the current iteration to propose an update for η_t. Specifically, these heuristics are only used to assist in selecting the current value, but both the top and the middle layers would agree on the update. However, sometimes it can be beneficial to allow the Short-term heuristics to do *small adjustments* in η_t, regardless of the decisions made by the other levels.

We present here four different examples of *Short-term η-strategies* that could be considered in the same implementation:

- Increase η_{t+1} as $\eta_t\eta_{incr}$, with $\eta_{incr} > 1$.

- Decrease η_{t+1} as $\eta_{t-1}\eta_{decr}$, with $0 < \eta_{decr} < 1$.

- When increased, η_{t+1} should not exceed η_M, with $\eta_M > 0$.

- When decreased, η_{t+1} should not go below η_m, with $0 < \eta_m < \eta_M$.

Some more elaborate *Short-term $\eta-$strategies* are possible. The first three are based on a quadratic interpolation of the function based on known data, while the fourth is the so-called reversal form of the poor man's quasi-Newton update.

- The first approach approximating the restriction of $\phi(\cdot)$ along $\eta_t \boldsymbol{w}^*(t)$ with a quadratic function $q(\cdot)$ satisfying $q(0) = \phi(\bar{\boldsymbol{\pi}}_t)$, $q(\eta_t) = \phi(\boldsymbol{\pi}_t)$ and having $q'(0) = v_t^*$. This leads to the following updating rule:

$$\eta_{t+1} \leftarrow \frac{\eta_t v_t^*}{2(v_t^* - (\phi(\boldsymbol{\pi}_t) - \phi(\bar{\boldsymbol{\pi}}_{t-1})))}.$$

- Another strategy rather fixes $q'(t) = \eta_t (\boldsymbol{w}^{(t)})^\top \boldsymbol{g}_t$ leading to the update rule:

$$\eta_{t+1} \leftarrow \frac{\alpha_t + (\phi(\boldsymbol{\pi}_t) - \phi(\bar{\boldsymbol{\pi}}_{t-1}))}{2\alpha_t}.$$

  This has the property that $\eta_{t+1} > \eta_t$ if and only if $(\boldsymbol{w}^{(t)})^\top \boldsymbol{g}_t > 0$

- The third possibility maximizes the piecewise linear function constructed from the two linearizations: one at 0 and the other obtained by restricting to $\eta_t \boldsymbol{w}^{(t)}$. This results in:

$$\eta_{t+1} \leftarrow \frac{\alpha_t - 0}{v - \eta_t (\boldsymbol{w}^{(t)})^\top \boldsymbol{g}_t}$$

- The fourth approach is known as the *reversal form* of the *poor man's quasi-Newton update*, whose nontrivial rationale is discussed in detail in [156]. Another variant of this approach is also presented in [213]. The proposed update in this case is:

$$\eta_{t+1} \leftarrow \frac{\langle \boldsymbol{q}, \boldsymbol{u} \rangle}{||\boldsymbol{q}||_2^2}$$

  where $\boldsymbol{u} = (\bar{\boldsymbol{\pi}}_t - \boldsymbol{q}^\top \boldsymbol{z}^*) - \bar{\boldsymbol{\pi}}_t + \eta_t \boldsymbol{q} = \eta_t \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{q}$ and $\boldsymbol{q} = \boldsymbol{g}_{t+1} - \boldsymbol{z}^*$, with $\boldsymbol{z}^*$ being an arbitrary choice. Often $\boldsymbol{z}^*$ is supposed to be the best (approximate) subgradient we have at $\bar{\boldsymbol{\pi}}_t$. In practice, $\boldsymbol{z}^*$ is typically chosen as $\boldsymbol{g}_t$, leading to $\boldsymbol{q} = \boldsymbol{g}_{t+1} - \boldsymbol{g}_t$.

Aggregated and Disaggregated Bundle The Bundle method can be further specialized to better handle the case where the function ϕ can be rewritten as the sum of functions, as

$$\phi(\boldsymbol{\pi}) = \sum_{j \in J} \phi_j(\boldsymbol{\pi}),$$

where $\phi_j : \mathbb{R}^m \to \mathbb{R}$ are concave functions. An example of this setting is the Lagrangian relaxation with block decomposition, as discussed in Section 1.1.2.

In this case, the Master problem can be reformulated in a specialized manner as:

$$\min_{\boldsymbol{\theta},\boldsymbol{\lambda}} \frac{\eta_t}{2} \left\| \sum_{j\in J} \sum_{i=1}^{|\beta_t|} \boldsymbol{g}_{j,i}\theta_{j,i} + \boldsymbol{\lambda} \right\|_2^2 + \sum_{j\in J} \sum_{i=1}^{|\beta_t|} \alpha_{j,i}\theta_{j,i} + \bar{\boldsymbol{\pi}}_t^\top \boldsymbol{\lambda} \tag{1.21}$$

$$\text{s.t.} \sum_{i=1}^{|\beta_t|} \theta_{j,i} = 1, \ \ \forall j \in J \tag{1.22}$$

$$\bar{\boldsymbol{\pi}}_t + \eta_t \left(\sum_{j\in J} \sum_{i=1}^{|\beta_t|} \boldsymbol{g}_{j,i}\theta_{j,i} + \boldsymbol{\lambda} \right) \geq 0 \tag{1.23}$$

$$1 \geq \theta_{j,i} \geq 0 \tag{1.24}$$

$$\boldsymbol{\lambda} \geq 0 \tag{1.25}$$

$$\tag{1.26}$$

in which we consider separate subgradients and linearization errors for each function ϕ_i.

This variant is known as the *Disaggregated Bundle method*, in contrast to the *Aggregated Bundle method*, presented so far, where the explicit decomposition of ϕ is not considered in the Dual Master problem. A comparison between the Aggregated and Disaggregated Bundle method can be found in [57], in the context of the Multicommodity Capacitated Fixed Charge Network Design problem.

Trust Region Bundle Another variant of the Bundle method involves incorporating the quadratic regularization term as an additional constraint:

$$|| \sum_{i=1}^{|\beta_t|} \boldsymbol{g}_i\theta_i + \boldsymbol{\lambda}||_\infty^2 \leq 2\eta_t.$$

in the Dual Master problem, rather than including it as regularization in the objective function.

This approach defines a family of Bundles known as *Proximal Bundle method*, in contrast to the variant presented so far, referred to as *Proximal Bundle method*. In [84], Frangioni demonstrates that Proximal and Trust Region Bundles can be unified within the same framework, differing only in the choice of regularization term.

Comparison of Bundle Methods - for Lagrangian Relaxations The Subgradient methodhas already been applied to solve the Lagrangian Dual [110] and specialized variants, as the Volume algorithm [17], has been proposed. In [86], Frangioni et al. compare different Subgradient methods and Bundle methods, which differ in their step-size rule, for the Subgradient methods, and the Master problem formulation, for the Bundle methods. They examine different step-size rules for the Subgradient method and various structures of the Master problem in the Bundle method. For small-scale instances, all cuts can be included in an LP formulation, which state-of-the-art LP solvers

can solve. In such cases, directly using an LP solver like CPLEX is preferable, whereas decomposition approaches become more competitive as problem size increases. The Bundle method with a *complex* Master problem is often the best option, providing high-quality dual and accurate primal solutions with only a minor increase in computational effort. However, as the problem size grows, solving the Master problem becomes increasingly expensive, necessitating the use of more efficient, less costly Master problem formulations.

In general, for nearly all tested cases, the Subgradient method with the TVColor step-size rule [86] outperforms the aggregated Master problem with quadratic stabilization while achieving a similar level of accuracy. Notably, while the Subgradient method didn't always achieve the required accuracy, the Aggregated Bundle method consistently did. This confirms what was already experimentally found in [36], which showed that aggregated Bundle methods do not necessarily exhibit faster convergence rates than well-tuned Subgradient methods, despite incorporating substantially more information and paying the corresponding computational cost in solving the Master problem. In [36], these approaches are also compared with classic Column Generation approaches.

In [87], Frangioni et al. propose a simple rule for dynamically adjusting the crucial smoothness parameter in the fast gradient approach. This rule leverages information about the optimal dynamic smoothness parameter to enhance the practical convergence properties of the method.

Chapter 2

Background on Machine Learning

Machine learning (ML) is a domain of computer science that develops and studies statistical algorithms capable of learning to approximate underlying properties of data and generalizing to unseen data. Neural networks [105] (NN) have gained increasing relevance in recent years due to the universal approximation theorems [59, 116], which state that, for each continuous function and each given accuracy, it is possible to construct a neural network that approximates the considered function up to the desired accuracy.

This chapter introduces the ML paradigms and models that will be used throughout this thesis. We focus exclusively on machine learning techniques here. Hence, readers already familiar with these concepts may choose to skip this chapter. For a broader literature review of machine learning methods as applied to the field of Optimization, we refer the reader to Chapter 3, which provides the necessary context for the main contributions of this work within this large research domain.

2.1 Learning Tasks

The theoretical foundations for constructing prediction functions with strong generalization properties for supervised learning tasks are rooted in the area of *statistical learning* [246]. In this chapter, we will use the following definitions and notations.

We consider a certain *input set* $\mathcal{X}$, generally we have a limited amount of samples $\{\boldsymbol{x}_i\}_{i=1}^S \subseteq \mathcal{X}$ drawn independently from a certain *unknown* probability $\mathcal{P}$. We assume that we can associate at each point $\boldsymbol{x} \in \mathcal{X}$ one point in the *output space* $\boldsymbol{y} \in \mathcal{Y}$. The points in the output space can be seen as outputs of a certain *unknown* function $f : \mathcal{X} \to \mathcal{Y}$, but in some cases, it is better to say that they are drawn from an *unknown* conditional probability, given the input, i.e., $\boldsymbol{y} \sim \mathcal{P}(\cdot|\boldsymbol{x})$. Hence we dispose of a limited amount of pairs $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^S \subseteq \mathcal{X} \times \mathcal{Y}$, called a *dataset*.

2.1.1 Supervised Learning

Supervised learning is an ML paradigm where, given a dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{S}$, the aim is to determine the parameters $\boldsymbol{W} \in \mathcal{W}$ of a model $f_{\boldsymbol{W}}$ such that the model's output $\hat{\boldsymbol{y}}_i = f_{\boldsymbol{W}}(\boldsymbol{x}_i)$ will be *similar* to the *true* output $\boldsymbol{y}_i$. The parameters $\boldsymbol{W}$ of the model are typically the solution of a particular optimization problem, called the *training problem*.

The *similarity* is in general described using a *loss function* $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that measures the *difference* of the prediction $\hat{\boldsymbol{y}}_i = f_{\boldsymbol{W}}(\boldsymbol{x}_i)$ to the *true* output $\boldsymbol{y}_i = f(\boldsymbol{x}_i)$. For an introduction to probabilistic machine learning, we refer to [188, 171].

Many supervised learning tasks can be formulated as the problem of finding the parameters $\boldsymbol{W}$ that minimize the expected loss value:

$$\boldsymbol{W}^* \in \arg\min_{\boldsymbol{W}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}}[\mathcal{L}(f_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{y})]. \tag{2.1}$$

In practice, since the underlying distribution $\mathcal{P}$ is unknown, we instead optimize the *empirical expectation* over a dataset composed of independent and identically distributed (i.i.d.) samples:

$$\boldsymbol{W}^* \in \arg\min_{\boldsymbol{W}} \frac{1}{S} \sum_{i=1}^{S} \mathcal{L}(f_{\boldsymbol{W}}(\boldsymbol{x}_i), \boldsymbol{y}_i) \tag{2.2}$$

Once we have found the model's optimal parameters $\boldsymbol{W}^*$, another fundamental problem in ML is the *inference problem*. That is the problem of computing the value $\boldsymbol{y} = f_{\boldsymbol{W}^*}(\boldsymbol{x})$ for some previously unseen data $\boldsymbol{x}$.

Stochastic Gradient Descent

The problem introduced in Equation (2.2) is a peculiar optimization problem, as it involves minimizing a non-convex objective function in a high-dimensional space.

Approximating optimal parameters $\boldsymbol{W}^*$ is typically achieved with Gradient-based method, similar to those presented in Section 1.3 of Chapter 1. Among these, *Adam* [134] is an established standard in ML.

A key challenge in solving non-convex problems with Gradient-based methods is the risk of getting trapped in a local minimum that is far from the global minimum. *Adam* addresses this issue by computing a new trial direction as a convex combination of the current gradient and the previous trial direction.

Furthermore, problems of the form (2.2) have a distinctive objective function structure, composed of the mean of S evaluations of the loss function at different points of the training dataset. This characteristic implies that executing a single step of the classic Gradient method requires computing S different function evaluations with their corresponding subgradients. Specifically, one iteration can be expressed as:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \frac{\eta_t}{S} \sum_{i=1}^{S} \partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i),$$

where η_t is the step size at iteration t, see Section 1.3 for more details.

The stochastic Gradient method offers faster iterations by considering each sample individually in the training set. This variant suggests performing S steps for $i = 1, \cdots, S$ as follows:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \frac{\eta_t}{S} \partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

This variant allows faster iterations, but this comes at the cost of slower convergence, as it is more unstable.

It is also possible to strike a balance between the two approaches by considering the batched version, which involves fixing a batch size B_S and partitioning the set $\{1, 2, \cdots, S\}$ into subsets of size B_S (with eventually one smaller set if B_S does not divide the size of S). In all the cases where we aim to solve the Problem (2.1), using a Gradient method to compute a step, we need the gradient $\partial_{\boldsymbol{W}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}} \left[\mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}), \boldsymbol{y})\right]$. However, this is impossible to compute directly since the probability distribution $\mathcal{P}$ is unknown. Therefore, we want to approximate it using the estimator provided by some samples $(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim \mathcal{P}$ obtained using that probability. Rather than considering all the samples in the dataset, we can use only a subset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in B_S}$. This gives us the following approximation of the *true* gradient:

$$\frac{1}{|B_S|} \sum_{i=1}^{|B_S|} \partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

and so we can use this direction to perform a step. When $B_S = S$, this corresponds to the *classic* Gradient method, while taking $B_S = 1$, we see the stochastic version of the Gradient method as previously discussed in this subsection.

The *bias* of our estimator is defined as the difference between the *true* quantity and the estimation:

$$\mathbb{E}_{(\boldsymbol{x}_i,\boldsymbol{y}_i)\sim\mathcal{P}} \left[\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}} \left[\partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}), \boldsymbol{y})\right] - \frac{1}{|B_S|} \sum_{i=1}^{|B_S|} \partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i) \right].$$

An estimator is said to be *unbiased* if its bias is zero. In our case, obtaining an unbiased estimation may require an infinitely large number of samples. However, in practice, we typically only have access to a finite dataset, which limits our ability to generate additional samples. Consequently, even when using all available samples in the dataset at each iteration, we may still encounter a biased estimator, as new, previously unseen samples could appear.

Backpropagation The gradient $\partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i)$ is typically computed using Automatic Differentiation (AD) techniques [20]. Backpropagation is an efficient way to compute the gradient of the loss with respect to the model parameters, and it is the most used AD technique for training neural networks. Backpropagation has been independently reinvented several times [102], but it essentially consists of applying the Leibniz chain rule [48] to NN. The Leibniz chain rule states that, given $f : X \to Y$

and $g : Y \to Z$ differentiable functions, we can compute the gradient of the composed function $h = g \circ f$ for all $\boldsymbol{x} \in X$ as

$$h'(\boldsymbol{x}) = (g' \circ f)(\boldsymbol{x}) \cdot f'(\boldsymbol{x}).$$

In machine learning, this rule is used for differentiating the loss with respect to the model's predictions and the neural network with respect to its parameters. Applying that rule, we can compute the gradient of the loss with respect to the model parameters as:

$$\partial_{\boldsymbol{W}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i) = \partial_{\hat{\boldsymbol{y}}} \mathcal{L}(f_{\boldsymbol{W}_t}(\boldsymbol{x}_i), \boldsymbol{y}_i) \cdot \partial_{\boldsymbol{W}} f_{\boldsymbol{W}_t}(\boldsymbol{x}_i),$$

where $\partial_{\hat{\boldsymbol{y}}} \mathcal{L}$ denotes the gradient of the loss with respect to the prediction of the model. Similarly, to differentiate through the layers of a neural network (i.e., computing $\partial_{\boldsymbol{W}} f_{\boldsymbol{W}_t}$), we still apply this rule, seeing the network as the composition of the functions that define each layer, which are sequentially applied. For more details on backpropagation and automatic differentiation, we refer to [98].

2.1.2 Empirical risk optimization

Energy-Based Models (EBMs) [151] capture dependencies between input/output variables by assigning scalar energy to each configuration of the variables. The EBM approach offers a unified theoretical framework for various learning models, including traditional discriminative [259] and generative approaches [185], graph-transformer networks [279], conditional random fields [235], maximum margin Markov networks [242], and manifold learning methods [122, 168]. In contrast to probabilistic models, which require proper normalization (often involving computationally challenging integrals over all possible configurations of variables), EBMs do not face this constraint.

We are interested in energy models such as those presented in [71]. In this model type, the optimal prediction can be expressed as the minimum of a convex function given the dataset's sample. More precisely, given an input $\boldsymbol{x} \in \mathcal{X}$ and the associated label $\boldsymbol{y} \in \mathcal{Y}$, the model is expressed as:

$$\boldsymbol{y}^*(\boldsymbol{x}; \boldsymbol{W}) = \arg\min_{\boldsymbol{y}} E(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{W}).$$

Domke [71] also presents a method for differentiating a generic loss that depends on both the prediction $\boldsymbol{y}^*(\boldsymbol{x}; \boldsymbol{W})$ and the true label $\boldsymbol{y}$. In this setting, different types of losses are considered. The differentiation becomes simpler in particular cases. For a dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_i$ the easier and more straightforward possibility is the *energy loss* defined as:

$$E(\boldsymbol{y}_i, \boldsymbol{x}_i; \boldsymbol{W}).$$

However, this loss function is not ideal for training most architectures. Although it effectively reduces the energy of the correct output, it does not increase the energy for other possible outputs. This can lead to a collapsed solution where the energy becomes uniformly zero. The energy loss is effective only for architectures specifically designed so that lowering $E(\boldsymbol{y}_i, \boldsymbol{x}_i; \boldsymbol{W})$ inherently raises the energies of alternative outputs.

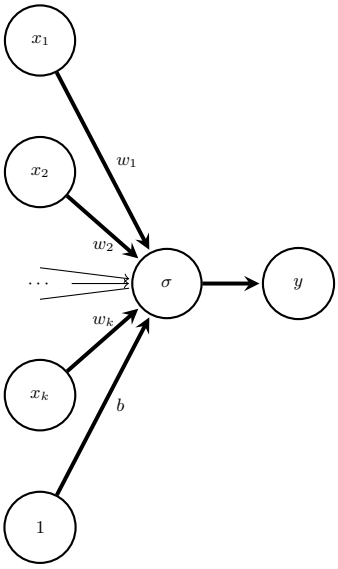

Figure 2.1: Perceptron. It receives as input k inputs $x_1, x_2, \cdots, x_k$, it computes a linear combination of those considering the weights $\{w_j\}_{j=1}^{k}$ (plus the bias contribution), and finally provides an output $y = \sigma\left(\sum_{j=1}^{k} w_j x_j + b\right)$ that consists of the application of an activation function σ to that linear combination.

Another, well-known, loss function in this domain is the *perceptron loss* defined as:

$$E(\boldsymbol{y}_i, \boldsymbol{x}_i; \boldsymbol{W}) - \min_{\boldsymbol{y} \in \mathcal{Y}} E(\boldsymbol{y}, \boldsymbol{x}_i; \boldsymbol{W}).$$

Further loss functions for solving energy-based learning problems are discussed in [151].

2.2 Learning Models

This section aims to provide formal definitions of the machine learning models employed in this Thesis. These models are widely recognized in the machine learning community, and for a more comprehensive overview, the reader is referred to [105].

2.2.1 Feed-Forward Neural Networks

Perceptron The key component of this architecture is the *perceptron*, illustrated in Figure 2.1. It consists of the composition of a linear function and a (non-linear) function $\sigma : \mathbb{R} \to I \subseteq \mathbb{R}$, called *activation function* [76]. Commonly used activation functions include: the rectified linear unit (ReLU) [107], the softplus [282], the hyperbolic tangent, and the sigmoid function. Figure 2.2 provides a visual comparison of these functions.

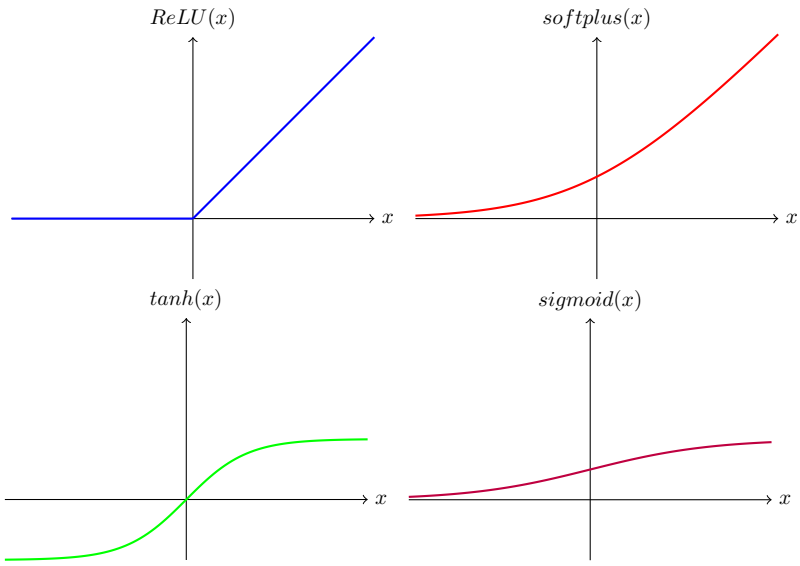

Figure 2.2: Four activation functions: ReLU (top-left), softplus (top-right), tanh (bottom-left), and the sigmoid (bottom-right).

Given an input $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^h$, the output of a *single perceptron* is given by:

$$y = \sigma(\sum_{j=1}^{h} w_j x_j + b),$$

where w_j for $j = 1, \cdots, h$ and b are the *parameters* of the network. The parameters $\boldsymbol{w}$ are called the *weights*, while b is called the *bias*. The bias is a *special* weight associated with the constant entry 1.

Multi-Layer Perceptron The *Multi-Layer Perceptron* (MLP), also known as a *Feed-Forward Neural Network* (FFNN), is the fundamental architecture in ML, often considered the simplest and one of the earliest models. The defining characteristic of the MLP is the one-way flow of information between its layers. Specifically, information in the model flows in a single direction: from the input nodes, through the hidden layers, to the output nodes, without any cycles or loops. This is in contrast to Recurrent Neural Networks, which feature a bidirectional flow of information, allowing data to cycle back through the network.

The Multi-Layer Perceptron, as used today, has a long history. The formal description of a layered network of perceptrons was first introduced in a psychology journal [215]. At that time, it was not yet explicitly thought to be trained, but rather it was presented just as a random probabilistic model to describe some cerebral functions. Some years later, the perceptron is combined into a more elaborate architecture to create a FFNN. In 1965, Ivakhnenko and Lapa [121] published the first deep-learning

feedforward network. However, this network had not yet employed stochastic gradient descent for training. In 1967, Amari [6] presented the first FFNN that was trained, demonstrating its ability to classify non-linearly separable classes.

One of the reasons for the increasing interest in neural networks lies in the Universal approximation theorems [58, 116]. These theorems state that, for any function f in a *specific* function space and a *specific* family of neural networks, there exists a sequence of neural networks from that family that converges to f according to a certain criterion. This means that the family of neural networks is dense in the function space. One of the most well-known versions states that feedforward networks with non-polynomial activation functions are dense in the space of continuous functions between two Euclidean spaces, under the compact convergence topology. Therefore, a sufficiently large MLP can approximate every continuous function between two Euclidean spaces, with a desired degree of accuracy.

These results are *existence* results, meaning they guarantee the existence of such a sequence but do not provide a method for finding it, nor do they assure that gradient descent with backpropagation will converge to the same limit. More recently, various studies have explored the convergence of gradient descent used to train neural networks, leveraging tools from probability and physics, such as mean-field theory [179].

In general terms, an MLP consists of L hidden layers, each composed of n_l perceptrons. Considering *fully connected* layers, i.e., connections between all the nodes of a certain layer (including the bias node) with its next layer, it is iteratively defined by:

$$\begin{aligned} \boldsymbol{h}^{(0)} &= \boldsymbol{x} \\ \boldsymbol{h}^{(l)} &= \sigma_l(\boldsymbol{W}_{l-1}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}_l) \qquad \text{for } l = 1, \cdots, L \\ \boldsymbol{y} &= \sigma_{L+1}(\boldsymbol{W}_L\boldsymbol{h}^{(L)} + \boldsymbol{b}_L) \end{aligned}$$

where, for $l = 0, 1, \cdots L$, $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{b}_l \in \mathbb{R}^{n_l}$ are the parameters of the network and σ_l are activation functions applied in parallel all along the components of the input vector. Each fully connected layer of a Multi-Layer Perceptron is also referred to as a *Dense* layer.

Figure 2.3 illustrates an example of a Multi-Layer Perceptron that receives an input vector $\boldsymbol{x}$ of size 2, sequentially elaborates it in two *hidden layers*, respectively composed of 4 perceptrons and 2 perceptrons, and provides an output of size 1.

2.2.2 Encoder-Decoder Models

Encoder-decoder models are employed when the architecture needs to handle inputs and outputs that may have different lengths. A first architecture q_ϕ, with parameters ϕ, learns a hidden representation $\boldsymbol{z}$ from the input data $\boldsymbol{x}$. This hidden representation is designed to describe the input data in such a way that it can be directly provided to the decoder to generate outputs of the correct size. Then the decoder architecture f_θ, with parameters θ, computes the output $\boldsymbol{y}$ from the hidden representation $\boldsymbol{z}$. The structure is schematically represented in Figure 2.4

A special case of the encoder-decoder model is the auto-encoder [145, 144, 16]. Auto-encoders are neural networks designed to encode input into a compressed and

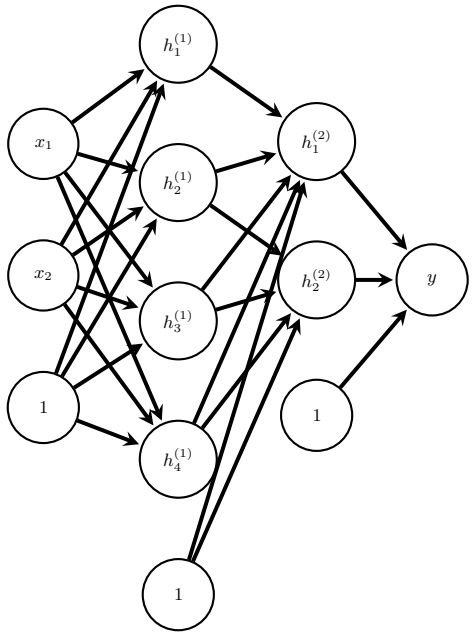

Figure 2.3: Example of Multi-Layer Perceptron. It receives as input a vector $\boldsymbol{x}$ of size 2, elaborating it in two *hidden layers* respectively composed of 4 perceptrons and 2 perceptrons, to provide an output of size 1. The output layer and each hidden layer have a constant input, also called *bias*.

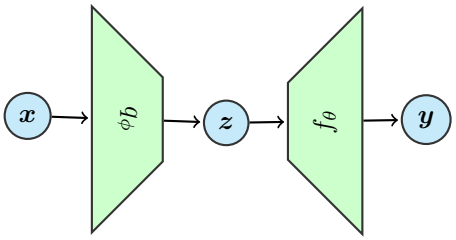

Figure 2.4: Encoder-decoder architecture.

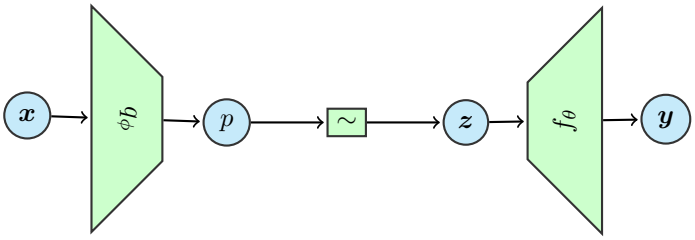

Figure 2.5: Variational encoder-decoder architecture, with probabilistic encoder. We denote by $\sim$ the operation of sampling one element $\boldsymbol{z}$ from the probability distribution p.

meaningful representation and then decode it back to resemble the original input as closely as possible. This architecture is first trained to reconstruct the representation. Subsequently, the encoder acts as an automatic feature extractor, with its parameters fixed, as it can then be used for other tasks by replacing the decoder with a new learnable architecture. The advantage of the automatic feature extraction of the encoder is that the encoder is sufficiently expressive to reconstruct the original data from it.

A particular type of auto-encoder is the *variational auto-encoder*, in which the encoder learns a probability distribution to sample the hidden representation rather than directly learning the hidden representation [136, 204]. Figure 2.5 schematically illustrates probabilistic encoder-decoders. This framework leads to variational auto-encoders when the output $\boldsymbol{y}$ is equal to the input $\boldsymbol{x}$.

In encoder-decoder architectures, the structure of the encoder can vary depending on the type of data being processed. For example, it can be an LSTM for sequence-structured data [161], Convolutional networks for images [182], or GNN for graphs [283, 129].

2.2.3 Graph Neural Networks

Graph Neural Networks are ML architectures designed to handle datasets *naturally* represented by an inherent graph structure. These networks are useful for datasets where each sample is represented by a different graph, as well as for datasets sharing a common graph structure, such as inferring predictions on the nodes of a graph. For example, this could involve predicting outcomes for users in a social network, where users are

represented as nodes, or making predictions about molecules, represented as graphs where atoms are nodes and bonds are edges [125]. Bronstein et al. [37] provide a first review of Graph Neural Networks and, more generally, the models for non-Euclidean domains, mainly focusing on Graph Convolutional Neural Networks. Today, a wide range of different Graph Neural Networks exists. In this section we focus on Graph Convolutional Neural Networks. The reader is referred to Wu et al. [265] for GNNs not based on convolution.

Graph Convolutional Neural Networks can be seen as a generalization of Convolutional Neural Networks in which we substitute the grid with a general graph. Let $G = (V, E)$ be a graph defined by its vertices $V = \{v_1, \cdots, v_{|V|}\}$ and edges E. We consider G an undirected graph, but Graph Neural Networks can also be used in directed graphs with minor adjustments. In the rest of this section, we will suppose that E also contains the self-loops, that is arcs that connect each vertex to itself. Although not mandatory, adding self-loops is a common practice in GNNs as it helps avoid losing the local information associated with the vertex during message passing.

Definition 2.2.1. We denote by $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$ the *adjacency matrix* of the graph, that is the matrix with components

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwhise} \end{cases}$$

This matrix represents the connectivity between the nodes in the graph.

Definition 2.2.2. We denote by $\boldsymbol{D} \in \mathbb{R}^{|V|\times|V|}$ the *degree matrix* of the graph, that is, the diagonal matrix with diagonal entries

$$D_{i,i} = deg(v_i) = \sum_{(j_1,j_2)\in E} \mathbf{1}_{\{j_1=i\}},$$

Let be $\boldsymbol{H}_0 \in \mathbb{R}^{|V|\times d_0}$ the input features matrix, where each row corresponds to the feature vector of a node. We denote by $\{d_i\}_{i=0}^{L+1}$ the dimensions of the nodes representation at each layer, where d_0 is the dimension of input nodes representations and d_L the one of the output representation. Similarly, $\boldsymbol{H}_l \in \mathbb{R}^{|V|\times d_l}$ is the matrix of node representation at the end of the layer l. The Graph Convolutional Layers can be expressed as:

$$\boldsymbol{H}_{l+1} = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}_l\boldsymbol{W}_l\right) \qquad \forall l = 0, \cdots, L$$

where $\boldsymbol{W}_l \in \mathbb{R}^{d_l\times d_{l+1}}$ is a matrix of trainable parameters and σ is a non-linear activation function applied element-wise to the matrix. The multiplication with the adjacency matrix allows sharing the information of the neighborhood, while the two matrices multiplication, with the squared inverse of the degree matrix, are used to *normalize* the received information related to the number of neighbors, preventing nodes with many neighbors from dominating the message passing.

In the case of a directed graph, the degree matrices are substituted with the input and output degree matrices, which are defined similarly but for the incoming and outgoing edges.

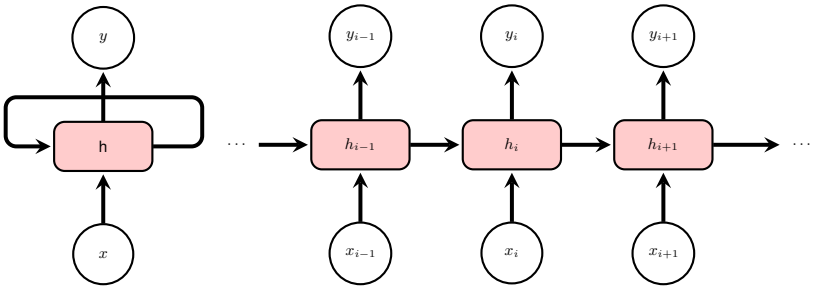

Figure 2.6: RNN general architecture (on the left) and unfolded representation (on the right).

A more general formulation for Graph Neural Networks uses the Message Passing Layer [96], where the feature $\boldsymbol{h}_u^{(l)}$ associated with the node u at layer l is still updated based on messages from its neighbors:

$$\boldsymbol{h}_u^{(l+1)} = \phi\left(\boldsymbol{h}_u, \bigoplus_{v \in N_u} \psi(\boldsymbol{h}_u^{(l)}, \boldsymbol{h}_v^{(l)})\right)$$

where $\bigoplus$ is a permutation invariant aggregation operator (such as the sum, the mean, or the max) and N_u is the set of neighbors of the node u, i.e., all the other nodes for which exists an edge from the former and u. The functions ψ and ϕ are two differentiable functions, which may include activation functions or more advanced neural networks with their trainable parameters.

2.2.4 Recurrent Models

Recurrent Neural Networks [278] (RNNs) are a class of learning architectures commonly used for processing data with a sequential structure, such as time series, text, audio, and video.

At each time step t, given an input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$, the output $\boldsymbol{y}_t$ of a *Sigma Recurrent cell* can be written as

$$\begin{aligned}\boldsymbol{h}_t &= \sigma(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}) \\ \boldsymbol{y}_t &= \sigma'(\boldsymbol{h}_t),\end{aligned}$$

where $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$ denote the weights of the network, $\boldsymbol{b}$ its bias, and σ and σ' are activation functions, which may differ. A major limitation of such simple RNN architectures is their inability to capture long-term dependencies, due to vanishing and exploding gradients. Gradient Clipping techniques [28, 100] can be considered to avoid gradients blowing up. While vanishing gradients are avoided by introducing *gates*.

Gates in RNNs control how information flows through the cells. They act as filters. They act as filters, deciding which parts of the information from the past are relevant and should be retained for future steps, which to discard, and which new information to incorporate.

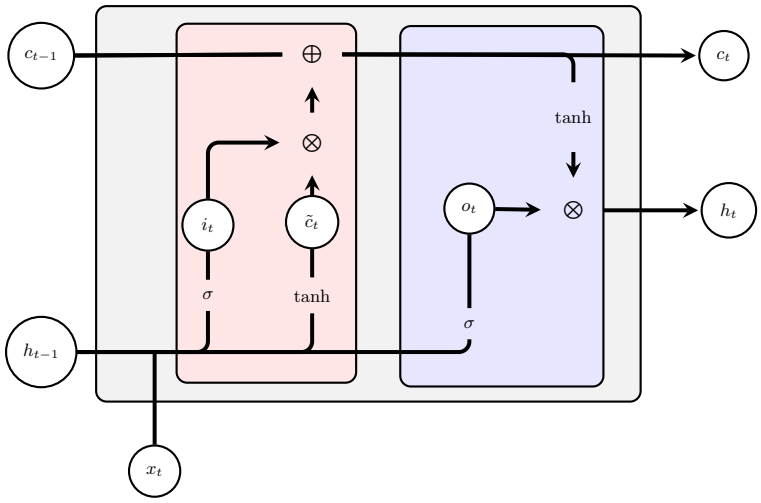

Figure 2.7: LSTM *without* forget gate.

In the next paragraph, we will focus on one type of RNNs addressing this problem and used in Chapter 5: the Long Short Term Memory (LSTM) [230, 114]. We emphasize the fact that Gated Recurrent Units [52, 50, 51] (GRU) provide an efficient alternative to LSTMs, as they generally reduce complexity while maintaining performance.

Long Short Term Memory LSTM networks were introduced to address the limitations of standard RNNs, particularly their inability to capture long-term dependencies due to the vanishing gradient problem. A common LSTM unit consists of a *cell state*, an *input gate*, an *output gate*, and an optional *forget gate*. An LSTM *without* forget gate is represented in Figure 2.7. The input gate receives as input the data $\boldsymbol{x}_t$ at time t, the current hidden state $\boldsymbol{h}_{t-1}$, and the current context $\boldsymbol{c}_{t-1}$. It updates the cell state as

$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \tilde{\boldsymbol{c}}_t \tag{2.3}$$

where

$$\begin{aligned} \boldsymbol{i}_t &= \sigma\left(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i\right) \\ \tilde{\boldsymbol{c}}_t &= \tanh\left(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i\right) \end{aligned}$$

are computed considering two separate dense layers with different activation functions. The output gate processes the input $\boldsymbol{x}_t$, the current hidden state $\boldsymbol{h}_{t-1}$, and the context $\boldsymbol{c}_t$ provided by the input gate to determine the next hidden state as:

$$\boldsymbol{h}_t = \boldsymbol{o}_t tanh(\boldsymbol{c}_t)$$

where

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o\right).$$

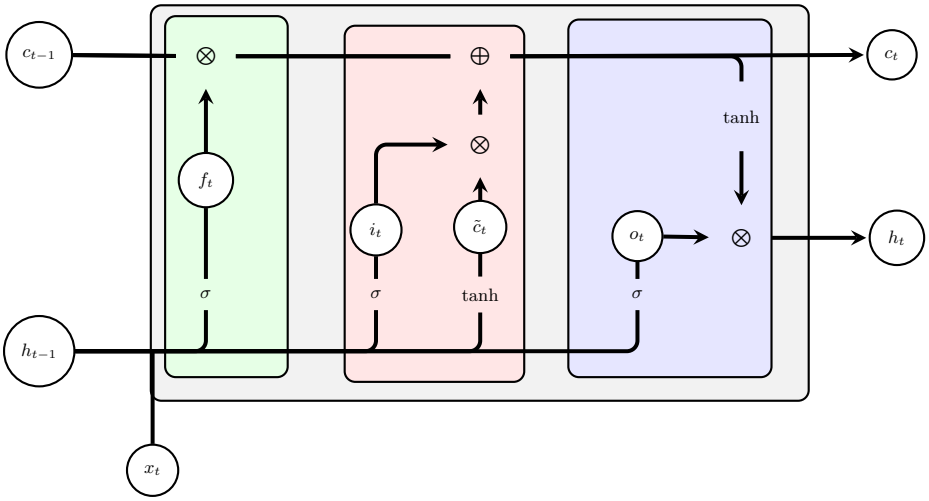

Figure 2.8: LSTM *with* forget gate.

LSTM with forget gate Forget gates [95] improve the original LSTM by allowing it to discard outdated information. It receives as input the data $\boldsymbol{x}_t$ at time t and the current hidden state $\boldsymbol{h}_{t-1}$ and it computes a *forget rate* $\boldsymbol{f}_t$ as

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f\right).$$

This factor is then used to compute the new cell state, instead of (2.3) as

$$\boldsymbol{c}_t = \boldsymbol{f}_t \cdot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \cdot \tilde{\boldsymbol{c}}_t.$$

Hence, when the value of a certain component of $\boldsymbol{f}_t$ is equal to 1, we keep the information, and when it is 0, we forget it.

2.3 Other Layers

This section provides technical details on other layers we use to define our architecture in Chapter 4. These layers help mitigate technical challenges such as vanishing gradients, exploding gradients, and overfitting when training large deep neural networks.

2.3.1 Residual Connections

Residual Connections [108] are a deep learning layer used in multi-layer neural network models to avoid vanishing gradient problems. Let $\boldsymbol{H} : \mathbb{R}^h \to \mathbb{R}^h$ be a function encoding a set of layers in the network with input $\boldsymbol{x} \in \mathbb{R}^h$. Applying the residual connections to this sub-network means considering as output:

$$\boldsymbol{x} + \boldsymbol{H}(\boldsymbol{x})$$

Note that $\boldsymbol{x}$ and $\boldsymbol{H}(\boldsymbol{x})$ should have the same dimension.

This operation can be interpreted not only as a connection within the network, but also as a layer that applies both the identity and the function $\boldsymbol{H}$ in parallel to the input $\boldsymbol{x}$, summing the two outputs. The sum is a common approach, but other operations, such as the mean, maximum, and minimum, can also be considered.

2.3.2 DropOut Layers

DropOut Layers [252] are a regularization technique in neural networks used to prevent overfitting [233]. Overfitting occurs when a network becomes overspecialized to fit the data in the training set, leadingto poor generalization to unseen data. Dropout layers mitigate this issue by randomly setting a fraction of the input components to zero with a probability determined as a hyperparameter.

2.3.3 Layer Normalization

Training state-of-the-art deep neural networks is computationally expensive. One way to decrease training time is to normalize neuron activities, which significantly accelerates the convergence during the training process in FFNNs [11]. Layer normalization computes the mean and variance for normalization from all summed inputs to the neurons in a layer for a single training case. This technique is highly effective at stabilizing the hidden state dynamics.

Given an input $\boldsymbol{x} \in \mathbb{R}^h$, we compute the statistics as:

$$\mu = \frac{1}{d}\sum_{j=1}^{h} x_j \;\text{ and }\; \sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{h}(x_j-\mu)^2}.$$

All hidden units in the same layer share the computed normalization terms μ and σ, but different training cases have distinct normalization values. The normalized output $\boldsymbol{y} \in \mathbb{R}^h$, of the same dimension as the input, is given by:

$$y_j = \frac{1}{\sigma}(x_j - \mu).$$

Many variants introduce a learnable weight g_j and a bias b_j for each input to apply after the normalization, leading to the output:

$$y_j = \frac{g_j}{\sigma}(x_j - \mu) + b_j.$$

2.3.4 Softmax

In certain cases, by leveraging the Karush-Kuhn-Tucker (KKT) conditions, discussed in Chapter 1, we can derive an explicit formulation of the solution as a function of problem parameters, allowing for direct differentiation with respect to these parameters. One well-known example is the *softmax* function [98, pp. 180-184]. Other examples will be presented in Chapter 3.

Let $\Delta_n = \{\boldsymbol{\theta} \in \mathbb{R}^n_+ | \langle \mathbf{1}, \boldsymbol{\theta} \rangle = 1\}$ denote the unit simplex in $\mathbb{R}^n$. Consider the optimization problem:

$$\max_{\boldsymbol{\theta} \in \Delta_n} \langle \boldsymbol{\alpha}, \boldsymbol{\theta} \rangle. \tag{2.4}$$

To obtain a differentiable approximation of the solution to (2.4), we introduce a log-entropy regularization term $H(\boldsymbol{\theta}) = -\langle \boldsymbol{\theta}, \log(\boldsymbol{\theta}) \rangle$, modifying the objective function as follows:

$$\max_{\boldsymbol{\theta} \in \Delta_n} \langle \boldsymbol{\alpha}, \boldsymbol{\theta} \rangle + \beta H(\boldsymbol{\theta}). \tag{2.5}$$

The corresponding Lagrangian function is:

$$\mathcal{L}(\boldsymbol{\theta}, \lambda, \boldsymbol{\mu}) = \langle \boldsymbol{\alpha}, \boldsymbol{\theta} \rangle + \beta H(\boldsymbol{\theta}) + \lambda(\langle \mathbf{1}, \boldsymbol{\theta} \rangle - 1) - \langle \boldsymbol{\mu}, \boldsymbol{\theta} \rangle$$

Since the Slater condition is satisfied, the KKT conditions are both necessary and sufficient for optimality. Let $(\boldsymbol{\theta}^*, \lambda^*, \boldsymbol{\mu}^*)$ be the optimal solution of the problem with the associated Lagrangian multipliers for the simplex constraints and the slack variable for the non-negative solution. The KKT conditions for the Problem (2.5) yield:

$$\begin{cases} \boldsymbol{\alpha} - \beta \log(\boldsymbol{\theta}^*) - \boldsymbol{\mu}^* - (\beta + \lambda^*)\mathbf{1} = 0 \\ \lambda^*(\langle \mathbf{1}, \boldsymbol{\theta}^* \rangle - 1) = 0 \\ \theta_i^* \mu_i^* = 0 \qquad\qquad \forall i = 1, \cdots, n \end{cases}$$

From the first condition, we derive:

$$\theta_i^* = \exp(\frac{\alpha_i - \lambda^* - \beta - \mu_i^*}{\beta}). \tag{2.6}$$

Since $\boldsymbol{\theta}^*$ is strictly positive, we obtain $\mu_i^* = 0$ for all the components due to the slackness conditions.

Furthermore, λ^* should be non-zero. Otherwise, the simplex constraints are not respected. Substituting $\boldsymbol{\theta}^*$ in the second condition, we obtain:

$$\sum_{j=1}^{n} \exp(\frac{\alpha_j - \lambda^* - \beta}{\beta}) = 1$$

leading to:

$$\lambda^* = \beta \log \left(\sum_{j=1}^{n} \exp(\frac{\alpha_j}{\beta}) \right) - \beta.$$

Using this equality in Equation2.6 we obtain:

$$\theta_i^* = \frac{\exp(\frac{\alpha_i}{\beta})}{\sum_{j=1}^{n} \exp(\frac{\alpha_j}{\beta})}.$$

Thus, we derive an analytical formulation of the optimal solution $\boldsymbol{\theta}^*$ as a function of the problem parameters $\boldsymbol{\alpha}$, enabling the differentiation:

$$\partial_{\alpha_j} \theta_i^* = \theta_j^*(\delta_{i,j} - \theta_i^*).$$

where $\delta_{i,j}$ is the Kronecker delta that is equal to one if $i = j$ and zero otherwise. This formulation facilitates integration within machine learning frameworks that employ automatic differentiation techniques.

Chapter 3

Machine Learning for Optimization

In Chapter 1, we discussed the fundamentals of Optimization, while Chapter 2 provided an overview of machine learning techniques and architectures used in this thesis. In this chapter, we present a literature review focused on the intersection of these two fields, specifically exploring the applications of machine learning in Optimization. This analysis provides context and facilitates a better understanding of the results presented in Chapter 4 and Chapter 5. The *reverse* perspective, namely the application of Optimization techniques in machine learning, is beyond the scope of this thesis. For readers interested in this topic, we refer to [232].

Many recent studies have leveraged machine learning to tackle Combinatorial Optimization problems. Various machine learning techniques, including supervised, unsupervised, and reinforcement learning, have been successfully applied to optimization tasks. In their survey, Lodi et al. [27] classify the applications of ML in optimization into three main approaches.

The first involves employing automated methods to search for the most appropriate parameter configuration for a parametrized algorithm (or to select an algorithm from a specific algorithmic family) to enhance performance. This application falls outside the scope of this survey. For readers interested in this topic, we refer to the surveys [223, 120]. Some other surveys exist on particular sub-domains of the Algorithm Configuration problem (ACP), such as *Algorithm Selection* [143, 130], *Hyperparameter Optimization problem* [77], specifically applied to machine learning hyper-parameter selection [270, 277] or jointly algorithm selection and hyper-parameters optimization for machine learning [169, 227, 119, 284].

A second family of approaches consists of directly predicting a solution to an optimization problem. This approach, referred to as *End-to-End Learning*, aims to replace traditional solvers with machine learning models. However, this approach typically generates heuristic solutions, as even predicting feasible solutions remains challenging, especially with complex constraints.

Finally, machine learning can be used alongside optimization algorithms to improve

performance. For example, ML can choose the variable to branch on or the node to process inside a Branch and Bound (B&B) algorithm. ML alongside optimization may, in some cases, overlap with ACP, when decision-making is reduced to hyperparameter tuning. However, a key distinction regards the timing of these decisions. Most Algorithm Configuration approaches lead to choosing the parameters *before* solving the optimization problem, and they choose configurations as preprocessing, before running an algorithm. Instead, machine learning alongside optimization also works in an iterative context in which an optimization problem should be repeatedly solved during the resolution process. When configuration is performed *online*, i.e., during the algorithm's execution, the machine learning component is typically simple and more aligned with Reactive Search [19] or Bayesian Networks [201].

The next sections are structured as follows. Following the categorization introduced above, Section 3.1 discusses machine learning techniques alongside Optimization solvers, while Section 3.2 discusses End-to-End Learning. In particular, Section 3.1 focuses on machine learning for decomposition techniques, emphasizing Lagrangian relaxation, which is a key topic in this thesis and a case of application of all the contributions of this thesis. Thus, this section offers a valuable tool for contextualizing the approaches presented in Chapter 4 and Chapter 5 within the broader research that aims to use machine learning alongside Optimization Techniques, with a particular focus on decomposition techniques. At the same time, the approach presented in Chapters 4 and 5 aims to directly predict vectors of Lagrangian multipliers in an end-to-end fashion, aligning it more closely with the methodologies discussed in Section 3.2 in terms of framework structure. Therefore, this section serves as a valuable reference point for understanding and positioning our methods within the broader spectrum of learning-based solution strategies.

3.1 Machine Learning Alongside Optimization Solvers

Optimization solvers play a critical role in solving complex combinatorial problems, and *machine learning* (ML) has emerged as a powerful tool to enhance their efficiency. Traditional optimization methods rely on predefined heuristics and rule-based approaches, which often require significant expert tuning and may not generalize well across different problem instances. ML techniques can address these limitations by learning patterns from data, automating key decision-making steps, and improving overall solver performance.

In the next sections, we provide further details on how ML can be integrated with optimization techniques. Section 3.1.1 focuses on machine learning inside Branch & Bound (B&B), a fundamental algorithm for the exact solution of MILPs. Section 3.1.2 then examines decomposition techniques, which are closely related to the contributions of this thesis. Finally, Section 3.1.3 provides further details on Lagrangian relaxation, as it is the setting of the approach presented in Chapter 4 and represents a potential application for the approaches presented in Chapter 5.

3.1.1 Machine Learning for Branch and Bound

B&B is a widely used method for solving combinatorial optimization problems, particularly MILPs. Since this algorithm is not central to the main results of this thesis, we do not describe this algorithm in detail, and we refer the reader to [262]. At a high level, B&B explores a tree of potential solutions by systematically branching on decision variables and pruning unpromising subproblems.

ML has been successfully applied to various components of optimization solvers, with notable contributions in Branch and Bound (B&B) and Cutting Plane method. In this area, ML has been used to enhance variable selection, node exploration, use of heuristics during the execution, and cut generation. These applications have received significant attention in the literature due to their fundamental role in Mixed-Integer Linear Programming (MILP) and other combinatorial optimization problems.

Variable selection determines which variable to branch on when splitting a node. A key heuristic for this task is *Strong Branching*, which evaluates multiple variable choices and selects the one leading to the best-bound improvement. Node selection determines which node to explore next in the search tree. Cut selection refers to the process of choosing which cutting planes (or cuts) to add to the current relaxation of an Integer Linear Program. Cutting planes are additional constraints respected by all the feasible integer solutions of the node and help eliminate fractional solutions from the relaxation.

Variable Selection, Node Selection and Heuristics in B&B ML techniques have been developed to improve each of these aspects. For Variable Selection, strong branching is computationally expensive. ML models, including supervised learning and reinforcement learning, have been used to predict strong branching scores, significantly reducing computational costs [4, 131, 90, 280]. In particular, Gasse et al. [90] present a *bipartite graph representation* of an optimization instance, which is also largely used to provide it as input to a neural network, and it will be presented in detail in Section 4.1.2 of Chapter 4. Similarly, node selection has a crucial impact on solver efficiency. ML models, such as Graph Neural Networks (GNNs), have been used to learn node selection policies that minimize tree size and solve problems faster [148, 212]. ML has also been applied to decide when and how to perform heuristics [132, 49] to improve performance without increasing computational costs and to predict partial solutions early in the search, allowing solvers to focus on promising regions of the search space [189, 275]. These advances demonstrate that ML can make B&B solvers significantly more efficient, leading to faster solutions for large-scale combinatorial optimization problems.

ML for Cutting Planes Cutting Plane methods, presented in Chapter 1, are essential for tightening relaxations in MILPs and improving MILP solvers' convergence. The effectiveness depends largely on which cuts to generate and which to keep. ML has been leveraged in several ways: to predict the impact of a given cut on solver performance to select the most effective cuts [15, 67, 118], to learn policies for selecting Gomory cuts [240] improving solver efficiency compared to traditional heuristics, to order candidate cuts based on their expected contribution to improve the bound [258], and to adjust cutting plane strategies dynamically based on problem characteristics [30].

Further Applications Beyond B&B and cutting planes, ML has been applied to several aspects of optimization solvers, including: Algorithm Configuration and Tuning, where the goal is to automate the selection of solver parameters for MILPs and heuristics [119, 270], Constraint Propagation in Constraint Programming solvers [93, 208, 142, 164].

In the next sections, we delve deeper into ML for decomposition techniques and Lagrangian relaxation, which are crucial for large-scale optimization and directly relevant to the contributions of this thesis.

3.1.2 Machine Learning in Decomposition Techniques

Mathematical programming formulations for solving MILPs often involve many constraints or variables, making direct solution approaches impractical. When the number of constraints and variables exceeds what standard methods like the Simplex algorithm can efficiently handle, decomposition techniques become necessary. However, selecting an appropriate decomposition strategy is itself a challenging task, as different formulations can lead to widely varying computational performance.

Reformulation and Automated Decomposition Kruber et al. [147] propose a supervised learning approach to determine when reformulation is beneficial and which decomposition technique is most suitable among multiple options.

Machine learning has been explored to guide and automate decomposition strategies. Basso et al. [18] use data-driven techniques to develop automatic decomposition methods by analyzing static properties of MIP instances and their connection to effective decomposition structures. Their results indicate that supervised learning can reveal valuable patterns that assist in selecting decomposition strategies.

Benders' Decomposition Benders' Decomposition [24] is an optimization technique for large-scale mathematical programming problems with a block structure. Complicating variables are isolated in a Master problem, while subproblems, often defined per scenario, are solved independently. The algorithm proceeds iteratively, generating feasibility or optimality cuts from subproblem duals to iteratively refine the Master problem until convergence. In the context of Benders Decomposition, the integration of machine learning has received relatively limited attention, with most contributions focusing on specific application domains. Machine learning has been used in a classification task to select the Benders cut to add in the Master problem formulation for the Multi-Stage Stochastic Transmission Expansion Planning problem [34], for Wireless Resource Allocation [152], and in a branch-cut-and-Benders for the dock assignment and truck scheduling problem in cross-docks [192]. In [152], machine learning is also used for regression tasks to predict the continuous performance indicators for each cut. However, this information is still leveraged to guide the selection of Benders cuts to add to the formulation. Another usage involves generating an initial set of feasible points for which Bender cuts are derived and incorporated into the Master problem as a warm start strategy [183]. While these contributions highlight interesting use cases, they are not directly related to the contributions developed in this thesis. For these reasons, we

chose not to introduce a dedicated subsection on Benders Decomposition, and instead briefly summarize these works here to acknowledge their relevance.

Machine Learning for Column Generation

Column Generation is a fundamental decomposition technique, particularly useful in solving MILPs with formulations composed of a large number of variables. However, it often suffers from slow convergence due to the need for many iterations. At each iteration, the pricing subproblem identifies one or more columns with negative reduced costs that can be added to the Master problem to improve the objective value. While selecting multiple columns per iteration can accelerate the process, adding too many columns increases the problem size, potentially slowing down overall computation. Efficiently selecting the most promising subset of columns remains a key challenge.

Choosing Columns To address this issue, Morabit et al. [187] propose an ML-based approach that formulates column selection as a classification problem. Their model predicts whether a column should be added to the solution pool, thereby reducing computational overhead. Other researchers have explored reinforcement learning techniques for this task. Chi et al. [47] use a learning-by-experience framework, where the reward decreases as the total number of Column Generation iterations increases.

Using Dual Information In some cases, dual information can be useful for learning. Babaki and Jena [13] employ an expert-guided reinforcement learning approach in which an expert rule selects columns whose dual solution values in the Reduced Master problem are closest to the optimal dual solution in the Reduced Master problem. During the Column Generation process, the number of variables grows while the number of constraints remains the same, at least for standard approaches. Furthermore, as highlighted in Chapter 1, Lagrangian relaxation is strictly related to Column Generation through duality theory. For this reason, we put in Section 3.1.3, also, all the approaches that learn dual variables to warm-start.

3.1.3 Machine Learning for Lagrangian Relaxation

Lagrangian relaxation is a fundamental technique in mathematical programming, particularly for decomposing large-scale optimization problems. Machine learning has been explored as a tool to enhance Lagrangian-based approaches by predicting dual multipliers, improving convergence, and reducing computational costs. In Table 3.1 we present an overview of the literature related to the interaction of ML and Lagrangian relaxation. The acronyms for the target problems are: Graph Coloring problem (GCP), Unit Commitment (UC), Cutting Stock problem (CSP), Markov Random Fields (MRF), Graph Matching (GM), Independent Set (IS), Knapsack problem (KP), Multi-Dimensional Knapsack problem (MDKP), nonlinear Resource-Constrained Production and Inventory Planning problem (RCPIPP), Multi-Commodity Network Design (MCND), and Generalized Assignment (GA). QAPlib [38] is the benchmark dataset for the quadratic assignment problems used in the combinatorial optimization community.

Links to Stabilized Column Generation Lagrangian relaxation and Column Generation are closely related, as discussed in Chapter 1. Generating variables in the primal can be viewed as generating constraints in the dual. Consequently, predicting dual variables for the Reduced Master problem in a Column Generation approach corresponds to predicting a Lagrangian multipliers vector.

Several approaches presented here [229, 146, 234] focus on predicting a dual stabilization point for a stabilized version of the Column Generation. All three approaches use the predicted vector as initialization for a Column Generation approach, which is known to suffer from severe oscillations in the dual variables of the Reduced Master problem. Two of these approaches [229, 146] employ supervised learning techniques, minimizing the mean squared error between the prediction and the optimal multipliers, interpreted here as dual variables of the Reduced Master problem in the final iteration. In contrast, the other one [234] directly optimizes the Lagrangian bound associated with the prediction.

The strategy of directly optimizing the Lagrangian bound is largely used [2, 1, 234, 241, 64, 197] as it enables training without requiring an optimal solution. Moreover, the subgradient of the Lagrangian bound, with respect to the Lagrangian multipliers vector, can be computed once the subproblem with the predicted Lagrangian multipliers is solved, making training feasible.

The main differences among the approaches directly optimizing the Lagrangian bound in the learning problem lie in the choice of model, feature representation, and the specific optimization problem addressed. Shen et al. [229] apply their approach to the Graph Coloring problem, using two different neural network architectures. The first is a parallel feed-forward neural network that extracts statistical features representing interactions of dual variables. The second is a Graph Neural Network based on a fully connected graph, where each node corresponds to a dual variable, allowing the model to learn structural relationships among them. Kraul et al. [146] focus on the Cutting Stock problem and compare two different ML models. The first is a feed-forward neural network that takes a fixed-size vector encoding the information of the entire instance as input and uses it to predict the Lagrangian multipliers vector. The second model takes as inputs a feature vector corresponding to a single component of the multipliers and predicts its associated value. This model, applied independently to each component, can generalize to instances of different sizes, but it lacks contextual information related to the other components. Their results suggest that models using full instance information achieve higher accuracy, whereas those with sparse feature representations generalize

Paper	Authors	Loss function		Features Size		Application			Target Problem	
		Supervised	Unsupervised	Fixed	Variable	Two-stage	SCG	Other	Generic	Tested on
[229]	Shen et al.	x		FFNN	GNN		x		x	GCP
[234]	Sugishita et al.		x	FFNN-RF		x	x			UC
[146]	Kraul et al.	x		FFNN	FFNN		x			CSP
[2]	Abbas & Swoboda		x		BDDs			x	x	MRF, GM, CT, QAPlib
[1]	Abbas & Swoboda		x		BDDs+GNN[90]			x	x	CT,GM,IS, QAPlib
[190]	Nair et al.		x	RL		x			x	KP and FL
[241]	Tanneau & Hentenryck		x	FFNN	*			x	x	MDKP, RCPIPP
[64]	Demelas et al.		x		GNN[90]			x	x	MCND, GA
[197]	Parjadis et al.		x		GNN			x		TSP

Table 3.1: Table summarizing the approaches presented in this section for Lagrangian relaxation.

better across problem sizes. Sugishita et al. [234] apply ML to predict initial dual values for Column Generation procedures in large-scale Unit Commitment problems. In this setting, similar instances should be solved repeatedly with small variations in the data. Their model takes as input only the vector parameterizing the unknown components in the right-hand side of a family of constraints, requiring no specialized feature extraction. However, this simplicity limits model generalization to instances of different sizes. They experiment using a random forest and a feed-forward neural network with skip connections.

Two-Stage Stochastic MILPs Other approaches focus more directly on Lagrangian relaxation. One application is in two-stage stochastic MILPs, where decomposition is often necessary for tractability. Nair et al. [190] introduce a reinforcement learning approach to predict Lagrangian multipliers that yield tight bounds across different second-stage scenarios. Their framework also incorporates a learnable local search solver that jointly optimizes two policies: one for generating an initial feasible solution and another for iteratively refining it. By leveraging contextual features, their method generalizes across problem instances and improves the effectiveness of dual decomposition techniques. Also, the already presented paper [234] focuses on Stabilized Column Generation in this two-stage setting.

Other Approaches Parjadis et al. [197] apply ML to predict Lagrangian multipliers for the Traveling Salesman problem (TSP), focusing on the one-tree Lagrangian relaxation [111]. Their approach uses a GNN trained on the graph representing the TSP, incorporating node features. While their study is specific to TSP, the authors suggest their framework could be extended to other problems. More recently, Tanneau and Van Hentenryck [241] propose a general framework to learn Lagrangian multipliers for conic programming. Their method predicts one set of dual variables given input data defining a conic problem using a neural network. A conic projection layer then computes a conic-feasible partial-dual solution, which is completed into a full dual-feasible solution. The authors emphasize that their approach, while not yet tested with GNNs, could be extended to variable-size instances using techniques presented in Section 4.1.2.

Lagrangian decomposition and Binary Decision Diagrams Another promising ML-driven approach applies Lagrangian decomposition in conjunction with binary decision diagrams. Abbas et al. [2] develop a parallel Lagrange decomposition method for solving 0–1 integer linear programs, where they represent subproblems as binary decision diagrams. Their approach minimizes synchronization overhead and takes full advantage of GPU parallelism. Abbas et al. [1] further improve this method by integrating Graph Neural Networks (GNNs), using the instance representation presented in Section 4.1.2. They modify a Lagrange decomposition-based algorithm to be (partially) differentiable, enabling end-to-end training. While the solver remains partially non-differentiable, learning is performed by running the algorithm over a limited number of iterations, with the gradient computed only for the final updates. This strategy effectively refines the last adjustment, enhancing performance through training, while preserving the algorithm's theoretical properties, such as dual feasibility and non-decreasing lower bound

Figure 3.1: Example of a framework where an ML prediction is followed by solving an optimization problem, which can be seen as a layer.

guarantees.

3.2 End-to-End Learning

Machine learning can also be used to directly generate a potentially near-optimal solution to a given problem; these approaches are referred to as *End-to-End* approaches. Often, End-to-End techniques are more likely to succeed when tailored to specific problems, where domain knowledge can be exploited to guide the learning process toward feasible solutions. Our review focuses on general-purpose methods; therefore, we omit End-to-End learning methods specifically designed for particular optimization problems. The interested reader may refer to [209, 256] for a survey on the usage of machine learning to solve the Traveling Salesman problem, to [256] for the Vehicle Routing problem, and to [272] for the Hydro Unit Commitment problem. The survey [178] explores the use of reinforcement learning for combinatorial optimization and examines its application to specific problems: the Traveling Salesman, Maximum Cut, Bin Packing, Minimum Vertex Cover, and Maximum Independent Set.

Developing an End-to-End framework, particularly for general-purpose applications, presents several challenges. First, many optimization problems have structured solutions that must satisfy many constraints. Ensuring the feasibility of the solutions when they are generated by a learning model presents a significant challenge. In addition, these methods typically do not guarantee optimality, which complicates efforts to provide theoretical guarantees. A well-designed model architecture can help mitigate some of these issues by improving prediction quality and incorporating constraints. However, it does not completely resolve the problem, as ensuring constraint satisfaction and proving optimality often requires additional techniques, such as hybrid approaches or post-processing corrections. Finally, encoding problem instances as input for a learning model remains a complex task, requiring thoughtful representation choices.

Differentiating through the Resolution of an Optimization Problem Differentiation is a fundamental component of training neural networks, as it is crucial in the back-propagation algorithm used to update model parameters.*Automatic differentiation* [21, 103] plays a crucial role in enabling end-to-end training by allowing gradients to be computed through the optimization layer.

The machine learning community has shown increasing interest in enabling back-propagation through otherwise non-differentiable operations. For example, quantization in neural networks [269] reduces the precision of the model weights and activation functions, typically using a lower bit-width format for floating-point representations. The Straight-Through Estimator [274] allows gradients to flow through quantized activations

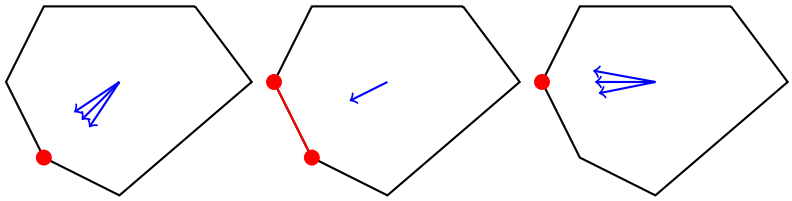

Figure 3.2: Changes in the optimal solution of an optimization problem as the objective function varies. The blue arrows indicate the direction of the objective function, while the red dots and lines represent the optimal solution(s). In the left and right figures, we observe that the optimal solution remains unchanged despite variations in the objective function's direction. However, in the center figure, when the objective function becomes perpendicular to a constraint boundary, the problem admits multiple optimal solutions, as illustrated by the red line segment.

by treating the quantization function as an identity during back-propagation. The Spigot Estimator [202] introduces an incremental greedy optimization approach for structured argmax problems, enabling differentiable training through structured prediction by approximating gradients via a sampling-based technique. The Gumbel-Softmax [123] distribution is a Continuous relaxation of the categorical distribution, enabling gradient propagation through the sampling step.

One of the main challenges in using machine learning to solve optimization problems is ensuring that predictions satisfy feasibility constraints and exhibit the required combinatorial structure. Standard machine learning models struggle to enforce all constraints of an optimization problem. In connection with Section 3.1.3, some approaches leverage Lagrange duality to improve constraint satisfaction [79, 248]. This approach does not guarantee feasibility. Hence, another strategy is to integrate an optimization problem within the framework, allowing constraints to be explicitly considered. In this case, a key difficulty is enabling differentiation through architectures that alternate neural network layers with optimization problem resolution. Typically, this type of framework predicts some parameters, such as the coefficients in the objective function, of an optimization problem, which is then solved using some external solver, as illustrated in Figure 3.1. The solution is then used in the loss function. The challenge in this approach is the computation of the solution gradient with respect to the parameters defining the problem structure.

A core challenge arises from the fact that small changes in the problem data can cause a discontinuous shift of the solution. In particular, the solution may remain unchanged while certain conditions hold, but once those conditions are violated, it can abruptly *jump* to a different point. This phenomenon is illustrated for a Linear Program in Figure 3.2, where the solution initially coincides with a single vertex and remains stable under small variations in the objective function. However, when the objective function is perpendicular to the constraint, the solution is no longer unique, as any point satisfying the constraints with equality is an optimal solution. In this scenario, every slight change in the objective function can shift the solution to a different vertex.

Connections to thesis Contributions and Broader Context In the rest of this section, we focus on approaches more closely related to the contributions presented in Chapters 4 and 5. Section 3.2.1 discusses a specific subclass of Energy-Based Models, which simplifies differentiation by imposing a particular structure on the loss function. Another line of research involves differentiating through the solution process of an optimization algorithm itself. More precisely, Section 3.2.2 explores techniques that derive an analytical formulation of the gradient passing through the KKT conditions. This is not the only possibility to differentiate through an optimization layer. Section 3.2.3 introduces additional relevant techniques for differentiating through an optimization layer that are not directly used in this thesis, but are important for contextualization. These include Learning Stochastic Smoothing, Differentiable Relaxation, and Finite Differences.

Finally, some methods modify the resolution process of the iterative algorithm to integrate machine learning components. This will be discussed in detail in Section 3.2.4 with a particular focus on the Gradient Descent, aligning with the approaches presented in Chapter 5.

3.2.1 Energy Based Models

A particular example where a model's prediction is obtained as the solution to an optimization problem is provided by Energy-based Models [151], already introduced in Section 2.1.2. These models capture dependencies between input/output variables by assigning scalar energy to each configuration of the variables. The prediction is then found by minimizing the energy associated with the corresponding input. The parameters of the learning task influence the shape of the energy and, consequently, the prediction too. While EBMs were originally introduced in the neural network literature, recent work has extended them to optimization-oriented settings, which we explore in this subsection.

Energy Models for Structured Outputs Some works focus on producing outputs with a certain combinatorial structure [22]. However, enabling deep learning techniques to solve tasks requiring nontrivial reasoning, such as algorithmic computation, remains challenging. Du et al. [74] follow the idea that humans solve such tasks with iterative reasoning. They present a framework for iterative reasoning based on an energy landscape over all possible outputs, where each step of the reasoning process is implemented as an energy minimization step to find a minimal energy solution. They test their approach on graphs and continuous domains, as well as on optimization problems such as the Shortest Path problem. Building on this idea, Du et al. [75] introduce a novel framework that combines Diffusion models [113] with energy-based optimization. They propose to learn energy functions to represent the constraints between input conditions and desired outputs. After training, thanks to the diffusion mechanism, the framework can dynamically adapt the number of optimization steps during inference based on problem difficulty, enabling it to solve problems outside its training distribution.

Energy Models for Stochastic Optimization In a different line of research, Kong et al. [140] propose a machine learning method for stochastic optimization problems using energy-based models. It explicitly parameterized the original optimization problem using a differentiable optimization layer based on energy functions. To better approximate the optimization landscape, they propose a coupled training objective that uses a maximum likelihood loss to capture the optimum location and a distribution-based regularizer to capture the overall energy landscape. For stochastic optimization problems involving unknown parameters, this approach demonstrates superior performance to the standard two-stage predict-then-optimize pipeline.

Links with Lagrangian Relaxation This subdomain is of particular interest as it is closely related to Lagrangian relaxation, discussed in Section 1.1. In this case, the learning problem can be viewed as a mean approximation of the Lagrangian Dual, where the energy bound corresponds to the objective value of the Lagrangian subproblem. This perspective represents a novel contribution of this thesis, offering a new interpretation of dual learning as an instance of energy-based modeling. We will show in Chapter 4 how the problem of learning Lagrangian multipliers can be cast within this family of approaches.

3.2.2 Differentiating the KKT system resolution

An approach to enable differentiation through optimization problems is to exploit the structure of the Karush-Kuhn-Tucker (KKT) conditions, presented in Chapter 1, which characterize the optimality of constrained problems. When applicable, these conditions allow for implicit or explicit gradient computation with respect to problem parameters.

Explicit Formulation of the Solution In certain cases, the KKT conditions allow the *explicit reformulation* of an optimization problem's solution, making its dependence on the problem parameters more direct. A simple example is the softmax function, discussed in Section 2.3.4 of Chapter 2. More recently, Martins and Astudillo introduced an alternative approach called *sparsemax* [176]. The key difference lies in the type of regularization: while softmax relies on log-entropy regularization, sparsemax employs quadratic regularization. Both softmax and sparsemax offer a relatively simple case of study, as they allow the solution to be explicitly expressed as a function of the parameters in an analytical form.

Quadratic Programming and Iterative Resolution of KKT Conditions In other settings, the resolution of the KKT system can require an iterative process. These approaches are closely related to the ones presented in Section 3.2.4. For instance, Sambharya et al. [222] tune the initialization of a quadratic problem, by reformulating the KKT conditions and solving them using an iterative algorithm that only relies on linear operations and projections into a cone, Douglas-Rachford (DR) splitting [73]. This approach allows the use of machine learning to predict a good initialization, knowing that the resolutive algorithm will be used to obtain the final solution. Another work [8] still focuses on quadratic programs, and it passes through the KKT conditions

to integrate quadratic programs as individual layers in larger end-to-end trainable deep networks.

Beyond Quadratic Programming and Regularization Tricks KKT conditions are not limited to quadratic problems. Donti et al. [72] use them to differentiate the solution of general stochastic programming problems. In the case of linear programming (LP), the optimal solution can be expressed as a solution to a linear system defined by the active constraints. This demonstrates the connection between the constraint matrix and the optimal solution, and that this connection is differentiable. In general, the solution can be non-unique or vary in discontinuous ways while changing the objective function. To address this issue, Wilder et al. [261] propose to add a quadratic regularization term to an LP and then use the framework proposed by Amos and Kolter [8] for differentiation. Moreover, they further extend the use of KKT multipliers for approximations of discrete problems by leveraging Lagrangian duality. Similarly to [261], Mandi and Guns [173] differentiate through the solution of an LP using the logarithmic barrier regularization.

Homogeneous Self-Dual Instead of differentiating through the resolution of the KKT conditions, it is possible to consider the homogeneous self-dual formulation [267] of the LP, and to show that the relation between the interior point step direction and the corresponding gradients required for learning. In the forward pass, Mandi and Guns [173] use the existing homogeneous interior point algorithm [9] to solve the LP, while the backward pass reduces to the resolution of a linear system.

3.2.3 Additional Approaches to Differentiate through an Optimization Layer

Considering energy-based models can simplify differentiation through optimization components. In some cases, the Karush-Kuhn-Tucker (KKT) conditions also enable differentiation through the solution of an optimization problem. Beyond the use of KKT conditions, several alternative approaches have been developed to enable differentiation through optimization layers.

Stochastic Smooting Some approaches [29, 60] use *Stochastic Smoothing* to differentiate through discrete optimization. This technique involves perturbing the problem's data and considering the mean value of the associated solutions. By using averages over multiple samples, the function becomes *smoother*, making it more amenable to gradient-based optimization. The goal then became to learn a probability distribution that maximizes the likelihood [80] of the observed solutions. At inference time, this probability can then be used to choose the *most likely* solution.

Differentiable Relaxations Another possibility is to consider *Differentiable Relaxations*. Some examples are provided by: using the Semi-definite relaxation of the Maximum Satisfiability problem [255], using submodular function approximations [69] to derive differentiable surrogates, adding a regularization to the objective function

[173, 261] to enforce smoothness or writing the descent direction for an ILP as a linear combination of integer points on a selected basis [199].

Finite Differences It is possible to approximate the loss gradient using *finite differences*. Vlastelica et al. [174] approximate gradients using the loss difference between the solution of a problem in the forward pass and a slightly perturbed version of the former. Paulus et al. [198] develop a flexible framework, called Lagrangian Proximal Gradient Descent (LPGD), based on the assumption that strong duality holds and rewriting the problem using the Lagrangian formulation. LPGD unifies and generalizes various state-of-the-art contemporary optimization methods, including: Direct Loss Minimization [106], Blackbox Back-propagation [174], Implicit Differentiation by Perturbation [70], Identity with Projection [221], Smart Predict then Optimize [99] and different frameworks based on Fenchel-Young losses [33]. These works [198, 174] use both a temperature parameter that determines a trade-off between smoothness (and so more informative gradients) and tightness of the approximation.

3.2.4 Iterative Amortization - Learning the Gradient Descent

In amortization techniques [7], the goal is to reduce the computational cost of *iterative* optimization methods by learning a direct mapping from inputs, in our case problem instances, to approximate solutions. This often involves training an inference network to quickly produce a high-quality approximate solution that would otherwise require costly iterative optimization. The definitions of amortization techniques and end-to-end learning often overlap but originate from different purposes. Amortization techniques can be iterative or not, but their core purpose is always to replace an iterative process where similar problems are solved repeatedly. End-to-end also aims to provide a solution directly, but it encompasses further frameworks that do not necessarily fall under amortization. The optimization algorithm being replaced is not always iterative, and the underlying frameworks can vary. An example is the predict-then-optimize framework, in which a neural network predicts some of the parameters of the original instance, and the resulting optimization problem is then solved during both training and inference.

Learning an Optimizer The research domain of amortized optimization is somehow referred to as *Learning-to-Optimize* [43, 157, 44], and the associated methods are called *amortized optimization* methods. The term *Learning-to-Optimize* is closely linked to meta-learning, where the goal is to learn an optimizer, often parameterized as a neural network, capable of efficiently updating another model's parameters. For this reason, this area is called *Learning-to-Learn* [244].

Unrolling Techniques Many amortized frameworks using an iterative structure for predictions often require some unrolling technique for back-propagation. *Unrolling*, also known as *deep unfolding*, involves reformulating the algorithm execution to enable the gradient computation in the backward pass. The distinction between amortization and unrolling is sometimes unclear in the literature. Some works use the term *unrolling*

in a broader sense, including hybrid models where parts of an optimization algorithm are replaced by learnable components. In this thesis, we use the term unrolling to describe the process of reformulating an algorithm's execution to enable the construction of a computation graph, which is essential for the backward pass. For a comprehensive survey on unrolling techniques, we refer to [186].

Unrolling becomes particularly interesting for Learning-to-Optimize approaches that do not have access to the optimal solution. In such cases, learning is framed as an *objective-based* problem, which requires the ability to differentiate the objective function with respect to the model's output.

Amortization and Unrolling Applications While unrolling and related Learning-to-Optimize methods are also widely applied in broader contexts such as Meta-Learning [224], a comprehensive review of those areas lies beyond the scope of this thesis. Therefore, we briefly mention here a few additional applications where unrolling plays a central role, even though they are not the primary focus of our work.

Relevant areas that are worth mentioning are Variational Inference [127] and Variational Autoencoders [135], Sparse Coding [193], Multitask Learning [40], fixed point computations for convex optimization [214], Optimal Transport [250], and Policy learning [237].

Unrolling has also been applied to a variety of algorithms beyond classical first-order methods. For example, it has been extended to second-order optimization techniques, including Newton and quasi-Newton methods [158, 266], Differential Equations [166, 165, 281] and Interior Point method [31, 55] (specifically for image reconstruction), and Frank-Wolfe algorithm [200, 162] (respectively for financial index tracking and sparse coding). We briefly note that some research focuses on unrolling iterative methods for Structured Prediction for Energy Networks (SPENs) [23], making a connection to Section 3.2.1. This work introduces an end-to-end learning framework where the energy function is trained discriminatively by back-propagating through gradient-based prediction. However, the approach is applied to image denoising rather than optimization-related problems.

Gradient Descent

Gradient Descent and its variants are widely used in machine learning to optimize neural network parameters. Early research of meta-learning in this domain focused on learning updates for a neural network (trained using Gradient Descent), considering simple evolutionary rules [26, 25] and then more elaborate neural networks [219]. Some works [276, 115] jointly train with Gradient Descent both the networks for the learning problem and the one for the meta-learning task. However, these approaches still struggle to scale to modern architectures with tens of thousands of parameters. To address this limitation, Andrychowicz et al. [10] introduce an LSTM-based optimizer that operates coordinate-wise on the network parameters. This design allows different behavior on each coordinate by using separate activation functions for each parameter, but making the optimizer invariant to parameter order, as the same update rule is independently applied to each coordinate.

Theoretical foundations More recently, research has emerged to provide stronger theoretical foundations for Learning to Optimize. In [239], Takabe et Wadayama provide a theoretical interpretation of the learned step size of deep-unfolded Gradient Descent. Liu et al. [163] analyze the Learning to Optimize framework from a more *theoretical* perspective. Drawing inspiration from theoretical convergence proof of Gradient-Descent-type resolution methods, they construct a mathematics-inspired framework that is broadly applicable and generalized well to out-of-distribution problems. Their work addresses separately the smooth case and the extension to convex non-smooth objective functions. In the latter case, they highlight the importance of a *proximality operator*. Hence, their approach is somehow related to the *Proximal Bundle method*, presented in Chapter 1.

Problematic of the domain One major challenge in this area is the meta-gradient instability: the gradient associated with tuning unrolling parameters can often explode or vanish. Wang et al. [257] address this by providing theoretical guarantees for the problem of tuning the step size for quadratic loss.

Another key challenge is that the learned optimizer may fail to generalize well to unseen tasks. This problem is solved using different techniques: optimizing the parameters of a distribution over the optimizer parameters [181], using a novel hierarchical RNN architecture with minimal per-parameter overhead [260] or training an architecture inspired both by [10] and [181] on thousands of tasks [180].

This trade-off between generalization and stability often stems from a fundamental tension: using short unrolls introduces *truncation bias*, which can harm generalization, while using long unrolls increases the risk of *gradient explosion*, which compromises training stability. Wu et al. [264] analyze this problem mathematically and show that while a greedy scheduler tends to decay the learning rate drastically to reduce the loss along high curvature directions, the optimal schedule maintains a high learning rate to ensure steady progress along low curvature directions, ultimately achieving lower final loss. Lv et al. [170] demonstrate that a random scaling over the optimizer parameters and the addition of a convex regularization allow the optimizer to remain effective on more epochs than those considered in the meta-training. This also generalizes to tasks involving training different neural networks. Another effective technique for this problem is a curriculum-based training [45], where the optimizer gradually increases the length of unrolled iterations.

Similarities with Bundle Method All of the above approaches fundamentally adopt the structure of Gradient Descent as the base for their learning framework. In the second approach presented in Chapter 5 we developed a similar amortized optimization method inspired by the Bundle structure. The benefits of using a proximal operator when optimizing a convex non-smooth function have already been demonstrated [163]. A further connection with the *Bundle method* can be found in [124], where Ji et al. propose a meta-learning approach to coordinate learning in master-slave distributed systems. Their method aggregates the gradients for Gradient Descent, improving the scalability of distributed learning.

It is worth noting, in connection with the approaches discussed in Section 3.1.3,

that some unrolled algorithms follow primal-dual schemes, particularly variations of primal-dual hybrid gradient (PDHG) and the alternating direction method of multipliers (ADMM). For instance, Shen et al. [228] apply a similar approach to min-max problems, where Lagrangian relaxation itself represents a special case. Their method employs iterative updates inspired by gradient-based techniques, using two parallel LSTMs to update the variables associated with the min and max components separately. The loss is computed based on the differences in the min-max function across iterations, considering updates for only one set of variables while keeping the other fixed.

3.3 Conclusions

Both approaches presented in this thesis fall within the domain of the learning method applied to Lagrangian relaxation, presented in Section 3.1.3. In this chapter, we contextualize that family of approaches in the domain of machine learning alongside Optimization approaches, in Section 3.1, with a particular focus on decomposition techniques, presented in Section 3.1.2, and Column Generation.

Nevertheless, the approaches presented in Chapter 5 can be applied more generally to learning for convex optimization. In particular, the second approach discussed in Chapter 5 could be extended to non-convex problems. Although this extension is beyond the scope of this thesis, it presents interesting possibilities for meta-learning, aligning with the methods described in Section 3.2.4. Specifically, it could enable predictions over the bundle space rather than directly over the model parameters, facilitating the adaptation of learning approaches to larger neural networks.

All the approaches presented in Chapter 4 and Chapter 5 directly minimize the Lagrangian bound of the predicted Lagrangian multipliers vector. As a result, they can be framed as Energy-based Models, presented in Section 3.2.1. Section 3.2 shows that this provides a structured way to differentiate through Optimization Layers. This method simplifies the process by introducing a specific structure on the loss function. However, this approach is not mandatory, and we dedicate further details to the possibilities provided to the analytical formulation due to the KKT conditions, discussed in Section 3.2.2, as these conditions play a crucial role in the first approach presented in Chapter 5. Alternative solutions to this issue are discussed in Section 3.2.3.

The next two chapters are dedicated to the main contributions of this thesis.

Chapter 4

Learning Lagrangian Multipliers

Lagrangian relaxation is a widely used technique for solving combinatorial optimization problems, as discussed in Chapter 1. The function $\boldsymbol{\pi} \mapsto LR(\boldsymbol{\pi})$ that associates a Lagrangian multiplier vector $\boldsymbol{\pi}$ to the value of the subproblem $LR(\boldsymbol{\pi})$ is piecewise linear and concave. Typically, this function is optimized using subgradient-based algorithms, as discussed in Chapter 1. However, as the dimension of $\boldsymbol{\pi}$ increases, the number of iterations required to obtain high-quality dual solutions tends to grow substantially, rendering the Subgradient method computationally expensive. This motivates the use of more sophisticated approaches, such as the Bundle method.

To fix notations, we consider instances ι as Formulation 1.1, that is:

$$
\begin{aligned}
(P) \qquad & \min_{\boldsymbol{x}} \ \boldsymbol{w}^\top \boldsymbol{x} \\
& \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\
& \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d} \\
& \boldsymbol{x} \in \mathbb{R}^m_+ \times \mathbb{N}^p
\end{aligned}
$$

Lagrangian subproblemis obtained by dualizing difficult constraints (1.1b) $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$ and penalizing their violation with Lagrangian multipliers (LMs) $\boldsymbol{\pi} \geq \mathbf{0}$:

$$
\begin{aligned}
\big(LR(\boldsymbol{\pi})\big) \qquad & \min_{\boldsymbol{x}} \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{\pi}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \\
& \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{d} \\
& \boldsymbol{x} \in \mathbb{R}^m_+ \times \mathbb{N}^p
\end{aligned}
$$

In the remainder of the chapter, we focus on the case where inequalities are dualized. However, the same approach can be applied, with minor modifications, to dualize equalities. Indeed, dualizing equalities is *simpler*, and the necessary adjustments to the framework will be explicitly discussed in the corresponding section on page 62.

In this chapter [1], we introduce a state-of-the-art encoder-decoder neural network designed to efficiently compute *good* duals $\boldsymbol{\pi}$ from the Continuous relaxation (CR) solution. Our method leverages a probabilistic encoder built on a Graph Neural Network (GNN). This encoder takes as input a given Mixed-Integer Linear Problem (MILP) instance ι, along with its primal and dual of the CR solutions, and produces an embedded representation of the instance. Each dualized constraint is mapped to a high-dimensional dense vector. A deterministic decoder then reconstructs scalar duals from these embeddings.

A crucial property of our approach is that it is unsupervised, as the Lagrangian Dual function $LR(\boldsymbol{\pi})$ provides a natural loss function that eliminates the need for manually labeled training data. We will see that our loss function is closely related to Energy-Based Models and Energy Losses, as discussed in Chapters 2 and 3.

It is important to note that our approach applies to compact MILPs where the bound provided by the Lagrangian relaxation is tighter than that of the CR. This advantage stems from the fact that both primal and dual of the CR solutions are incorporated into the GNN input, enhancing the quality of the learned duals. Note that this approach does not apply to non-compact formulations. Moreover, the problem in this setting involves more than just solving the CR. Indeed, the graph convolution associated with the bipartite graph representation, discussed in Section 4.1.2, also becomes computationally intractable due to the large number of variables and constraints.

4.1 Overall Architecture

Iterative algorithms for optimizing Lagrangian multipliers (LMs), such as the Subgradient method (SM) [207, Chap 5.3] or the Bundle method (BM) [112, 150] presented in detail in Chapter 1, can benefit from initializing LMs. While a common choice is to set them to zero, a more effective approach, widely adopted by the Optimization community, leverages the bound provided by the CR and its dual solution. This bound, and its dual solution, is often computationally inexpensive for compact MILPs. Specifically, optimal values of the CR dual variables identified with the constraints dualized in the Lagrangian relaxation can be understood as LMs. In many problems of interest, these LMs are not optimal and can be improved by SM or BM. We leverage this observation by trying to predict a deviation from the LMs corresponding to the CR dual solution.

The architecture is depicted in Figure 4.1. We start from an input instance ι of MILP P with a set of constraints for which the Lagrangian subproblem is easy to compute, then solve the CR and obtain the corresponding primal and dual solutions. In the following $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_D, \boldsymbol{\lambda}_{ND})$ denotes the dual solution of the CR, where $\boldsymbol{\lambda}_D$ is the subvector associated with the dualized constraints in the Lagrangian relaxation and $\boldsymbol{\lambda}_{ND}$ with the non-dualized constraints.

The input enriched with the CR solutions is then passed through a probabilistic encoder, composed of three parts: *(i)* the input is encoded as a bipartite graph in a similar way to [90], also known as a *factor graph* in probabilistic modeling, and initial graph node feature extraction is performed, *(ii)* this graph is fed to a GNN in charge of

[1] Code in JULIA available at `https://github.com/FDemelas/Learning_Lagrangian_Multipliers.jl`.

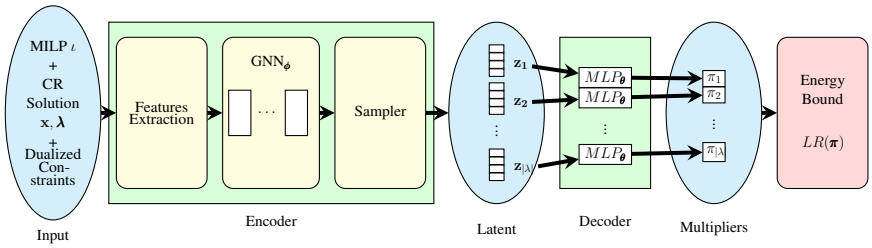

Figure 4.1: Overall Architecture. The model computes a Lagrangian Dual solution from the bipartite graph representation of an MILP and its CR solution. First, the MILP is encoded by a GNN, from which we parameterize a sampler for constraint representations. These representations are then passed through a decoder to compute Lagrangian multipliers.

refining the node features by taking into account the structure of the MILP, *(iii)* the last layer of the GNN is used to parameterize a distribution from which vectors $\boldsymbol{z}_c$ can be sampled, providing a latent representation for each dualized constraint c.

The decoder then translates each $\boldsymbol{z}_c$ to a positive LM $\pi_c = \lambda_c + \delta_c$ by predicting a deviation δ_c from the CR dual solution variable λ_c associated to the dualized constraint c. Finally, the predicted LMs can be used in several ways, in particular to compute a Lagrangian bound or to warm-start an iterative solver.

4.1.1 Objective

As seen in Section 3.1.3, numerous works [234, 2, 1, 190, 241, 197] directly optimize the Lagrangian subproblem value associated with the predicted multipliers, following an approach similar to our work [64], presented in this chapter.

The approaches that directly optimize the Lagrangian subproblem value associated with the predicted multipliers can be cast as an Energy-Based Model [151], discussed in Section 3.2.1 of Chapter 3.

The Energy-Based Models aims to construct an energy E, parametrized by a vector $\boldsymbol{\pi}$, such that, given a context ι, a *good* prediction is provided by

$$\min_{\boldsymbol{x}} E(\boldsymbol{x}, \iota; \boldsymbol{\pi}). \tag{4.1}$$

In the Lagrangian relaxation setting, the context can be seen as determined by an instance ι, and the energy E is the objective value of the Lagrangian subproblem, given a Lagrangian multiplier vector $\boldsymbol{\pi}$ and a feasible solution $\boldsymbol{x}$ of the Lagrangian subproblem.

We observe that all Energy-Based Models require solving an optimization problem to produce an output. This aligns perfectly with the framework of Lagrangian relaxation, where, for each choice of Lagrangian multipliers, we must solve the corresponding Lagrangian subproblem to obtain both the associated solution and the corresponding bound. Hence, the *final* prediction of our model, casting it as an Energy-Based Model, is the solution $\boldsymbol{x}$ of the Lagrangian subproblem and not the Lagrangian multipliers

vector $\boldsymbol{\pi}$. However, the latter provides a parametrization of the Energy associated with the Lagrangian subproblem (4.1) bound, and the learning task consists of finding a good Lagrangian multipliers vector $\boldsymbol{\pi}$ for each instance ι. More precisely, the learning problem consists of choosing the parameters of a learning model that predicts a vector of Lagrangian multipliers for each given instance.

A *classic* loss function is the energy loss:

$$E(\boldsymbol{x}^*, \iota; \boldsymbol{\pi}),$$

where $\boldsymbol{x}^*$ represents the optimal labels associated with ι. In our case, $\boldsymbol{x}^*$ is the primal solution of the subproblem obtained using the optimal Lagrangian multipliers $\boldsymbol{\pi}^*$. However, in our setting, applying this formulation may be less precise, as the subgradient used for training depends on the solution $\boldsymbol{x}$. Using a fixed solution $\boldsymbol{x}^*$ could result in missing information crucial for updating the problem.

Another widely used loss function is the *perceptron loss*:

$$E(\boldsymbol{x}^*, \iota; \boldsymbol{\pi}) - \min_{\boldsymbol{x}} E(\boldsymbol{x}, \iota; \boldsymbol{\pi}).$$

This loss is designed such that the first term provides an upper bound for the second.

In the Lagrangian setting, the first term can be replaced with $E(\boldsymbol{x}^*, \iota; \boldsymbol{\pi}^*)$, as the optimal configuration $\boldsymbol{\pi}^*$, is theoretically known. Meanwhile, in the generic Energy-Based Models, finding $\boldsymbol{\pi}^*$ is not trivial. This substitution simplifies the optimization process, leading to directly maximizing the bound provided by the solution of the Problem (4.1).

The reason why we do not use classical loss functions as in other Energy-Based Models lies in the concavity of the Lagrangian function with respect to the multipliers. Specifically, for a generic Energy-Based model, as we have no convexity/concavity guarantees for E as a function of $\boldsymbol{\pi}$, the same choice made for Lagrangian relaxation can lead to an energy that increments all the possible predictions $\boldsymbol{x}$, whether they are optimal or not. While a deeper theoretical analysis of this behavior could be insightful, it falls outside the scope of this thesis.

We train the network's parameters in an end-to-end fashion by maximizing the average Lagrangian bound $LR(\boldsymbol{\pi})$ obtained from the predicted LMs $\boldsymbol{\pi}$ over a training set. This can be formulated as an empirical risk optimization problem or as an Energy-Based Model [151] with latent variables, where the Lagrangian bound acts as the (negative) energy associated with the coupling of the instance and subproblem solutions. The LMs, or more precisely, their high-dimensional representations, serve as the latent variables.

To improve the duality gap, we aim to maximize LR, which provides a natural measure of prediction quality. Given an instance ι, our objective is to learn latent representations $\boldsymbol{z}$ of the LMs that maximize the Lagrangian bound:

$$\max_{\boldsymbol{\phi}, \boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\cdot|\iota)} \left[LR([\boldsymbol{\lambda}_D + f_{\boldsymbol{\theta}}(\boldsymbol{z})]_+; \iota) \right].$$

Here $q_{\boldsymbol{\phi}}$ is the probabilistic encoder, mapping each dualized constraint c in ι to a latent vector $\boldsymbol{z}$ computed by independent Gaussian distributions, $f_{\boldsymbol{\theta}}$ is the decoder mapping

each[2] $\boldsymbol{z}_c$ to its corresponding LM deviation $\delta_c = f_{\boldsymbol{\theta}}(\boldsymbol{z}_c)$ from the CR dual value λ_c, and $[\cdot]_+$ is the component-wise softplus function. We can observe that the function $[\cdot]_+$ is used only to ensure non-negative multipliers, as we are dualizing inequality constraints. In the case of dualizing equality constraints, this function must be omitted.

We can observe that this objective has the following properties amenable to gradient-based learning:

1. $LR(\boldsymbol{\pi})$ is bounded from above: optimal LMs $\boldsymbol{\pi}^*$ maximize $LR(\boldsymbol{\pi})$ over all possible LMs, that is $LR(\boldsymbol{\pi}^*) \geq LR(\boldsymbol{\pi})$ for any $\boldsymbol{\pi} = [\boldsymbol{\lambda}_D + f_{\boldsymbol{\theta}}(\boldsymbol{z})]_+$. Moreover, $LR(\boldsymbol{\pi})$ is a concave piece-wise linear function. In other words, all optimal solutions will give the same bound. This tells us that the chosen loss is a *good* choice for our task, as it correctly represents the original problem.

2. It is straightforward to compute a subgradient with respect to parameters $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} LR([\boldsymbol{\lambda}_D + f_{\boldsymbol{\theta}}(\boldsymbol{z})]_+; \iota) \approx \left(\frac{\partial [\boldsymbol{\lambda}_D + f_{\boldsymbol{\theta}}(\boldsymbol{z})]_+}{\partial \boldsymbol{\theta}} \right)^\top \nabla_{\boldsymbol{\pi}} LR(\boldsymbol{\pi}; \iota) \tag{4.2}$$

   To be formally correct, we can consider some subgradient generalization, as the Clarke subgradient, presented in Chapter 1. The theory in this case tells us that, considering the Chain rule, we can obtain a vector that possibly is not a generalized subgradient of the function in the left-hand side of Equation 4.2. However, Equation 4.2 holds with equality at the points where the function is differentiable, so almost everywhere. This makes this gradient approximation a good practical choice for a learning problem.

   The Jacobian on the left of Equation 4.2 is computed via backpropagation, while $LR(\boldsymbol{\pi}; \iota)$ is simple enough for a subgradient to be given analytically. Provided that $\bar{\boldsymbol{x}}$ is an optimal solution of the relaxed Lagrangian subproblem of ι associated with $\boldsymbol{\pi}$, we derive:

$$\nabla_{\boldsymbol{\pi}} LR(\boldsymbol{\pi}; \iota) = \boldsymbol{b} - \boldsymbol{A}\bar{\boldsymbol{x}}. \tag{4.3}$$

   This means that to compute a subgradient for $\boldsymbol{\theta}$, we first need to solve each subproblem. Since subproblems are independent, this can be done in parallel.

   Informally, the subgradient corresponds to the violation of the relaxed constraint for an optimal solution of the Lagrangian subproblem associated with the provided Lagrangian multipliers.

3. For parameters $\boldsymbol{\phi}$, we again leverage function composition and the fact that $q_{\boldsymbol{\phi}}$ is a Gaussian distribution, so we can approximate the expectation by sampling and use the reparameterization trick [135, 226] to perform standard backpropagation. More specifically, the parametrization trick is based on the fact that the value of a Gaussian random variable Y with law $\mathcal{N}(\mu, \sigma^2)$ can be rewritten as $Y = \mu + \sigma X$ with $X \sim \mathcal{N}(0, 1)$ a normal random variable. The mean μ and the variance σ are the outputs of a neural network model, and we can then differentiate all the parameters using standard backpropagation techniques, as all the stochastic

[2] With a slight abuse of notation, we use function $f : \mathbb{R}^m \to \mathbb{R}^n$ on *batches* of size p to become $\mathbb{R}^{m \times p} \to \mathbb{R}^{n \times p}$.

components remain parameter-free in the normal variable X. We implement $q_{\boldsymbol{\phi}}$ as a neural network, described in detail in the following section, returning a mean vector $\boldsymbol{\mu}_{c,\boldsymbol{\phi}}$ and a variance vector $\boldsymbol{\sigma}_{c,\boldsymbol{\phi}}$ for each dualized constraint c, from which a sampler returns a representation vector $\boldsymbol{z}_c$ component-wise as:

$$\boldsymbol{z}_c = \boldsymbol{\mu}_{c,\boldsymbol{\phi}} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}_{c,\boldsymbol{\phi}} \tag{4.4}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, Id_h)$ is a Gaussian vector with independent identically distributed components. For numerical stability, the variance is clipped to a safe interval [220].

Dualizing Equalities

Let us note that the framework can also be applied when equalities are dualized, a case explicitly considered in the numerical results. In this setting, we consider constraints of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. The Lagrangian subproblem keeps the same formulation, and the only difference is that the multipliers are now unconstrained, that is $\boldsymbol{\pi} \in \mathbb{R}^n$, rather than being restricted to non-negative values, that is $\boldsymbol{\pi} \in \mathbb{R}^n_+$.

This results in a simplification within our framework, as we no longer need to account for a function that ensures non-negativity of the multipliers. The learning problem can thus be formulated as:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\cdot|\iota)} \left[LR(\boldsymbol{\lambda}_D + f_{\boldsymbol{\theta}}(\boldsymbol{z}); \iota) \right].$$

with the only difference being that the projection operator $[\cdot]_+$ is no longer required.

4.1.2 Instance Bipartite Graph Representation

In this subsection, we will talk about how we can encode an Optimization instance to be provided as input to a machine learning model. What we present here is related to the works presented in Chapter 3, but we choose to talk in this Chapter for two different reasons. First, it is related to how we encode one instance and not the machine learning method used for a specific task. Secondly, different research works use this technique for various proposals that are categorized in different sections of Chapter 3.

Choosing the appropriate feature representation to encode an instance is essential but non-trivial, because this representation should be provided as input to a neural network model. Many problem-specific approaches focus on constructing ad-hoc feature vectors.

Machine learning frameworks based on Graph Neural Networks(GNNs) are popular and many surveys exist [39, 117, 249, 253, 203]. However, the majority of these approaches focus on problems defined by an underlying graph. For example, the Traveling Salesman problem [177] (TSP), Maximum Independent Set [206] (MIS), Multi-Commodity-Capacitated-Network Design problem [91] (MCND), and many other Optimization problems are described using a graph structure. An increasing number of works focus on approaches that can be used for a larger class of problems, and in this case, the construction of the feature vector should be completely automated.

In their analysis, Alvarez et al. [5] identified three essential properties that a set of features should possess. The first is that feature representation (and ML model) should

allow for generalize across instances of varying sizes. The other two are related to invariance properties concerning irrelevant changes within the instance (such as row or column permutations) and to parameter rescaling, such as costs, right-hand sides, or the coefficient matrix.

In the following, we will discuss a particular feature representation of the instance becoming increasingly popular in the domain.

In [39], Cappart et al. provide a survey on using GNNs in Combinatorial Optimization. The bipartite graph representation of the instance, described in Gasse et al. [90], provides a representation for a linear program input-output that can allow automatic feature extraction.

The primary advantage of this representation is that it respects the first two properties of the feature representation proposed in [5], that is:

1. the number of features needs to be independent of the size of the problem instance,

2. the features should be invariant with respect to irrelevant changes in the problem, such as row or column permutation.

Meanwhile, the third property:

3. the developed features need to be independent of the scale of the problem, i.e., if the costs, coefficient matrix, or RHS vector are multiplied by some factor, the features should remain identical.

The third property depends only on the chosen node features or how we perform the convolution over the graph. This representation allows us to consider datasets with different sizes in terms of variables and constraints. Furthermore, it enables the model to share information between variables and constraints.

Consider the following linear problem:

$$\begin{aligned} &\min_x \sum_{i=1}^{n} c_i x_i \\ &s.t. \sum_{i=1}^{n} a_{ij} x_i \leq b_j \quad \forall j \in \{1, \cdots, m\}. \end{aligned}$$

We can construct a bipartite graph to represent this problem, considering a node for each variable and a node for each constraint. If a variable appears in a constraint with a non-zero coefficient ($a_{ij} \neq 0$), we create an edge between the variable node i and the constraint node j. The weight of the edge is set to the coefficient a_{ij}.

In the original work by Gasse et al. [90], this representation is used for the task of variable selection in the B&B tree. The approaches developed for this particular task have been discussed in more detail in Section 3.1. They used Markov Random Trees [251] to represent the decision process of the B&B, while the bipartite graph representation is used at each node of the branching tree to represent the instance and compute a probability distribution over variables to choose the next variable to branch on.

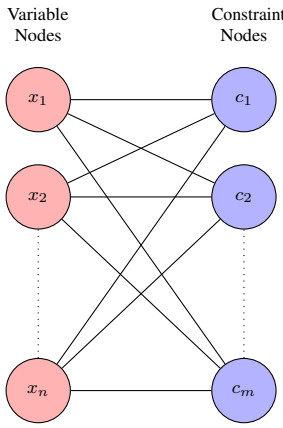

Figure 4.2: Bipartite graph representation of an instance

The research community has shown increasing interest in this representation. Recently, Chen et al. [46] proposed a theoretical analysis in which they showed that using the bipartite graph representation as input, GNNs can effectively distinguish between linear programming problems with different characteristics. Additionally, their study demonstrated that GNNs can accurately approximate the feasibility, optimal objective value, and optimal linear programming solution with minimal errors, particularly on finite datasets or within compact domains.

This representation has also been applied to other tasks, such as selecting the *good* subset of columns at each iteration of the Column Generation process as a classification problem [187] and using Reinforcement Learning [47], achieving a high-quality variable partial assignment [189], learn the node selection [160] in a Branch & Bound (B&B), even again warmstart B&B by finding an integer feasible solution [133] and learn a reformulation of linear programming (LP) [159]. Recently, some works have used this representation for data construction, producing a deep generative framework for MILP instances [92, 271]. We also use this technique in our work [64], described in detail in this chapter, to learn a vector of Lagrangian multipliers.

To our knowledge, the bipartite graph representation of Gasse et al. [90] is the unique instance representation that can be used in machine learning to represent a general MILP instance and can handle a dataset composed of instances of different sizes, in terms of number of variables and constraints, and with equivariance properties. We are aware of only one other minor modification of this representation, presented by Ding et al. [67], where the MIP instance is encoded as a tripartite graph, which extends the bipartite graph as [90] with a further part composed to a single node to denote the objective function, connected to all the other nodes in the graph.

4.1.3 Encoding and Decoding Instances

Encoder One of the challenges in machine learning applications to Combinatorial Optimization is that instances have different input sizes. So, the encoder must be able to

cope with these variations to produce high-quality features. Of course, this is also the case in many other applications, for instance, in natural language processing, texts may differ in size, but there is no consensus as to what a good feature extractor for MILP instances looks like, contrary to other domains where variants of Recurrent Neural Networks or Transformers have become the de facto standard encoders.

We depart from previous approaches to Lagrangian prediction [234] restricted to instances of the same size and follow more generic approaches to MILP encoding such as [90, 189, 132] where each instance is converted into a bipartite graph, described in detail in Section 4.1.2, and further encoded by GNNs to compute meaningful feature vectors associated with dualized constraints. The Energy-Based modeling does not inherently require the use of a GNN. In principle, any suitable encoder architecture could be employed to map an instance representation to a latent vector for each dualized constraint. However, we choose to adopt a GNN due to the suitability of the bipartite graph representation, introduced in Section 4.1.2. This representation enables us to encode optimization instances in a way that is both scalable across varying instance sizes and well-suited for neural network processing. Furthermore, it provides a consistent feature representation for each dualized constraint, independent of the total number of constraints, by leveraging information from both variables and constraints. Each MILP is converted to a bipartite graph composed of one node for each variable and one node for each constraint. An edge exists between a variable node n_v and a constraint node n_c if and only if v appears in c. Each node (variable or constraint) is represented by an initial feature vector $\boldsymbol{e}_n$. We use features similar to those proposed in [90], see Section 4.1.2 for more details. Following [189], variables and constraints are encoded as the concatenation of variable features followed by constraint features, of which only one is non-zero, depending on the type of nodes.

To design our stack of GNNs, we take inspiration from structured prediction models for images and texts, where Transformers [247] are ubiquitous. However, since our input has a bipartite graph structure, we replace the multi-head self-attention layers with simple linear graph convolutions [138]. Alternatively, this layer can be seen as masked attention, where the mask is derived from the input graph adjacency matrix. We follow [189], which showed that residual connections [109], dropout [233], and layer normalization [12] are essential for the successful implementation of feature extractors for MILP bipartite graphs. More details on these components can be found in Chapter 2.

Before the actual GNNs, initial feature vectors $\{\boldsymbol{e}_n\}_n$ are passed through an MLP F to find feature combinations and extend node representations to high-dimensional spaces: $\boldsymbol{h}_n^0 = F(\boldsymbol{e}_n), \forall n$. Then, interactions between nodes are taken into account by passing vectors through blocks, represented in Figure 4.3, consisting of two sublayers, that compute for each node representation $\boldsymbol{h}_n^i$ an updated representation $\boldsymbol{h}_n^{i+1}$

- The first sublayer connects its input via a residual connection to a layer normalization LN followed by a linear graph convolution $CONV$, followed by a dropout regularization DO:

$$\boldsymbol{h}'_n = \boldsymbol{h}_n + DO(CONV(LN(\boldsymbol{h}_n)))$$

  The graph convolution passes messages between nodes. In our context, it passes information from variables to constraints, and conversely.

- The second sublayer takes the result of the first one as input. It connects it with a residual connection to a sequence made of a layer normalization LN, a Multi-Layer Perceptron (MLP) transformation, and a dropout regularization DO:

$$\boldsymbol{h}_n = \boldsymbol{h}'_n + DO(MLP(LN(\boldsymbol{h}'_n)))$$

  The MLP is responsible for finding non-linear interactions in the information collected in the previous sublayer.

This block structure, depicted in Figure 4.3, is repeated several times, typically five times in our experiments, to extend the locality domain. The learnable parameters of a block are the parameters of the convolution in the first sublayer and the parameters of the MLP in the second sublayer. Remark that we start each sublayer with normalization, as it has become the standard approach in Transformer recently [42]. We note in passing that this has also been experimented with by [90] in the context of MILP, although only once before the GNN input, whereas we normalize twice per block, at each block.

Finally, the GNN returns the vectors associated with dualized constraints $\{\boldsymbol{h}_c\}_c$. Each vector $\boldsymbol{h}_c$ is interpreted as the concatenation of two vectors $[\boldsymbol{z}_\mu; \boldsymbol{z}_\sigma]$ from which we compute $\boldsymbol{z}_c = \boldsymbol{z}_\mu + \exp(\boldsymbol{z}_\sigma) \cdot \boldsymbol{\epsilon}$ where elements of $\boldsymbol{\epsilon}$ are sampled from the normal distribution. This concludes the implementation of the probabilistic encoder $q_{\boldsymbol{\phi}}$.

Decoder Recall that, in our architecture, from each latent vector representation $\boldsymbol{z}_c$ of dualized constraint c, we want to compute the scalar deviation δ_c to the CR dual value λ_c so that the sum of the two improves the Lagrangian bound given by the CR dual solution. In other words, we want to compute $\boldsymbol{\delta}$ such as $\boldsymbol{\pi} = [\boldsymbol{\lambda}_D + \boldsymbol{\delta}]_+$ gives a *good* Lagrangian bound $LR(\boldsymbol{\pi})$. Its exact computation is combinatorial and problem-specific. [3] It is important to note that, since sampling is performed in the *latent space* rather than directly in the *output space*, this does not directly affect the structural properties of the Lagrangian multiplier vector. Therefore, to ensure that the predicted multipliers satisfy the required structural constraints, it may be necessary to apply a projection operator that enforces the prediction to be in the appropriate subspace. This is the approach we take for dualizing inequalities, since the corresponding multiplier vector must be non-negative.

The probabilistic nature of the encoder-decoder can be exploited further. During evaluation, when computing a Lagrangian relaxation, we sample constraint representations 5 times from the probabilistic encoder and return the best $LR(\boldsymbol{\pi})$ value from the decoder.

Link with Energy-Based Models in Structured Prediction The Lagrangian subproblem usually decomposes into independent subproblems due to the dualization of the linking constraints. Hence, we want to find an optimal variable assignment for each independent Lagrangian subproblem, usually with local combinatory constraints, for its objective reparameterized with $\boldsymbol{\pi}$. This approach is typical of structured prediction: we leverage neural networks to extract features to compute local energies (scalars), which

[3] $LR(\boldsymbol{\pi})$ is described in Section 4.2.1 and Section 4.2.2 for the two problems on which we evaluate our method.

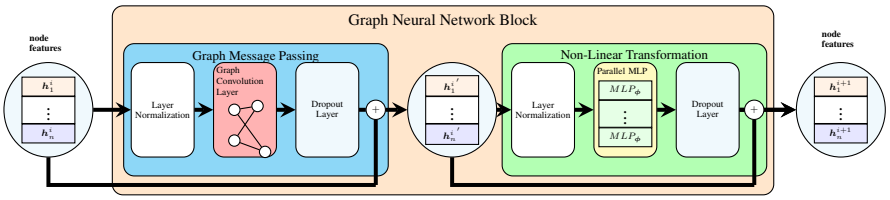

Figure 4.3: The Graph Neural Network block. The first part is graph message-passing: we apply layer normalization to node features, then convolution over the instance's bipartite graph representation, and finally, dropout. The second phase consists of normalization, a Multi-Layer Perceptron in parallel over all the nodes of the bipartite graph, then dropout. Both sublayers use residual connections between input and output. We apply this block several times to improve feature representations.

are used by a combinatorial algorithm to output a structure whose objective value can be interpreted as a global energy. For instance, this is similar to how graph-based syntactic parsing models in natural language processing compute parse scores (global energies) as sums of arc scores (local energies) computed by a Recurrent Neural Network followed by MLPs, where the choice of arcs is guided by well-formedness constraints enforced by a maximum spanning tree solver, see for instance [137]. Thus, the decoder is local to each dualized constraint, and we leverage subproblems to interconnect predictions:

1. We compute LMs (local energies) $\pi_c = [\lambda_c + f_{\boldsymbol{\theta}}(\boldsymbol{z}_c)]_+$ for all dualized constraints c, where $f_{\boldsymbol{\theta}}$ is implemented as a feed-forward network computing the deviation.

2. For parameter learning or if the subproblem solutions or the Lagrangian bound are the desired output, vector $\boldsymbol{\pi}$ is then passed to the Lagrangian subproblem which compute independently and in parallel their local solutions $\boldsymbol{x}$ and the corresponding values are summed to give the global energy $LR(\boldsymbol{\pi})$.

4.1.4 Initial Features

To extract useful features, we define a network based on graph convolutions presented in Figure 4.1 in the line of the work of [90] on MILP encoding. We detail the initial node features $\{\boldsymbol{e}_n\}_n$ of the MILP-encoding bipartite graph presented in Section 4.1.3.

Given an instance of the form:

$$(P) \qquad \min_{\boldsymbol{x}} \boldsymbol{w}^\top \boldsymbol{x} \tag{4.5a}$$

$$\boldsymbol{A}\boldsymbol{x} \begin{pmatrix} \geq \\ = \end{pmatrix} \boldsymbol{b} \tag{4.5b}$$

$$\boldsymbol{x} \in \mathbb{R}^m_+ \times \mathbb{N}^p \tag{4.5c}$$

we consider the following initial features for a variable x_j:

- its coefficient w_j in the objective function;

- its value in the primal solution of CR;
- its reduced cost $\bar{c}_j = w_j - \boldsymbol{\lambda}^T(\boldsymbol{A}_j|\boldsymbol{C}_j)$ in CR where $(\boldsymbol{A}_j|\boldsymbol{C}_j)$ is the vector constructed by appending $\boldsymbol{A}_j$, the j^{th} column of $\boldsymbol{A}$, and $\boldsymbol{C}_j$, the j^{th} column of $\boldsymbol{C}$. As previously $\boldsymbol{\lambda}$ is the dual solution of CR;
- a binary value indicating whether x_j is integral or continuous.

For constraint $\boldsymbol{a}^\top \boldsymbol{x} \begin{pmatrix} \geq \\ = \end{pmatrix} b$ of (4.5b), we consider:

- the right-hand side b of the constraint;
- the value of the associated dual solution in CR;
- one binary value indicating whether the constraint is an equality or an inequality;
- one binary value stating whether c is dualized in the relaxed Lagrangian subproblem.

We use for each node n of the bipartite graph a feature vector $\boldsymbol{e}_n \in \mathbb{R}^8$. The first four components are used to encode the initial features if n corresponds to a variable and are set to 0 otherwise. In contrast, the next four components are used only if n is associated with a constraint and are set to 0 otherwise.

4.2 Evaluation

We evaluate our approach on two standard Operations Research problems: Multi-Commodity Fixed-Charge Network Design and Generalized Assignment. In terms of Lagrangian relaxation, we consider the relaxation of equality constraints in the Multi-Commodity Fixed-Charge Network Design problem, which leads to an unconstrained (that is, unrestricted-sign) Lagrangian multiplier vector. For the Generalized Assignment problem, we relax inequality constraints, resulting in a nonnegative Lagrangian multiplier vector. We review briefly the two problems and the data generation process.

4.2.1 Multi-Commodity Fixed-Charge Network Design

Given a network with arc capacities and a set of commodities, Multi-Commodity Fixed-Charge Network Design (MC) involves activating a subset of arcs and routing each commodity from its origin to its destination, possibly fractionated across several paths, using only the activated arcs. The goal is to minimize the overall cost incurred by activating arcs and routing commodities. This problem has been used in many real-world applications for a long time, see for instance [172] for telecommunications. It is NP-hard, and its CR provides poor bounds when arc capacities are high. Hence, it is usually tackled with Lagrangian relaxation-based methods [3].

A typical instance of the MC problem is defined by a directed simple graph $D = (N, A)$, a set of commodities K, an arc-capacity vector c, and two cost vectors r and f. Each commodity $k \in K$ corresponds to a triple (o^k, d^k, q^k) where $o^k \in N$ and

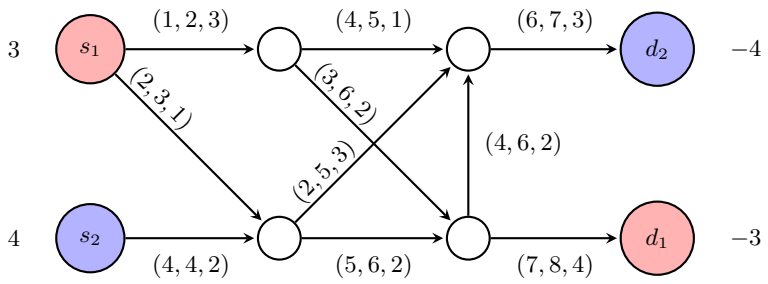

Figure 4.4: Example of MCND instance. It has two demands, the first in red and the second in blue. The labels next to the demand sources s_i and destinations d_i are the amount of flow volume that should go out (if positive) or into (if negative) the sources/destinations. The labels over the edges represent (in order) the capacity of the edge, the fixed cost, and the routing costs (assumed in this example to be the same for the two commodities).

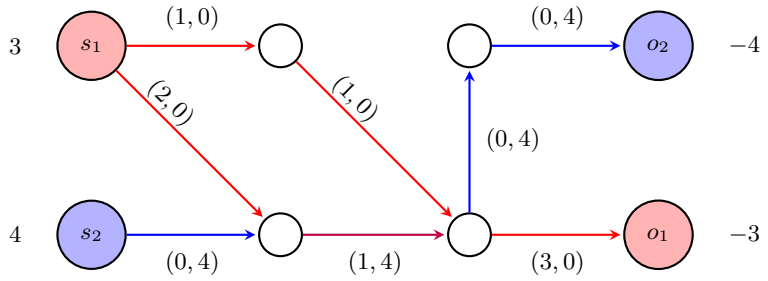

Figure 4.5: Solution of the example MCND instance represented in Figure 4.4. The red edges are the ones used by the first commodity, and the ones in blue are only by the second one. Both the commodity uses the purple edge. The labels over the edges represent the amount of flow sent by the commodities through the edge.

$d^k \in N$ are the nodes corresponding to the origin and the destination of commodity k, and $q^k \in \mathbb{N}^*$ is its volume. For each arc $(i,j) \in A$, $c_{ij} > 0$ corresponds to the maximum amount of flow that can be routed through (i,j) and $f_{ij} > 0$ corresponds to the fixed cost of using arc (i,j) to route commodities. For each arc $(i,j) \in A$ and each commodity $k \in K$, $r_{ij}^k > 0$ corresponds to the cost of routing one unit of commodity k through arc (i,j).

An MC solution consists of an arc subset $A' \subseteq A$ and, for each commodity $k \in K$, in a flow of value q^k from its origin o^k to its destination d^k with the following requirements: all commodities are routed only through arcs of A', and the total amount of flow routed through each arc $(i,j) \in A'$ does not exceed its capacity c_{ij}. The solution cost is the sum of the fixed costs over the arcs of A' plus the routing cost, the latter being the sum over all arcs $(i,j) \in A$ and all commodities $k \in K$ of the unitary routing cost r_{ij}^k multiplied by the amount of flow of k routed through (i,j).

This problem has been studied in several works considering different relaxations. In particular, we will consider the *Flow relaxation* [85] for the MC problem. For the MC problem, other relaxations exist that are more expensive to solve than the *Knapsack relaxation* but can provide even better bounds. The reader is referred to [3] for more details.

In many cases, the Bundle method is an efficient solver for such decomposition approaches [85].

MILP formulation

A standard model for the MC problem [91] introduces two sets of variables: the continuous flow variables x_{ij}^k representing the amount of commodity k that is routed through arc (i,j) and the binary design variables y_{ij} representing whether arc (i,j) is used to route commodities. Denoting respectively by $N_i^+ = \{j \in N \mid (i,j) \in A\}$ and $N_i^- = \{j \in N \mid (j,i) \in A\}$ the sets of forward and backward neighbors of a vertex $i \in N$, the MC problem can be modeled as follows:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{(i,j)\in A} \left(f_{ij} y_{ij} + \sum_{k \in K} r_{ij}^k x_{ij}^k \right) \tag{4.6a}$$

$$\sum_{j \in N_i^+} x_{ij}^k - \sum_{j \in N_i^-} x_{ji}^k = b_i^k \qquad \forall i \in N, \forall k \in K \tag{4.6b}$$

$$\sum_{k \in K} x_{ij}^k \leq c_{ij} y_{ij}, \qquad \forall (i,j) \in A \tag{4.6c}$$

$$x_{ij}^k = 0 \qquad \begin{array}{l} \forall k \in K, \forall (i,j) \in A \\ \text{s.t. } i = d^k \text{ or } j = o^k \end{array} \tag{4.6d}$$

$$0 \leq x_{ij}^k \leq q^k \qquad \forall (i,j) \in A, \forall k \in K \tag{4.6e}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A \tag{4.6f}$$

where

$$b_i^k = \begin{cases} q^k & \text{if } i = o^k, \\ -q^k & \text{if } i = d^k, \\ 0 & \text{otherwise.} \end{cases}$$

The objective function (4.6a) minimizes the sum of the routing and fixed costs. Equations (4.6b) are the flow conservation constraints that define the flow of each commodity through the graph, since $\boldsymbol{x} \geq 0$. Constraints (4.6c) are the capacity constraints ensuring that the total amount of flow routed through each arc does not exceed its capacity or is zero if the arc is not used to route commodities. Equations (4.6d) ensure that a commodity is not routed on an arc entering its origin or leaving its destination. Finally, inequalities (4.6e) are the bounds for the $\boldsymbol{x}$ variables, and inequalities (4.6f) are the integer constraints for the design variables.

Lagrangian Knapsack Relaxation

A standard way to obtain good bounds for the MC problem is to solve the Lagrangian relaxation obtained by dualizing the flow conservation constraints (4.6b) in Formulation (4.6). Let π_i^k be the Lagrangian multiplier associated with node $i \in N$ and commodity $k \in K$. Note that, since the dualized constraints are equations, $\boldsymbol{\pi}$ has no sign constraints. Dualizing the flow conservation constraints gives the following Lagrangian subproblem$LR(\boldsymbol{\pi})$:

$$\min_{(\boldsymbol{x},\boldsymbol{y}) \text{ satisfies (4.6c)}-\text{(4.6f)}} \sum_{(i,j)\in A} \left(f_{ij} y_{ij} + \sum_{k\in K} r_{ij}^k x_{ij}^k \right)$$
$$+ \sum_{k\in K} \sum_{i\in N} \pi_i^k \left(b_i^k - \sum_{j\in N_i^+} x_{ij}^k + \sum_{j\in N_i^-} x_{ji}^k \right)$$

Rearranging the terms in the objective function and observing that the Lagrangian subproblem is decomposed by arcs, we obtain a subproblem for each arc $(i,j) \in A$ of the form:

$$(LR_{ij}(\boldsymbol{\pi})) \quad \min_{\boldsymbol{x},\boldsymbol{y}} f_{ij} y_{ij} + \sum_{k\in K_{ij}} w_{ij}^k x_{ij}^k \tag{4.7a}$$
$$\sum_{k\in K_{ij}} x_{ij}^k \leq c_{ij} y_{ij} \tag{4.7b}$$
$$0 \leq x_{ij}^k \leq q^k \qquad \forall k \in K_{ij} \tag{4.7c}$$
$$y_{ij} \in \{0,1\} \tag{4.7d}$$

where $w_{ij}^k = r_{ij}^k - \pi_i^k + \pi_j^k$ and $K_{ij} = \{k \in K \mid j \neq o^k \text{ and } i \neq d^k\}$ is the set of commodities that may be routed through arc (i,j). Lagrangian duality implies that

$$LR(\boldsymbol{\pi}) = \sum_{(i,j)\in A} LR_{ij}(\boldsymbol{\pi}) + \sum_{i\in N} \sum_{k\in K} \pi_i^k b_i^k$$

is a lower bound for the MC problem, and the best one is obtained by solving the following Lagrangian Dual problem:

$$(LD) \quad \max_{\boldsymbol{\pi}\in\mathbb{R}^{N\times K}} LR(\boldsymbol{\pi})$$

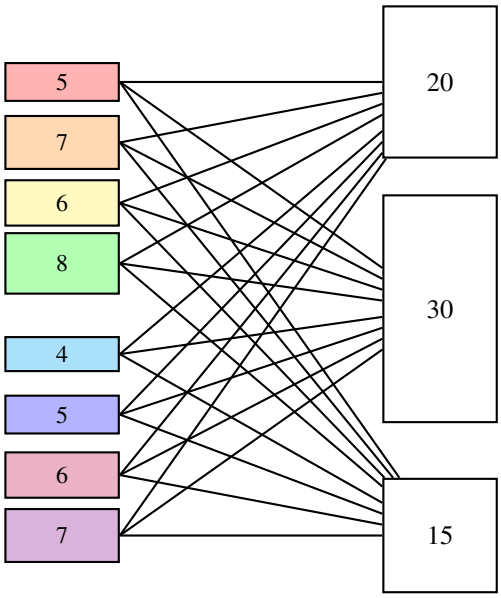

Figure 4.6: An example of a Generalized Assignment Instance with 3 bins (in white) and 8 objects (with different colors). The bins have different capacities (here 20, 30, and 15) while the objects have different sizes (written inside the boxes). Each edge should have a label, corresponding to the reward associated to assign an object to a bin. In this figure, we did not write them as it would be too confusing.

Solving the Lagrangian Subproblem For each $(i, j) \in A$, $LR_{ij}(\boldsymbol{\pi})$ is a MILP with only one binary variable. If $y_{ij} = 0$, then, by (4.7b) and (4.7c), $x_{ij}^k = 0$ for all $k \in K_{ij}$. If $y_{ij} = 1$, the problem becomes a continuous knapsack problem. An optimal solution is obtained by ordering the commodities of K_{ij} for decreasing values w_{ij}^k and setting for each variable x_{ij}^k the value $\max\{\min\{q^k, c_{ij} - \sum_{k \in K(k)} q^k\}, 0\}$ where $K(k)$ denotes the set of commodities that preceded k in the sorted list. This step can be done in $O(|K_{ij}|)$ by computing x_{ij}^k in the computed order. Hence, the complexity of the continuous Knapsack problem is $O(|K_{ij}| \log(|K_{ij}|))$. The solution of $LR_{ij}(\boldsymbol{\pi})$ is the minimum between the cost of the continuous Knapsack problem and $\mathbf{0}$.

Note that the continuous knapsack can be solved in linear time [205].

4.2.2 Generalized Assignment (GA)

Generalized Assignment (GA) [216] consists, given a set of items and a set of capacitated bins, of assigning items to bins without exceeding their capacity to maximize the assignment's profit. GA is a well-known problem in Operations Research and has numerous applications [194] such as job-scheduling in Computer Science [14], distributed caching [81], or even parking allocation [184]. We refer to [41] for a survey.

Lagrangian relaxationhas already been studied for this problem [128, 191].

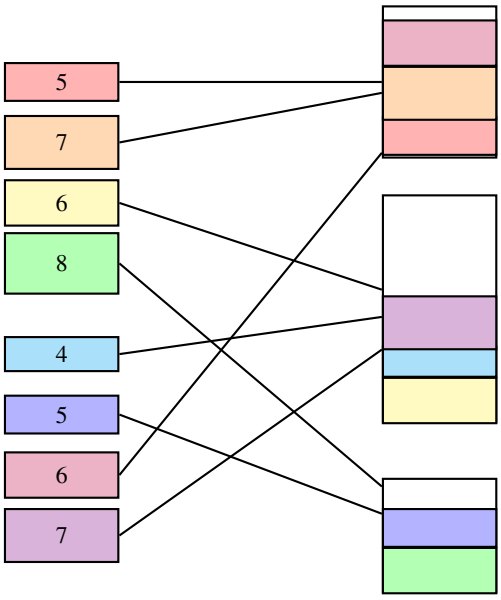

Figure 4.7: An example of a solution of the GA instance in Figure 4.6. Each object is assigned to exactly one bin without exceeding the capacity of the bin. The optimal solution will be the one that minimizes the costs (not represented in Figure 4.6).

A GA instance is defined by a set I of items and a set J of bins. Each bin j is associated with a certain capacity c_j. For each item $i \in I$ and each bin $j \in J$, p_{ij} is the profit of assigning item i to bin j, and w_{ij} is the weight of item i inside bin j.

MILP formulation

Considering a binary variable x_{ij} for each item and each bin that is equal to one if and only if item i is assigned to bin j, the GA problem can be formulated as:

$$\max_{\boldsymbol{x}} \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \tag{4.8a}$$

$$\sum_{j \in J} x_{ij} \leq 1 \qquad \forall i \in I \tag{4.8b}$$

$$\sum_{i \in I} w_{ij} x_{ij} \leq c_j \qquad \forall j \in J \tag{4.8c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I,\ \forall j \in J. \tag{4.8d}$$

The objective function (4.8a) maximizes the total profit. Inequalities (4.8b) assert that each item is contained in no more than one bin. Inequalities (4.8c) ensure that the sum of the weights of the items assigned to a bin does not exceed its capacity. Finally, constraints (4.8d) assure the integrality of the variables.

Lagrangian Relaxation

A Lagrangian relaxation of the GA problem is obtained by dualizing (4.8b). For $i \in I$, let $\pi_i \geq 0$ be the Lagrangian multiplier of inequality (4.8b) associated with item i. For each bin j, the subproblem becomes:

$$\begin{aligned}(LR_j(\boldsymbol{\pi})) \quad & \max_{\boldsymbol{x}} \sum_{i \in I} \sum_{j \in J} (p_{ij} - \pi_i) x_{ij} \\ & \sum_{i \in I} w_{ij} x_{ij} \leq c_j \\ & x_{ij} \in \{0,1\} \qquad\qquad \forall i \in I\end{aligned}$$

It corresponds to an integer Knapsack with $|I|$ binary variables. For $\boldsymbol{\pi} \geq \mathbf{0}$, the Lagrangian bound $LR(\boldsymbol{\pi})$ is:

$$LR(\boldsymbol{\pi}) = \sum_{j \in J} LR_j(\boldsymbol{\pi}) + \sum_{i \in I} \pi_i.$$

The Lagrangian Dual can then be written as:

$$\min_{\boldsymbol{\pi} \in \mathbb{R}^{|I|}_{\geq 0}} LR(\boldsymbol{\pi})$$

Solving the Lagrangian Subproblem In this case, the Lagrangian subproblem reduces to a binary Knapsack problem [61]. To solve it, we implement a well-known dynamic programming algorithm [175]. This algorithm is pseudo-polynomial, and more precisely, for a subproblem associated with the bin $j \in J$, the complexity is $O(|I| \cdot c_j)$. Hence, the cost to solve all the subproblems is $O(|I| \cdot \sum_{j \in J} c_j)$.

4.2.3 Dataset collection details

In this sub-section, we provide further details on the dataset construction.

Multi-Commodity Fixed-Charge Network Design

While the Canad dataset is the standard and well-established dataset of instances for evaluating MC solvers [57], it is too small to be used as a training set for machine learning, where large collections of instances sharing common features are required. Thus, we generate new instances from a subset of instances of the Canad dataset [57], which we divide into four datasets of increasing difficulty. The first two datasets, MC-SML-40 and MC-SML-VAR, contain instances that all share the same network (20 nodes and 230 edges) and the same arc capacities and fixed costs, but with different values for origins, destinations, volumes, and routing costs. Instances of the former all involve the same number of commodities (40), while for the latter, the number of commodities varies from 40 to 200. Dataset MC-BIG-40 is generated similarly to MC-SML-40 but upon a bigger graph containing 30 nodes and 520 arcs. Finally, MC-BIG-VAR contains

examples generated using either the network of MC-SML-40 or the one of MC-BIG-40, with the number of commodities varying between 40 and 200.

We generate four datasets of 2000 instances each (1600 for training, 200 for validation, and 200 for test) based on Canad instances [57]. These Canad instances have been chosen such that the Lagrangian Dual bound can be solved in nearly one second for the easiest instances and in approximately one hour for the hardest ones.

The first two datasets MC-SML-40 and MC-SML-VAR consider the same graph with 20 nodes and 230 edges, and the same capacity and fixed cost vectors. The first dataset has only instances with 40 commodities, whereas the second one has instances with 40, 80, 120, 160, or 200 commodities.

Origins and destinations are randomly chosen using a uniform distribution. Volumes and routing costs are randomly sampled using a Gaussian distribution. Sampling uses four different means μ and variances σ^2, which are determined from the four canad instances p33, p34, p35 and p36 (having the same graph and fixed costs as the datasets) in order to generate four different types of instances: whether the fixed costs are high with respect to routing costs, and whether capacities are high with respect to commodity volumes.

The third dataset MC-BIG-40 is generated similarly to the first one, except that it is based on a graph with 30 nodes and 520 edges. The means and variances used to sample the fixed costs and the volumes are determined from the four Canad instances p49, p50, p51, and p52. The number of commodities is equal to 40 in each instance.

Finally, the last dataset MC-BIG-VAR contains instances with either the graph, capacities, and fixed costs of the first two datasets or the ones of the third dataset. Sampling uses either the Canad instances p33, p34, p35, and p36 or the Canad instances p49, p50, p51, and p52 for determining the mean and variance, depending on the size of the graph. The number of commodities varies from 40 to 200 if the graph is one of the first two datasets and from 40 to 120 otherwise.

Generalized Assignment

We create two datasets of GA instances containing 2000 instances each (1600 for training, 200 for validation, and 200 for test). The first one contains instances with 10 bins and 100 items, whereas the second one contains instances with 20 bins and 400 items. For each dataset, all instances are generated by randomly sampling capacities, weights, and profits using a Gaussian distribution of mean μ and variance σ^2, and the values are clipped to an interval $[a, b]$. The values μ, σ^2, a and b are determined from the instance e10100 for the first dataset, and from the instance e20400 for the second one [4]. More specifically, for each type of data (capacities, weights and profits), μ and σ^2 are given by the average and variance of the values of the instance, and a and b are fixed to 0.8 times the minimum value and 1.2 times the maximum value, respectively.

[4] Instances e10100 and e20400 are GA instances generated by [268] and available at `http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/gap/`.

4.3 Numerical Results

We want to evaluate how our Lagrangian bound prediction compares to an iterative model based on subgradient and how useful the former is as an initial point to warm-start the latter. For that purpose, we choose a state-of-the-art proximal Bundle solver provided by SMS++ [88], which allows writing an MILP in a block structure fashion and using decomposition techniques to solve subproblems efficiently. We also compare our approach with the CR computed using the CPLEX[5] optimizer.

All MILP instances for which we want to evaluate our model are first solved by SMS++. For an instance ι we denote $\boldsymbol{\pi}^*_\iota$ the LMs returned by SMS++.

4.3.1 Hyperparameters and Implementation Details

In this subsection, we provide technical details about the framework implementation. More specifically, we define the model architecture in terms of the number of layers, layer sizes, and activation functions. We define hand-crafted features for the baseline methods used in our comparison, such as MLP and k-NN, which do not rely on GNN-based automatic feature extraction. We detail the optimizer hyperparameters used during training, as well as the GPU and CPU specifications of the machines employed in the numerical experiments. Finally, we define the metrics and the validation techniques used to analyze the quality of the provided solutions.

Model Architecture For all datasets, the MLP F from initial features to high dimensions is implemented as a linear transformation (8 to 250) followed by a non-linear activation. Then, we consider a linear transformation of the size of the internal representation of nodes for the GNN. The choice of this hidden space dimension is based on preliminary results indicating that higher dimensions (as 500) do not yield improved performance and require longer computation times. In contrast, smaller dimensions (as 125) result in weaker performance.

For MC, we use 5 blocks, while for GA, we use only 3. The observation that fewer layers suffice for GA can be explained by examining the bipartite graph representation of the instance, which is denser in GA than in MC. For example, in MC, a variable x^k_{ij} appears in three constraints involving several variables, while in GA, each variable x_{ij} appears in $|I| + |J|$ constraints. Hence, the propagation needs fewer convolutions for the information to be propagated. Additional information regarding the number of blocks is provided in Tables 4.5 and 4.6, at page 81.

The hidden layer of the MLP in the second sub-layer of each block has a size of 1000. The decoder is an MLP with one hidden layer of 250 nodes, which is the same size as the hidden representation.

All non-linear activation functions are implemented as ReLU. Only the one for the output of the GA is a softplus. The dropout rate is set to 0.25.

Features used for MLP and k-NN Since MLP and k-NN do not use a mechanism such as convolution to propagate the information between the representations of the

[5] https://www.ibm.com/products/ilog-cplex-optimization-studio

dualized constraints, we consider for initial features of each dualized constraint all the information provided to our model, as well as a weighted linear combination of variable feature vectors. The weights are the variable coefficients in that constraint, and each feature vector contains the initial features provided to our model for the variable and the following additional information:

- the mean values and deviations of the coefficients of that variable on the dualized constraints and the non-dualized ones,
- its lower and upper bounds.

Optimiser Specifications We use RAdam as optimizer, with learning rate 0.0001 for MC and 0.00001 for GA, a Clip Norm (to 5) and exponential decay 0.9, step size 100000 and minimum learning rate 10^{-10}.

GPU specifics For the training on the datasets MC-SML-40, MC-BIG-40, GA-10-100, and GA-20-400, we use Nvidia Quadro RTX 5000 GPUs with 16 GB of RAM. To train the datasets MC-SML-VAR and MC-BIG-VAR we use Nvidia A40 GPUs accelerators with 48 GB of RAM. To test performance, we use Nvidia A40 GPU accelerators with 48Gb of RAM for all models and datasets in validation and test sets.

CPU specifics The warm start of the proximal Bundle in SMS++ needs only CPU; the experiments are done on an Intel Core i7-8565U CPU @ 1.80GHz × 8.

Metrics We use the percentage gap as a metric to evaluate the quality of the bounds computed by the different systems, averaged over a dataset of instances $\mathcal{I}$. For a system returning a bound B_ι for an instance ι the percentage GAP is:

$$100 \times \frac{1}{|\mathcal{I}|} \sum_{\iota \in \mathcal{I}} \frac{LR(\boldsymbol{\pi}^*_\iota) - B_\iota}{LR(\boldsymbol{\pi}^*_\iota)}$$

GAP measures the quality of the bound B_ι, and is zero when B_ι equals the optimal Lagrangian bound.

Data for Evaluation We divide each dataset of 2000 instances into train (80%), validation (10%), and test (10%). Parameters are learned on the training set, model selection is performed on the validation set, and test proxies for unseen data. Results are averaged over three different random initializations.

4.3.2 Bound Accuracy

Table 4.1 reports the performance of different Lagrangian multipliers prediction methods on our 6 datasets. We compare the bound returned by CR, and the bound of the Lagrangian subproblem obtained with LMs computed by different methods:

- LR(0) is the LR value computed with LMs set to zero.

- LR(CR) is the LR value computed with LMs set to the CR dual solution.
- LR(k-NN) is the LR value computed with LMs set to the average value of LMs from the train set returned by a k-NN regressor.
- LR(MLP) is the LR value computed with LMs returned by an MLP[6] instead of the GNN-based encoder-decoder.
- Ours, that is, the LR value computed with LMs set the output of our encoder-decoder. As written in the previous section, we sample 5 LM assignments per instance and return the best LR value.

Let us observe that all baseline methods, except for $LR(0)$, require solving the CR to generate their predictions. This implies that their runtime is at least as large as that of solving the CR itself, and therefore always greater than or equal to both CR and $LR(CR)$, as they also require solving the Lagrangian subproblem. Moreover, baselines that exploit information from the CR are only useful if they lead to a smaller GAP. When such methods yield worse GAP values, as in the case of $LR(k\text{-}NN)$, this indicates that the prediction fails to effectively leverage the information from the CR, rendering the approach ineffective in practice.

All the approaches considered here are one-shot, meaning they provide a bound in a non-iterative way, without any time limit. Some of these approaches also produce a vector of Lagrangian multipliers that can be used to warm-start an iterative solver for the Lagrangian Dual, thereby enabling the computation of more accurate solutions. A comparison of different initialization strategies for such a solver, specifically the Bundle method, is presented in Table 4.2.

For all datasets, our method outperforms other approaches. Our model can reach 2% difference with BM on MC-SML-40, the easiest corpus with a small fixed network and a fixed number of commodities. This means that if we can accept an average 2% bound error, one pass through our network can save numerous iterations. The margin with other methods is quite large for MC datasets where the CR bound is far from the optimum, with a gap reduction ranging from 77% (MC-BIG-VAR) to 84% (MC-SML-40) depending on the dataset. For GA, where the CR is closer to the optimum, our model still finds better solutions. Even though the gap absolute difference may seem small, the gap reduction from the second-best model LR(CR) ranges from 30% (GA-10-100) to 44% (GA-20-400), a significant error reduction.

Compared to more straightforward ML approaches, we see *(i)* that retrieving LM values from k-NN is not a viable solution, even when the validation instances are close to the training instances (MC-SML-40), k-NN cannot find meaningful neighbors, and *(ii)* the graph feature extractor (GNN) is paramount: the LR(MLP) architecture seems unable to deviate LMs consistently from the CR solutions. It can even perform worse than CR or LR(CR) (GA-20-400).

Regarding speed, LR(0) is the fastest since it simply amounts to solving the relaxed Lagrangian subproblem with the original costs. Then CR and LR(CR) are second, the difference being that for the latter, after solving CR, the dual solution $\boldsymbol{\lambda}_D$ associated with

[6] Additional initial features are used, as shown in Section 4.3.1

Table 4.1: Bound accuracies of different *one-shot* methods on test sets, averaged over instances. All methods, except for LR(0), require solving the CR to make predictions. Therefore, in these cases, their runtimes must be greater than that of the CR.

Dataset	Methods	GAP %	time (ms)
MC-SML-40	CR	12.99	90.63
	LR(0)	100.00	**0.35**
	LR(CR)	12.97	90.98
	LR(k-NN)	38.80	219.42
	LR(MLP)	10.70	142.48
	ours	**2.09**	120.96
MC-SML-VAR	CR	22.29	283.63
	LR(0)	100.00	**1.32**
	LR(CR)	22.29	285.03
	LR(k-NN)	44.12	371.51
	LR(MLP)	16.71	369.61
	ours	**4.42**	374.20
MC-BIG-40	CR	15.94	220.91
	LR(0)	100.00	**0.75**
	LR(CR)	15.85	229.57
	LR(k-NN)	54.57	334.99
	LR(MLP)	13.67	556.89
	ours	**4.20**	283.40
MC-BIG-VAR	CR	20.66	287.20
	LR(0)	100.00	**1.37**
	LR(CR)	20.63	288.55
	LR(k-NN)	49.74	886.91
	LR(MLP)	16.14	515.60
	ours	**4.77**	374.78
GA-10-100	CR	1.91	9.59
	LR(0)	3.13	**0.44**
	LR(CR)	0.79	10.15
	LR(k-NN)	1.07	11.70
	LR(MLP)	0.78	51.71
	ours	**0.55**	16.19
GA-20-400	CR	0.44	71.40
	LR(0)	2.70	**7.51**
	LR(CR)	0.27	78.80
	LR(k-NN)	0.43	89.68
	LR(MLP)	0.28	114.41
	ours	**0.15**	124.96

the dualized constraint is used to compute the $LR(\boldsymbol{\lambda}_D)$. Slowness for LR(MLP) and LR(k-NN) is mainly caused by additional feature extraction. discussed in Section 4.3.1.

4.3.3 Warm-starting the Bundle Method

We want to test whether the Lagrangian multipliers predicted by our model can be used as an informed starting point for an iterative solver for the Lagrangian Dual (LD), namely the Bundle method as implemented by SMS++. The Bundle method is stabilized with a quadratic penalty, assuring a smooth objective at the expense of longer computation times. We hope our model can produce good starting points and thus avoid many early iterations.

In Table 4.2 we compare different initial LM vectors on the validation set of MC-

Table 4.2: Impact of initialization for a Bundle solver on MC-BIG-VAR. We consider initializations from the null vector (zero), the CR duals (CR), and our model (Ours).

ϵ	zero		CR		Ours	
	time (s)	# iter.	time (s)	# iter.	time (s)	# iter.
1e-1	34.34 (±81.22)	90.12 (±52.41)	31.67 (±75.12)	83.00 (±50.73)	**16.09** (± 42.79)	**60.39** (± 41.37)
1e-2	68.80 (±188.21)	141.43 (±112.72)	62.09 (±171.71)	133.26 (±109.60)	**36.10** (± 106.22)	**105.93**(± 97.04)
1e-3	100.71 (±288.16)	188.14 (±167.33)	89.26 (±251.15)	179.40 (±170.15)	**57.23** (± 177.24)	**143.36** (± 142.58)
1e-4	105.03 (±298.53)	207.90 (±198.92)	101.14 (±283.52)	200.42 (±197.60)	**63.25** (± 190.47)	**159.42** (± 162.32)

BIG-VAR for the Bundle method. We run our Bundle solver until the difference between $LR(\boldsymbol{\pi}^*)$ and the current bound is smaller than the threshold ϵ. We average resolution times and numbers of iterations over instances and compute the standard deviation. We compare three initialization methods: zero initializations, using the CR dual solutions, and our model's predictions.

We can see that CR is not competitive with the null initialization since the supplementary computation absorbs the small gain in the number of iterations. However, despite the additional prediction time, our model's predictions significantly improve over the other two initialization methods. Resolution time is roughly halved for the coarsest threshold and above one-third faster for the finest one. This is expected, as gradient-based methods naturally slow down as they approach convergence.

We also note that this initialization could be further improved by incorporating more bundle information. For example, using an initial bundle of gradients and predicting a good initial value for the regularization parameters. More generally, one could consider learning an initialization specifically optimized to perform well over a fixed number of Bundle method iterations. However, achieving this would require differentiating through the Bundle method execution, which is a nontrivial challenge. These considerations motivate adapting the Bundle method to support differentiation through its steps, thereby leading to the research direction presented in Chapter 5.

4.3.4 Ablation Study

In this subsection, we present and discuss additional numerical experiments designed to: better understand how the main components of our architecture affect performance, evaluate generalization to larger instances within the same problem family, and assess the impact of model size on performance.

Model Modifications In Table 4.3 we compare three variants of our original model, denoted `ours`, on MC-SML-40 and MC-BIG-VAR. Results are averaged over 3 runs.

In the first variant, `-max`, we sample a single multiplier per constraint, instead of drawing multiple candidates per dualized constraint and selecting the best. We can see that this has a minor impact on the quality of the returned solution. In `-sum`, the dual solution values are passed as constraint node features but are not added to the output of the decoder to produce LMs, *i.e.* the network must transport these values from its input layer to its output. This has a significant negative impact on GAP scores. In the third

Table 4.3: Ablation studies comparing the prediction of our model (`-ours`) with predicting LMs, or using the same model to perform 5 predictions and take the best (`-max`), or on rather than deviation from the CR (`-sum`), not using the CR features at all (`-cr`), or replacing the probabilistic encoder by a deterministic one (`-sample`).

	GAP %	
model	MC-SML-40	MC-BIG-VAR
ours	2.09	4.77
-max	2.10	4.79
-sum	2.63	6.77
-cr	20.26	23.78
-sample	2.18	5.86

variant, `-cr`, the CR solution is not given as input features to the network (nor added to the network's output). This is challenging because the network lacks a good starting point, effectively equivalent to initializing the Lagrange multipliers to zero. The last variant, `-sample`, uses the CR as `ours` but does not sample representations $\boldsymbol{z}_c$ in the latent domain. We interpret the vector $\boldsymbol{h}_c$ associated with dualized constraint c after the GNN stack directly as vector $\boldsymbol{z}_c$, making the encoder deterministic.

We can see that the performance of `-sum` is just below `ours`, while `-cr` cannot return competitive bounds. These results suggest that incorporating the CR solution as input features is critical to the effectiveness of our architecture. In contrast, the direct computation of the deviation instead of the full LM is not an important trait. Still, we note that the performance of `-cr` should be compared with $LR(0)$ in Table 4.1 rather than $LR(CR)$. In that case, the GAP is reduced by approximately 80%, showing that our model does not simply repeat the CR solutions. This means that our model could be used without exploiting the CR solution as input information, opening our methods to a wider range of problems and paving the way for faster models.

Finally, `-sample` is a system trained without sampling at training time, *i.e.* the encoder-decoder is deterministic. We see that sampling results in a slight performance increase in both small and large instances.

Generalization Properties We test the model trained on MC-BIG-VAR on a dataset composed of 1000 bigger instances. They are created using the biggest graph used to generate the MC-BIG-VAR dataset, but contain 160 or 200 commodities, whereas the instances of MC-BIG-VAR with the same graphs only contain up to 120 commodities.

In Table 4.4, we can see that our model still performs well in these instances, dividing the gap provided by LR(CR) by four.

Number of Layers In Tables 4.5 and 4.6, we present the gaps of three different architectures with varying numbers of layers. The columns represent three different architectures: "Ours," the architecture we introduce in this work; "Nair," the architecture proposed by Nair et al. in [189]; and "Gasse," the one presented in [90]. The rows indicate an incremental number of layers from one to ten.

Table 4.4: Generalization results over larger instances.

	GAP %		time (s)	
# commodities	Ours	LR(CR)	Ours	LR(CR)
160	6.51	27.85	1.533	0.8915
200	7.62	30.18	1.4328	1.0889

Table 4.5: Test set 1 sample - MC-SML-40 - GAP

# Layers	Ours	Nair	Gasse
1	7.56	7.63	9.59
2	5.27	5.23	10.11
3	3.18	3.30	9.47
4	2.62	3.06	2.72
5	2.29	2.47	2.59
6	1.90	2.23	2.76
7	1.80	1.91	2.80
8	1.69	1.76	2.68
9	1.64	1.70	2.84
10	1.56	1.56	3.16

From Table 4.6, we observe that using more than four layers for GA seems to be counterproductive. This can be explained by examining the bipartite graph representation of the instance. In the Generalized Assignment problem, the shortest path between two different nodes associated with the relaxed constraints always consists of four edges. For the Multi-commodity problem, there is no similar bound, as the shortest path between two relaxed nodes depends on the specific structure of the instance. From Table 4.5, we see that adding more than six layers leads to diminishing improvements, though we can still enhance solution quality by increasing the number of layers.

The computation times for all models and all numbers of layers are similar, ranging from 0.012 seconds to 0.015 seconds. There is a slight increase in time with more layers. Among the architectures, Gasse's model is only 0.001 seconds faster than ours, which requires approximately the same time as Nair's model.

Gasse's architecture is also more unstable, which could result in significantly higher gaps with some layers compared to fewer layers. This instability may be due to the absence of Layer Normalization, leading to very high gradient values.

Notice that the results in Table 4.5 and Table 4.6 show slight differences compared to the ones on the main part for 5 layers, as they correspond to other training runs.

4.4 Conclusions

In this chapter, we present an approach for predicting a vector of Lagrangian multipliers.

A key advantage of our method, compared to existing approaches in the literature,

Table 4.6: Test set 1 sample - GA-10-100 - GAP (s)

# Layers	Ours	Nair	Gasse
1	0.553	0.557	0.70
2	0.543	0.546	0.699
3	0.533	0.524	0.695
4	0.509	0.524	0.690
5	0.512	0.517	0.682
6	0.513	0.518	0.720
7	0.510	0.511	0.785
8	0.512	0.517	0.752
9	0.511	0.515	0.785
10	0.512	0.516	0.722

is its ability to handle instances of varying sizes, thus accommodating Lagrangian multipliers vectors of different dimensions. This flexibility is enabled by employing the *bipartite graph representation* [90] of a problem instance.

The model begins with a probabilistic encoder that processes the bipartite graph representation and produces one latent representation per dualized constraint. Instead of outputting a deterministic representation, the encoder predicts the mean and variance of a Gaussian distribution, from which multiple latent representations can be sampled. These sampled representations are then passed through a decoder, which operates in parallel across the dualized constraints and predicts deviations from the corresponding components of the dual solution in the Lagrangian relaxation.

When dualizing inequality constraints, we apply a final non-negative activation function to ensure that the predicted Lagrangian multipliers remain feasible.

It is worth noting that using the CR solution constrains the applicability of our approach to instances where the CR is relatively easy to compute, but at the cost of producing weaker bounds compared to those obtained from Lagrangian relaxation. These limitations motivated us to investigate more generalizable, generative alternatives. Preliminary steps in this direction are discussed in the future work outlined in Chapter 6.

We demonstrate that our approach can be trained to maximize the Lagrangian bound, eliminating the need for optimal multipliers during training. Furthermore, we show how this framework can be interpreted as an Energy-Based Model (EBM), highlighting a novel application of EBMs to the Lagrangian relaxation setting.

Numerical results show that our method can significantly improve the quality of the primal solution obtained from dualized constraints in the CR and outperforms standard machine learning baselines. Additionally, we evaluate our predictions as initializations for the Bundle method, an exact solver of the Lagrangian Dual. Our approach consistently reduces both the number of iterations and the time required to solve it within a specified accuracy.

Chapter 5

Machine Learning for Non-Smooth Optimization

This chapter presents two machine learning-based approaches to tackle non-smooth optimization. The first approach is developed to work explicitly inside the Bundle method, while the second approach takes inspiration from it to provide a differentiable framework that enables end-to-end differentiation using Automatic Differentiation techniques. We restrict ourselves to dualizing only equality constraints, which leads to a Lagrangian Dual that is an unconstrained concave problem. It is possible to extend the presented approach to handle dualization of inequalities. However, this case is not explicitly considered in the numerical results, and we prefer to discuss it in Chapter 6.

As discussed in Chapter 1, the Bundle method is an iterative algorithm needing, at each iteration, the selection of a scalar that simultaneously serves as the step size and the weight of the regularization term in the Master problem. This *regularization* parameter, denoted by η in Chapter 1, controls both the step magnitude and the update direction, and significantly influences the formulation and solution of the Dual Master problem. Traditionally, selecting this parameter relies on heuristic approaches, referred to as η*-strategies*, often requiring an extensive grid search to achieve high performance. Some of those are presented in Chapter 1.

The first approach presented in this chapter constructs an η*-strategy* using a machine-learning model. Although it can be seen as hyperparameter tuning of the η regularization in the Bundle method, the approach differs from *classic* approaches of Algorithm Configuration. In the latter, algorithmic parameters are mainly chosen using techniques that do not require differentiation during algorithm execution, as they generally need to run the algorithm for all the datasets with several parameter configurations. Our approach differs from Algorithm Configuration approaches as it is closer to the Amortization and Unrolling methods. We propose an *objective-based* learning approach that requires differentiation through the algorithm execution.

An RL approach could tackle this problem, but we take advantage of a heuristic differentiation of our algorithm to propose an Unrolling approach. A key challenge while keeping the structure of the Bundle method is differentiating the search direction

with respect to the regularization parameter. To address this problem, we obtain an analytical formulation of the solution to the Dual Master problem (1.20) as a function of the same iteration regularization parameter, allowing us to approximate the derivative of this step. This methodology aligns with techniques presented in Chapter 3 concerning differentiation through the Karush-Kuhn-Tucker (KKT) conditions.

However, this approach does not account for dependencies on previous search directions, as it only expresses the direction in the current iteration as a function of the regularization parameter at the same iteration. Additionally, standard Bundle method implementations involve non-differentiable computations, such as piecewise-constant operations with uninformative derivatives.

To overcome these limitations, the second approach replaces these non-differentiable components with smoother approximations based on a neural network model. Specifically, we propose a neural network that approximates the search direction using information from the current bundle and the step size. This neural network replaces the $\eta-$strategy heuristic and the Dual Master problem (1.20) that computes the search direction from the current bundle and the $\eta-$strategy. This neural network integrates an approach inspired by the Bundle method, as outlined in Algorithm 1, incorporating *soft-updates* for the stabilization point. The result is a differentiable approach that enables end-to-end differentiation using Automatic Differentiation techniques. This approach is presented to handle unconstrained concave optimization problems, which corresponds, in the Lagrangian relaxation, to dualizing equality constraints. It can be seen as a learned optimizer for unconstrained concave maximization, thus belonging to the Learning-to-Optimize domain.

Both approaches are based on unrolling techniques, briefly presented in Section 3.2.4 of Chapter 3, consisting of reformulating the algorithm execution to allow for a computation graph of the process that can be used during back-propagation. Furthermore, both share the same loss function, which is presented in Section 5.1.

The following sections focus separately on the two approaches, presenting the provided prediction and showing how we can differentiate through the execution of those methods. More precisely, Section 5.1.1 is dedicated to learning only the regularization parameter η, while Section 5.1.3 is dedicated to the approach that is based on machine learning.

5.1 Overall Architecture

Since the decisions taken by the two approaches are made dynamically during algorithm execution, Reinforcement Learning [236] is inherently relevant. To enable the application of standard Reinforcement Learning techniques, the problem has to be framed as a Markov Decision Process [210]. However, this is usually done when differentiating the execution is not possible. We primarily focus on algorithm differentiability, allowing the use of a *conventional* learning approach by directly differentiating the objective function. Recent advancements in machine learning have explored differentiating the output of iterative algorithms with respect to iteratively selected parameters, as discussed in the section on Amortization in Chapter 3. Building upon this research direction, we explicitly demonstrate how it is possible to differentiate through the execution of the

Bundle method.

Amortized optimization radically differs from Reinforcement Learning. The former is trained using supervised learning techniques, such as minimizing the expected error or directly maximizing/minimizing the objective function over many instances. In contrast, Reinforcement Learning approaches are trained to maximize the expected reward through exploration, relying on trials and rewards that are not directly differentiable.

The approaches presented in this section address problems of the form:

$$\max_{\boldsymbol{\pi} \in \mathbb{R}^m} \phi(\boldsymbol{\pi})$$

where $\phi : \mathbb{R}^m \to \mathbb{R}$ is a concave function and we have no constraints in the feasible region. We recall that the Lagrangian Dual is an application case. As we are considering unbounded $\boldsymbol{\pi}$, this corresponds to dualizing equality constraints. Observe that the first approach can also be applied as it is to $\boldsymbol{\pi} \geq 0$, that is, dualizing inequalities. Instead, the second approach needs some modifications to be applied in the setting of non-negative solutions. Anyway, these modifications are not explicitly tested in the numerical experiment, hence we restrict ourselves to $\boldsymbol{\pi} \in \mathbb{R}^m$.

In both proposed approaches, we assume that a fixed maximum number of iterations, T, is performed, and the learning problem can be formulated as the maximization of the final objective function. Learning-to-Optimize approaches aim to maximize the function:

$$\mathcal{L}_{\boldsymbol{\gamma}}(\boldsymbol{W}) = \sum_{t=1}^{T} \gamma_t \phi(\boldsymbol{\pi}_t(\boldsymbol{W})) \tag{5.1}$$

where $\boldsymbol{\pi}_t(\boldsymbol{W})$ denotes the trial point at iteration t. It is computed by a neural network parametrized by $\boldsymbol{W}$. This neural network architecture depends on the approach, and is described in Section 5.1.1 and 5.1.3. Furthermore, we consider $\gamma_t = \gamma^{T-t}$, where $0 \leq \gamma \leq 1$ is a hyperparameter.

The function $\mathcal{L}\boldsymbol{\gamma}(\boldsymbol{W})$ is a linear combination of the function values ϕ evaluated at different points along the trajectory. Consequently, its subgradient with respect to $\boldsymbol{W}$ can be written as

$$\partial_{\boldsymbol{W}} \mathcal{L}\boldsymbol{\gamma}(\boldsymbol{W}) = \left\{ \sum_{t=1}^{T} \gamma_t \boldsymbol{v}_t \ : \ \boldsymbol{v}_t \in \partial_{\boldsymbol{W}} \phi(\boldsymbol{\pi}_t(\boldsymbol{W})) \text{ for } t = 1, \cdots, T \right\}.$$

Thus we only need the subgradients $\partial_{\boldsymbol{W}} \phi(\boldsymbol{\pi}_t(\boldsymbol{W}))$ for each $t \leq T$.

Although the function $\phi(\boldsymbol{\pi}_t(\boldsymbol{W}))$ could be *non-concave*, and so the *standard subgradient* can be undefined. This poses no practical issues when using it as a loss function for learning: optimization proceeds without global-convexity guarantees, seeking only parameter improvements rather than certificates of global optimality. Moreover, the functions involved are differentiable almost everywhere, which makes this approximation a good practical choice.

In the smooth case, Clarke's [53, Chapter 2] chain-rule results [104] justify the approximation

$$\partial_{\boldsymbol{W}} \, \phi\big(\boldsymbol{\pi}_t(\boldsymbol{W})\big) \ \approx \ \big\{ \, \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 \ : \ \boldsymbol{v}_1 \in \partial_{\boldsymbol{\pi}} \, \phi\big(\boldsymbol{\pi}_t(\boldsymbol{W})\big), \ \boldsymbol{v}_2 \in \partial_{\boldsymbol{W}} \, \boldsymbol{\pi}_t(\boldsymbol{W}) \big\}. \tag{5.2}$$

Moreover, at any point where $\boldsymbol{W} \to \phi(\boldsymbol{\pi}_t(\boldsymbol{W}))$ is differentiable, the approximation in (5.2) becomes formally correct. Given that the set of non-differentiable points has Lebesgue measure zero, Equation (5.2) serves as a valid surrogate *gradient* in learning-based optimization, where the focus lies on improving parameters rather than certifying global optimality. Hence, this gradient approximation is effective for a learning task.

Here, $\partial_{\boldsymbol{\pi}}\phi(\boldsymbol{\pi}_t(\boldsymbol{W}))$ is a subgradient of ϕ at $\boldsymbol{\pi}_t(\boldsymbol{W})$, which can be computed under the assumption that ϕ is concave. Moreover, we can observe that it is already computed during the execution of the Bundle method. As discussed in Chapter 4, in the context of Lagrangian relaxation, we can find a subgradient by solving the subproblem to a given Lagrangian multiplier. Moreover, the subgradient corresponds to the violation of the relaxed constraints in the subproblem's primal solution for the given multipliers. Further details can be found in Equation (4.3) in Section 4.1.1.

In the case in which $\boldsymbol{\pi}_t(\boldsymbol{W})$ is differentiable, to compute the gradient $\partial_{\boldsymbol{W}}\boldsymbol{\pi}_t(\boldsymbol{W})$, it is necessary to unroll the iterative process used to obtain $\boldsymbol{\pi}_t(\boldsymbol{W})$. The exact computation of this gradient depends on the algorithm structure, and it is discussed in detail in the following sections.

5.1.1 Learning the step size

For the first approach, we aim to learn a sequence $\boldsymbol{\eta} \equiv \boldsymbol{\eta}(\boldsymbol{W}) \equiv (\eta_t(\boldsymbol{W}))_{t=1,\cdots,T}$ of regularization parameters. We follow here the Bundle method structure presented in Pseudocode 1 and we demonstrate how to obtain a subgradient for a point $\boldsymbol{\pi}_t(\boldsymbol{W})$ found at iteration t, considering $\boldsymbol{\eta}$ provided by parameters $\boldsymbol{W}$. Here $\boldsymbol{\pi}_t(\boldsymbol{W})$ depends on the entire sequence $\boldsymbol{\eta}$ generated by the model with parameters $\boldsymbol{W}$ during the execution.

We start by explicitly writing the trial point $\boldsymbol{\pi}_t(\boldsymbol{W})$ obtained in an iteration $t \leq T$. For each iteration $t \leq T$ we can define the partition of the iteration indexes into serious steps and null steps as $\{1, 2, \cdots, t\} = I_{SS}(t) \cup I_{NS}(t)$, as defined in Chapter 1. Serious steps refer to iterations in which the stabilization point is modified, while null steps correspond to those in which it remains unchanged. Further details can be found in Chapter 1.

Now that we have defined the terms, we can express the trial point as follows

$$\boldsymbol{\pi}_t(\boldsymbol{W}) = \bar{\boldsymbol{\pi}}_0 + \eta_t(\boldsymbol{W})\boldsymbol{w}^{(t)}(\eta_t(\boldsymbol{W})) + \left(\sum_{t' \in I_{SS}(t-1)} \eta_{t'}(\boldsymbol{W})\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})) \right).$$

where $\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})) = \sum_{i \in \beta_{t'}} \boldsymbol{g}_i \theta_i^{(t)}(\eta_{t'}(\boldsymbol{W}))$ is the new search direction, obtained as a convex combination of the gradients in the bundle $\beta_{t'}$ with coefficient provided by the solution $\boldsymbol{\theta}^{(t)}$ of the Problem (1.20).

It is important to note that for $t_1 \leq t_2$, the direction $\boldsymbol{w}^{(t_2)}(\eta_{t_2}(\boldsymbol{W}))$ depends on the entire trajectory up to iteration t_2. This means it depends on all the associated $\eta_{t_1}(\boldsymbol{W})$. This dependence is *indirect* in the sense that, for $t_1 \leq t_2$, $\boldsymbol{w}^{(t_2)}$ depends on η_{t_1} only through the gradient $\boldsymbol{g}_{t_1}$ and linearization error α_{t_1} found at iteration t_1.

Here, we forget this dependence by supposing that the search direction depends only on the regularization parameter of the same iteration. We will discuss this dependence in Section 5.1.2.

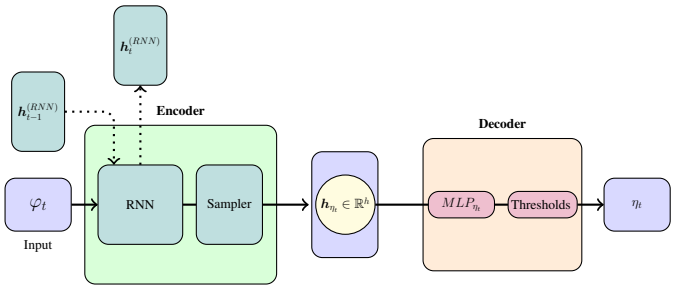

Figure 5.1: NN architecture predicting only the η_t at each iteration.

Using the property of linearity, we can decompose the associated subgradient as:

$$\partial_{\boldsymbol{W}} \boldsymbol{\pi}_t(\boldsymbol{W}) = \sum_{t' \in I_{SS}(t-1)} \partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W})\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W}))) + \partial_{\boldsymbol{W}}(\eta_t(\boldsymbol{W})\boldsymbol{w}^{(t)}(\eta_t(\boldsymbol{W}))).$$

Thus, it suffices to compute $\partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W})\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})))$ for all $t' \in I_{SS}(t-1)$ and for $t' = t$.

For a generic t', using the product rule:

$$\partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W})\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W}))) = \eta_{t'}(\boldsymbol{W})\partial_{\boldsymbol{W}}(\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W}))) + \partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W}))\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})).$$

Here $\partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W}))$ can be computed using Automatic Differentiation, as it only depends on the NN architecture used to predict $\eta_{t'}$.

NN architecture $\eta_t(\boldsymbol{W})$ We employ a Recurrent Neural Network consisting of first a Recurrent Neural Network layer, such as LSTM, to predict the mean $\boldsymbol{\mu} \in \mathbb{R}^h$ and the variance $\boldsymbol{\sigma} \in \mathbb{R}^h$ that will be the mean and the variance of a Gaussian distribution used to sample a hidden representation $\boldsymbol{h} \in \mathbb{R}^h$. We recall that the output of an RNN model is a sequence, but we always use the last one dynamically during the bundle execution. We use a parametrization trick to allow differentiation through this distribution, already presented in Equation 4.4 of Section 4.1.1. The size of the hidden representation, as well as other hyperparameters of the network, will be discussed in Section 5.2.1. This parametrization trick provides a Gaussian distribution with mean $\boldsymbol{\mu_h}$ and variance $\boldsymbol{\sigma_h}$ leading to $\boldsymbol{h} \sim \mathcal{N}(\boldsymbol{\mu_h}, Id_{\boldsymbol{h}} \odot \boldsymbol{\sigma_h})$.

The hidden representation $\boldsymbol{h}$ is provided as input to an MLP that predicts a scalar value. This scalar is ensured to lie within a range (between 0.00001 and 10000). More details on feature selection and hyperparameters of the NN architecture are provided in Section 5.2.

The search direction $\boldsymbol{w}^{(i)}(\eta_{t'}(\boldsymbol{W}))$ depends also on the regularization parameter $\eta_t(\boldsymbol{W})$, as it is obtained from the solution of the Problem (1.20). We can apply the chain rule, obtaining

$$\partial_\eta(\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})))\partial_{\boldsymbol{W}}(\eta_{t'}(\boldsymbol{W})).$$

The contribution $\partial_\eta(\boldsymbol{w}^{(t')}(\eta_{t'}(\boldsymbol{W})))$ complicates back-propagation. In the next Section, we provide further details on how to approximate that term.

5.1.2 Details on Computing Derivatives of the Direction for Learning the Step Size

This section is dedicated to computing the subgradient of the direction $\boldsymbol{w}^{(t_1)}$ with respect to the regularization parameters η_{t_2} of different iterations. We first provide details on computing the approximation of the subgradient of $\boldsymbol{w}^{(t_1)}$ with respect to the regularization parameters η_{t_1} at the same iterations. Then we show why we did not consider dependencies for further iterations $t_2 \neq t_1$.

Approximating $\partial_{\eta_t} \boldsymbol{w}^{(t)}(\eta_t)$

We denote by $\boldsymbol{\theta}^{(t)} \equiv \boldsymbol{\theta}^{(t)}(\eta_t)$ the solution of the DMP at iteration t, found using the regularization parameter η_t. In [82], one method is presented to solve the Dual Master problem iteratively and efficiently. Denoting by B the set of indices for which the solution of the Dual Master problem (1.20) at iteration t is positive, that is $\theta_i^{(t)} > 0$. We recall that $\boldsymbol{\theta}^{(t)}$ is an element of the $|\beta_t|$-dimensional Simplex, and so each component is comprised between 0 and 1, and the components of the vector sum to one. This vector represents the coefficients of the convex combination of the gradient in the bundle, which will be used to obtain the new trial direction. Moreover, we denote by $\boldsymbol{G}$ the matrix of the gradients contained in the bundle, with columns $\boldsymbol{g}_i$ for $i \in \beta_t$. Given a set of indexes B, we define the matrix $Q_B = \boldsymbol{G}_B^\top \boldsymbol{G}_B$ obtained as the product of the gradient matrices $\boldsymbol{G}$ restricted to the bundle components in B.

When Q_B is invertible, the solution found in [82] can be written as:

$$\boldsymbol{\theta}_B^{(t)} = Q_B^{-1} \left[\frac{\boldsymbol{e}}{\boldsymbol{e} Q_B^{-1} \boldsymbol{e}} + \frac{1}{\eta_t} \left(\frac{\boldsymbol{e} Q_B^{-1} \boldsymbol{\alpha}_B}{\boldsymbol{e} Q_B^{-1} \boldsymbol{e}} \boldsymbol{e} - \boldsymbol{\alpha}_B \right) \right]. \tag{5.3}$$

Where $\boldsymbol{e}$ is the vector, of size $|B|$, with all the components equal to one, $\boldsymbol{\alpha}_B$ is the vector composed of the linearization errors for the indices in B and Q_B^{-1} is the inverse of the matrix $\boldsymbol{G}^\top \boldsymbol{G}$ restricted to the indexes in B, and so it is well defined.

By definition of B, $\theta_i^{(t)} = 0$ for $i \notin B$. This allows us to find two vectors $\tilde{\boldsymbol{\theta}}_1^{(t)}$ and $\tilde{\boldsymbol{\theta}}_2^{(t)}$ such that the components in B can be computed as:

$$(\tilde{\boldsymbol{\theta}}_1^{(t)})_B = \frac{Q_B^{-1} \boldsymbol{e}}{\boldsymbol{e} Q_B^{-1} \boldsymbol{e}} \quad \text{and} \quad (\tilde{\boldsymbol{\theta}}_2^{(t)})_B = Q_B^{-1} \left(\frac{\boldsymbol{e} Q_B^{-1} \boldsymbol{\alpha}_B}{\boldsymbol{e} Q_B^{-1} \boldsymbol{e}} \boldsymbol{e} - \boldsymbol{\alpha}_B \right)$$

and they are zeros outside B.

Hence, we can express $\boldsymbol{\theta}^{(t)}(\eta_t)$ locally as:

$$\boldsymbol{\theta}^{(t)}(\eta_t) = \tilde{\boldsymbol{\theta}}_1^{(t)} + \frac{1}{\eta_t} \tilde{\boldsymbol{\theta}}_2^{(t)}$$

Note that $\tilde{\boldsymbol{\theta}}_1^{(t)}$ and $\tilde{\boldsymbol{\theta}}_2^{(t)}$ does not depend to η_t. Thus, we can compute the subgradient as:

$$\partial_\eta \boldsymbol{\theta}^{(t)}(\eta_t) = -\frac{1}{\eta_t^2} \tilde{\boldsymbol{\theta}}_2^{(t)}$$

The problem, in the case in which $rank(Q_B) < |B|$, is that Q_B is not invertible, and so we use this gradient strategy only when B defines a set $\{g_i\}_{i \in B}$ of independent vectors

and we put otherwise this subgradient to zero. This is a simple strategy, and further improvement can be discussed. Using the resolution process presented by Frangioni [82], only two cases are possible for the solution $\boldsymbol{\theta}^{(t)}$ of the Dual Master problem (1.20). The first case is the one previously used to approximate the subgradient. It is the case in which the non-zero components B are such that $\{\boldsymbol{g}_i\}_{i\in B}$ is a set of linearly independent vectors. The second case, in which we can still express $\boldsymbol{\theta}^{(t)}$ as a function of η_t, is when the non-zero components in B are such that removing only one vector $\{\boldsymbol{g}_i\}_{i\in B}$ the set contains only linear independent vectors. Using the resolution process presented by Frangioni [82], no other cases are possible, as when we obtain more than two linearly independent vectors, we can start a reduction process at the end of which we find one of the previous cases.

Other simple strategies are possible. For example, to compute the gradient, we can remove indices from B, associated with small values of $\theta_i^{(t)}$, up to obtaining a set of independent vectors.

Given $\boldsymbol{\theta}^{(t)}(\eta_t)$ the search direction $\boldsymbol{w}^{(t)}(\eta_t) = \sum_{i\in\beta_t} \boldsymbol{g}_i\theta_i^{(t)}(\eta_t) = \boldsymbol{G}\boldsymbol{\theta}^{(t)}(\eta_t)$ and then

$$\partial_\eta \left(\boldsymbol{G}\boldsymbol{\theta}^{(t)}(\eta_t)\right) = -\frac{1}{\eta_t^2}\boldsymbol{G}\tilde{\boldsymbol{\theta}}_2.$$

Approximating $\partial_{\eta_{t_1}} \boldsymbol{w}^{(t_2)}$ for $t_1 \leq t_2$

As seen in the previous section, we compute the derivative $\partial_{\eta_t} \boldsymbol{w}^{(t)}$, that is a gradient of the search direction with respect to the regularization parameter η_t of the same iteration t, using an analytical formulation. This section focuses on an analysis of the dependencies of the search direction with respect to the η parameters of *other* iterations.

As previously mentioned, these cross-iteration dependencies are omitted in practical computations. Therefore, this section does not propose practical approximation methods for these dependencies. Instead, it offers a formal justification for their omission.

Here, we slightly abuse notations by allowing the sum and product of sets to denote the corresponding set of all possible sums or products of their elements. Technically, subgradients are sets, not vectors, but in practice, we typically work with a single vector from the subgradient.

We also employ the chain rule at several moments, which is non-trivial for subgradients, even in the context of concave/convex functions. Furthermore, the functions involved in the resolution process may not be convex or concave. This implies that the *classical* subgradient may not exist, and we must instead rely on a generalized notion of *local* subgradient, such as Clarke subgradient [53].

Observe that $\partial_{\eta_{t_1}} \boldsymbol{w}^{(t_2)} = 0$ when $t_1 \geq t_2$. Since η_{t_1} does not influence the optimal value $\boldsymbol{w}^{(t_2)}$ of prior iterations, the subgradient is null for $t_1 \geq t_2$. Hence, we only focus on $t_1 \leq t_2$.

It is possible to observe that the direction $\boldsymbol{w}^{(t_2)}$ depends on η_{t_1} only through the trial point $\boldsymbol{\pi}_{t_1} = \bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1}\boldsymbol{w}^{(t_1)}$. More precisely, we can see that the solution of the Dual Master problem 1.20 depends only on the gradient $\boldsymbol{g}_{t_1}$ and the linearization error α_{t_1} associated with this point. Hence, to compute $\partial_{\eta_{t_1}} \boldsymbol{w}^{(t_2)}$, it becomes necessary to compute the dependencies of this gradient and linearization error: $\partial_{\eta_{t_1}} \boldsymbol{g}_{t_1} \partial_{\boldsymbol{g}_{t_1}} \boldsymbol{w}^{(t_2)}$

and $\partial_{\eta_{t_1}} \alpha_{t_1} \partial_{\alpha_{t_1}} \boldsymbol{w}^{(t_2)}$.

For both $\partial_{\alpha_{t_1}} \boldsymbol{w}^{(t_2)}$ and $\partial_{\boldsymbol{g}_{t_1}} \boldsymbol{w}^{(t_2)}$ we can use the same approximation presented in Equation 5.3 for estimating $\partial_{\eta_{t_2}} \boldsymbol{w}^{(t_2)}$.

For the term $\partial_{\eta_{t_1}} \alpha_{t_1}$ we recall the definition of linearization error

$$\alpha_{t_1} = \phi(\bar{\boldsymbol{\pi}}_{t_1-1}) - \phi(\boldsymbol{\pi}_{t_1}) + \boldsymbol{g}_{t_1}^\top(\boldsymbol{\pi}_{t_1} - \bar{\boldsymbol{\pi}}_{t_1-1}).$$

Since the stabilization point is fixed before iteration t_1, it is independent of η_{t_1}:

$$\partial_{\eta_{t_1}} \alpha_{t_1} = -\partial_{\eta_{t_1}} \phi(\boldsymbol{\pi}_{t_1}) + \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top \boldsymbol{\pi}_{t_1} \right)$$

Using the trial point definition $\boldsymbol{\pi}_{t_1+1} = \bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}$ we can rewrite this as

$$\partial_{\eta_{t_1}} \alpha_{t_1} = -\partial_{\eta_{t_1}} \phi(\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) + \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) \right).$$

Applying the chain rule:

$$\partial_{\eta_{t_1}} \alpha_{t_1} = -\partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1}) \partial_{\eta_{t_1}} (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) + \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) \right).$$

Using the product rule for derivatives and the definition of $\boldsymbol{g}_{t_1} = \partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1})$ we obtain

$$\partial_{\eta_{t_1}} \alpha_{t_1} = -\boldsymbol{g}_{t_1} \partial_{\eta_{t_1}} (\eta_{t_1} \boldsymbol{w}^{(t_1)}) + \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top \right) (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) + \boldsymbol{g}_{t_1}^\top \partial_{\eta_{t_1}} \left(\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)} \right).$$

By linearity and using the fact that $\bar{\boldsymbol{\pi}}_{t_1-1}$ does not depends on η_{t_1} we have

$$\partial_{\eta_{t_1}} \alpha_{t_1} = -\boldsymbol{g}_{t_1}^\top \partial_{\eta_{t_1}} \left(\eta_{t_1} \boldsymbol{w}^{(t_1)} \right) + \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top \right) (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) + \boldsymbol{g}_{t_1}^\top \partial_{\eta_{t_1}} \left(\eta_{t_1} \boldsymbol{w}^{(t_1)} \right)$$

and so

$$\partial_{\eta_{t_1}} \alpha_{t_1} = \partial_{\eta_{t_1}} \left(\boldsymbol{g}_{t_1}^\top \right) (\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}).$$

We then reduce the problem to computing the gradient of $\boldsymbol{g}_{t_1}$ with respect to η_{t_1}. We have

$$\partial_{\eta_{t_1}} \boldsymbol{g}_{t_1} = \partial_{\eta_{t_1}} \partial_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1}) = \partial_{\eta_{t_1}} \partial_{\boldsymbol{\pi}} \phi(\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}).$$

Even if ϕ were differentiable, this would yield

$$\partial_{\eta_{t_1}} \boldsymbol{g}_{t_1} = \partial_{\eta_{t_1}} \nabla_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1}) = \partial_{\eta_{t_1}} \nabla_{\boldsymbol{\pi}} \phi(\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) =$$

$$= \nabla^2_{\boldsymbol{\pi}} \phi(\bar{\boldsymbol{\pi}}_{t_1-1} + \eta_{t_1} \boldsymbol{w}^{(t_1)}) \partial_{\eta_{t_1}} (\eta_{t_1} \boldsymbol{w}^{(t_1)}) = \nabla^2_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1}) \partial_{\eta_{t_1}} (\boldsymbol{w}^{(t_1)}) \boldsymbol{w}^{(t_1)}.$$

Hence, computing $\partial_{\eta_{t_1}} \boldsymbol{g}_{t_1}$ and $\partial_{\eta_{t_1}} \boldsymbol{\alpha}_{t_1}$ ultimately requires evaluating the Hessian $\partial^2_{\boldsymbol{\pi}} \phi(\boldsymbol{\pi}_{t_1})$, i.e., a second-order subgradient. Even in the twice-differentiable setting, this will be expensive.

However, this is not feasible in general. While an approximation of this second-order term might be theoretically possible, such an investigation falls beyond the scope of this thesis.

5.1.3 Bundle Network: a Machine-Learning Based Bundle Method

In this section, we propose an alternative machine learning-based approach that simultaneously learns both the step size (i.e., regularization parameter) and the search direction. Unlike the previous approach, we do not provide an *analytical* differentiation of this approach, as it is based solely on operations that can be differentiated using automatic differentiation tools commonly employed in machine learning. Furthermore, this approach allows us to capture *longer-term* dependencies compared to the method presented in the previous section. This is particularly interesting because this method could be combined with one that learns the initialization, such as the one presented in Chapter 4. This aspect is not explicitly developed in this thesis.

For this approach, we restrict ourselves to the case in which the Bundle is used to solve unconstrained optimization problems. In the case of the Lagrangian relaxation, this corresponds to dualizing equality constraints. Further generalizations are possible, but not implemented, and we refer to Chapter 6 for more details. We begin by considering an initial point $\boldsymbol{\pi}_0 \in \mathbb{R}^n$ and setting the initial stabilization point $\bar{\boldsymbol{\pi}}_0$ equal to the starting trial point $\boldsymbol{\pi}_0$. Next, we compute the subgradient $\mathbf{g}_0 = \partial\phi(\boldsymbol{\pi}_0)$ and the linearization error $\alpha_0 = 0$, since the trial point and the stabilization point coincide. We add this information to the bundle.

At each iteration, we proceed as follows. First, we compute the features associated with the last component added to the bundle; we do not back-propagate through the feature extraction. More details on feature extraction can be found in Section 5.2.1. These features are fed as input to a neural network, which outputs a vector $\boldsymbol{\theta}^{(t)} \in \Delta^{|\beta_t|}$, in the $|\beta_t| = t$ dimensional simplex, and a step size η_t. More details on the network architecture can be found in Section 5.2. The vector $\boldsymbol{\theta}^{(t)}$ is used, as in the standard Bundle method, to compute the search direction $\boldsymbol{w}^{(t)}$ as a convex combination of the gradients in the bundle β_t, i.e. $\boldsymbol{w}^{(t)} = \sum_{i=0}^{|\beta_t|} \theta_i^{(t)} \boldsymbol{g}_i$. This direction is then used to update the trial point as $\boldsymbol{\pi}_{t+1} = \bar{\boldsymbol{\pi}}_t + \eta_t \boldsymbol{w}^{(t)}$. We then compute the objective value, the subgradient, and the linearization error in the new trial point, adding this information to the bundle:

$$\beta_{t+1} = \beta_t \cup \{\boldsymbol{g}_{t+1}, \alpha_{t+1}, v_{t+1}\}.$$

In the *classic* Bundle method, presented in Chapter 1, the stabilization point at iteration t is updated whenever

$$\phi(\boldsymbol{\pi}_{t+1}) - \phi(\bar{\boldsymbol{\pi}}_t) > m \cdot (\eta_t \cdot || \sum_{i \in \beta_t} \theta_i^{(t)} \boldsymbol{g}_i||^2 + \sum_{i \in \beta_t} \alpha_i \theta_i^{(t)})$$

where the right-hand side corresponds to the objective value of Problem (1.20), scaled by a constant $0 \leq m < 1$, typically small (e.g., 0.01). When $m = 0$, this rule updates the stabilization point whenever the new trial point yields a strictly better value of the objective function ϕ. Even using the simpler condition $\phi(\boldsymbol{\pi}_{t+1}) > \phi(\bar{\boldsymbol{\pi}}_t)$ still leads to a non-differentiable update mechanism. Indeed, this operation is piecewise constant, and its derivative, when it exists, is zero almost everywhere.

We begin by observing that, denoting by $\boldsymbol{r} = \arg\max((\phi(\boldsymbol{\pi}_{t+1}), \phi(\bar{\boldsymbol{\pi}}_t)))$ the one-hot vector encoding equal to $(1, 0)$ if $\phi(\boldsymbol{\pi}_{t+1}) \geq \phi(\bar{\boldsymbol{\pi}}_t)$ and $(0, 1)$ otherwise. So it is a vertex of the two-dimensional simplex. The update of the stabilization point can be

rewritten as:

$$\boldsymbol{r}^\top(\boldsymbol{\pi}_{t+1}, \bar{\boldsymbol{\pi}}_t) = r_1\boldsymbol{\pi}_{t+1} + r_2\bar{\boldsymbol{\pi}}_t.$$

Still, there is a concavity/convexity issue with the updates as the stabilization point remains the same since $\phi(\boldsymbol{\pi}_{t+1}) \geq \phi(\bar{\boldsymbol{\pi}}_t)$ and it moves with non-continuity to $\boldsymbol{\pi}_{t+1}$ when the previous condition is no longer satisfied. Moreover, the gradient is zero almost everywhere, and it is not defined at the singular point. A common strategy in machine learning to obtain smoother approximations of the $\arg\max$ operator is to replace it with a soft version, such as the softmax. To ensure smooth updates of the stabilization point, we apply the softmax of the objective value of the stabilization and the new trial point. Hence, we consider:

$$\boldsymbol{r} = \textbf{softmax}(\phi(\boldsymbol{\pi}_{t+1}), \phi(\bar{\boldsymbol{\pi}}_t))$$

and the updates are performed as previously. This corresponds to choosing a possibly different convex combination of the two vectors, which will not necessarily be a vertex of the two-dimensional simplex but any point in it.

We iterate up to a certain fixed number of iterations T. The rationale behind the fact that we consider only the objective values to update the stabilization point, ignoring the term associated with the objective value of the Dual Master problem, can be found in the fact that the SS formula, line 12 of Algorithm 1, is crucial for the abstract convergence proofs, but setting $m = 0$ does not substantially affects the practical behavior of the method.

Algorithm 3 Machine learning-based version of the Bundle method 1 that substitutes the resolution of the Master problem and the non-continuous operations with a smoother version based on neural networks.

1: **Choose** $\boldsymbol{\pi}_0, T$
2: $\bar{\boldsymbol{\pi}}_0 \leftarrow \boldsymbol{\pi}_0$ ▷ Initialize stabilization point
3: $(\boldsymbol{g}_0, \boldsymbol{\alpha}_0, v_0) \leftarrow (\partial\phi(\boldsymbol{\pi}_0), 0, \phi(\boldsymbol{\pi}_0))$
4: $\beta_0 \leftarrow \{(\boldsymbol{g}_0, \boldsymbol{\alpha}_0)\}$
5: **for** $t = 1, \cdots, T$ **do**
6: $\varphi_t =$ features_extraction (β_t) ▷ We do not back-propagate through the feature extraction
7: $\eta_t, \boldsymbol{\gamma}^{(t)} \leftarrow nn(\varphi_t)$ ▷ Output of a neural network
8: $\boldsymbol{\theta}^{(t)} \leftarrow sparsemax(\boldsymbol{\gamma}^{(t)})$ ▷ Approximate Solution of the DMP 1.20
9: $\boldsymbol{w}^{(t)} \leftarrow \sum_{i=0}^{|\beta_t|} \boldsymbol{g}_i\boldsymbol{\theta}_i^{(t)}$ ▷ New trial direction
10: $\boldsymbol{\pi}_{t+1} \leftarrow \bar{\boldsymbol{\pi}}_t + \eta_t\boldsymbol{w}^{(t)}$ ▷ Compute new trial point
11: $(\boldsymbol{g}_{t+1}, \boldsymbol{\alpha}_{t+1}, v_{t+1}) \leftarrow (\partial\phi(\boldsymbol{\pi}_{t+1}), \phi(\bar{\boldsymbol{\pi}}_t) - \phi(\boldsymbol{\pi}_{t+1}) + \boldsymbol{g}_{t+1}^\top(\boldsymbol{\pi}_{t+1} - \bar{\boldsymbol{\pi}}_t), \phi(\boldsymbol{\pi}_{t+1}))$ ▷ Gradient and linearization error
12: $\beta_{t+1} \leftarrow \beta_t \cup (\boldsymbol{g}_{t+1}, \boldsymbol{\alpha}_{t+1}, v_{t+1})$ ▷ Update Bundle
13: $\bar{\boldsymbol{\pi}}_{t+1} \leftarrow softmax(\phi(\boldsymbol{\pi}_{t+1}), \phi(\bar{\boldsymbol{\pi}}_t)) \odot (\boldsymbol{\pi}_{t+1}, \bar{\boldsymbol{\pi}}_t)$ ▷ Smooth updates of the stabilization point
14: Update $\alpha_i\ \forall i = 0, \cdots, |\beta_t|$ ▷ Update the linearization errors
15: **end for**

Step and Direction Architecture

This section provides further details on the architecture, schematically presented in Figure 5.2, which is used to predict η_t and $\boldsymbol{\theta}^{(t)}$.

Given the features $\varphi_t \in \mathbb{R}^d$ at the current bundle β_t, a Recurrent Neural Network, specifically an LSTM layer, maps this vector to a hidden space by predicting six vectors of size h. We denote these vectors as $\mu_{\boldsymbol{h}_{q_t}}, \sigma_{\boldsymbol{h}_{q_t}}, \mu_{\boldsymbol{h}_{k_t}}, \sigma_{\boldsymbol{h}_{k_t}}, \mu_{\boldsymbol{h}_{\eta_t}}, \sigma_{\boldsymbol{h}_{\eta_t}} \in \mathbb{R}^h$, and they represent the mean and the variance used to extract, by sampling through a Gaussian distribution, the hidden representation of the keys and the queries used to represent the current iterations as in attention mechanisms [247], as well as the hidden representation for η_t. More precisely, to obtain the hidden representations $\boldsymbol{h}_{q_t}, \boldsymbol{h}_{k_t}, \boldsymbol{h}_{\eta_t}$ we perform the same parametrization trick as for the other model, presented in Equation 4.4. The size of the hidden representations and other hyperparameters of the network will be discussed in Section 5.2.1.

The sampled representations are then fed into three independent Multi-Layer Perceptrons that predict a scalar η_t and two vectors $\boldsymbol{q}_t$ and $\boldsymbol{k}_t$.

The vectors $\boldsymbol{q}_t$ and $\boldsymbol{k}_t$ are, respectively, the query and the key of the current iteration. The vector $\boldsymbol{k}_t$ is stored in memory for use in subsequent iterations. Using $\boldsymbol{q}_t$ and all the vectors $\{\boldsymbol{k}_i\}_{i=0}^{|\beta_t|}$, we can compute a scalar for each component in the bundle using an *Attention Layer* defined as follows:

$$\boldsymbol{\gamma}^{(t)} = (\boldsymbol{k}_i^\top \boldsymbol{q}_t)_{i=0}^{|\beta_t|} \in \mathbb{R}^t.$$

To obtain the approximate solution of the Dual Master problem 1.20, we pass this vector to a sparsemax function:

$$\boldsymbol{\theta}^{(t)} = \psi(\boldsymbol{\gamma}^{(t)}) \in \Delta^t$$

here $\Delta^t = \left\{\boldsymbol{\theta} \in [0,1]^{t+1} \mid \sum_{i=1}^{t+1} \theta_i = 1\right\}$ is a simplex set and $\psi : \mathbb{R}^m \to \Delta^m$ is a function assuming values in the simples, some examples are **softmax** or **sparsemax**. This yields a vector $\boldsymbol{\theta}^{(t)}$ whose components lie in $[0, 1]$ and sum to one.

Sparsemax [176] has already been discussed in Section 3.2.2. It is a differentiable function that allows mapping an n-dimensional vector to one element of the simplex. It is similar to softmax, presented in Section 2.3.4, but it allows for obtaining a sparser vector, in the sense that more components are equal to zero.

5.2 Numerical Results

In this section, we compare our approaches to classical iterative methods for solving the Lagrangian Dual, specifically those based on subgradient information. In particular, we evaluate their performance against both simple subgradient-based methods and the classical Bundle method, using different heuristic η-strategies.

More specifically, we consider ADAM and Gradient Ascent using a simple strategy for regularizing the step size: if we pass more than two iterations without improving the objective value, we divide the step size by two. These two baselines are well-established in the context of concave maximization (or convex minimization) and serve as relevant

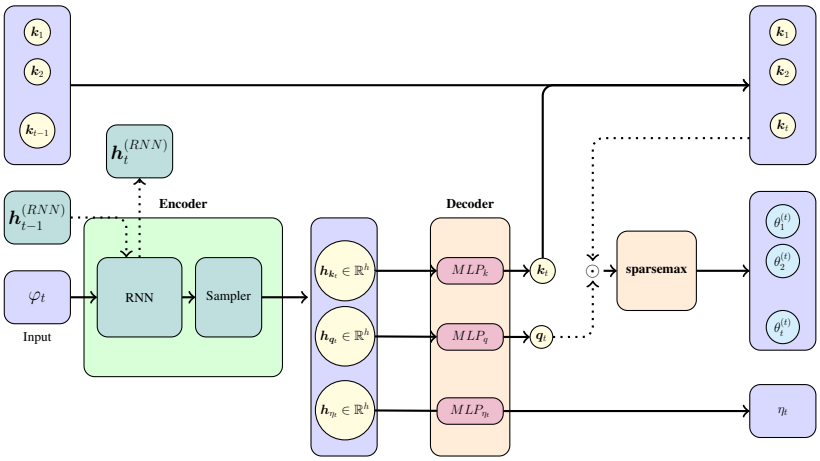

Figure 5.2: Schematic representation of the architecture used to predict the step-size η_t and the weights $\boldsymbol{\theta}^{(t)}$ for the convex combination of the gradients in the bundle at the current iteration.

points of comparison. They are of particular interest because they are typically faster than the Bundle method, relying only on the latter point (for Descent) and the gradient of the last inserted point and the previous search direction. These two methods are discussed in detail in Section 1.3 of Chapter 1.

We further compare our approach with *classic* variants of the Bundle method. For the Bundle method, we consider the four long-term η-strategies presented in Section 1.3.3 of Chapter 1, inspired by the ones used in the state-of-the-art bundle-based *BundleSolver* component of SMS++ [88]. With the non-constant ones, we also consider the middle-term η-strategy and the short-term $\eta-$strategy proposing increasing as $\eta_{t+1} = 1.1 \cdot \eta_t$ and decreasing as $\eta_{t+1} = \eta_t \cdot 0.9$, presented in the same Section. For all these six *classic approaches*, we perform a grid search to choose the best initial parameter η_0 considering different values. We evaluate the initialization of the η_0 parameter using the values 10^4, 10^3, 10^2, 10^1, 10^0 and 10^{-1}. For each fixed maximum number of iterations (10, 25, 50, and 100), we select the value of η_0 that achieves the lowest average GAP over the test set. The results of this grid search are summarized in Table 5.1.

The values are so large, so we rescale the objective ϕ dividing the function value using the norm $\sqrt{\boldsymbol{g}_0^\top \boldsymbol{g}_0}$ of the gradient $\boldsymbol{g}_0$ in the starting point $\boldsymbol{\pi}_0$, always taken equal to the zero vector. Note that even if the objective value changes, the optimum of the function ϕ and $\frac{\phi}{\sqrt{\boldsymbol{g}_0^\top \boldsymbol{g}_0}}$ are the same. Exploring alternative rescaling techniques could be an interesting direction for future work, but this is beyond the scope of the present thesis.

We consider the Lagrangian Dualof the Multi-Commodity Network Design presented in Section 4.2.1 of Chapter 4. We consider the same datasets as in the previous Chapter, plus a larger dataset that was not possible to consider previously due to the large size of the bipartite graph representation.

For each instance ι on which we evaluate our model, the Lagrangian relaxation is solved using SMS++. The obtained Lagrangian multipliers, denoted by $\boldsymbol{\pi}_\iota^*$, are used for computing the GAP, already presented in Chapter 4, that serves as a measure of optimality for all methods.

5.2.1 Hyperparameters and Implementation Details

In this subsection, we provide technical details about the implementation. More specifically, we specify the hyperparameters of the NN architecture (regarding layers, layer sizes, and activation functions) for both architectures: the one that learns only the step size and the one that learns the step size and the direction. We also define the features (hand-crafted). We specify the hyperparameters for the optimizer used for the training and the GPU and CPU specifics of the machines used for the numerical experiments. Finally, we define the metrics and the validation techniques used to analyze the quality of the provided solutions.

Loss functions To weigh the contributions of the different iterations, we use $\gamma = 0.999$. In other works focusing on learning optimizers, as [10], this parameter is taken equal to 1. For the first approach, we consider a slight variant for the γ in the loss. More precisely, we consider a telescopic sum only for the contributions of the serious steps. Meanwhile, we use a smaller contribution (constant) for all the Null Steps equal to 0.00001, which is smaller than all the (possible) contributions of the null steps ($0.00001 \ll 0.999^{10} \approx 0.990$). As

Model Architecture - ML-η-Learning For all datasets, the model first predicts a hidden representation of size 64 using a Recurrent Neural Network (LSTM, presented in Section 2.2.4 of Chapter 2).

This hidden representation $\boldsymbol{h}$ of size 128 is provided to a a Multi-Layer Perceptron composed of two hidden layers of size 512 and 128, considering a softplus activation function. Then, a final output layer predicts a two-component vector. These represent the mean and the standard deviation of a Gaussian distribution used to sample the hidden representation of η_t. The parameter η_t is then predicted as the output of an MLP, and it is set into a predefined interval (in between 10^{-6} and 10^4)

Dataset	Methods	10 iter.	25 iter.	50 iter.	100 iter.
MC-SML-40	Bundle hard	100.0	10.0	10.0	10.0
	Bundle balancing	100.0	10.0	10.0	100.0
	Bundle soft	100.0	100.0	100.0	100.0
	Bundle constant	100.0	100.0	100.0	100.0
	Descent	10.0	10.0	10.0	10.0
	Adam	1.0	1.0	1.0	1.0
MC-SML-VAR	Bundle hard	100.0	100.0	100.0	100.0
	Bundle balancing	100.0	100.0	100.0	100.0
	Bundle soft	100.0	100.0	100.0	100.0
	Bundle constant	100.0	100.0	100.0	100.0
	Descent	100.0	10.0	10.0	10.0
	Adam	1.0	1.0	1.0	1.0
MC-BIG-40	Bundle hard	10.0	10.0	10.0	10.0
	Bundle balancing	10.0	10.0	10.0	10.0
	Bundle soft	10.0	10.0	10.0	10.0
	Bundle constant	10.0	10.0	10.0	10.0
	Descent	10.0	10.0	10.0	10.0
	Adam	1.0	1.0	1.0	1.0
MC-BIG-VAR	Bundle hard	10.0	10.0	10.0	100.0
	Bundle balancing	10.0	10.0	10.0	10.0
	Bundle soft	10.0	10.0	10.0	10.0
	Bundle constant	100.0	10.0	10.0	10.0
	Descent	10.0	10.0	10.0	10.0
	Adam	1.0	1.0	1.0	1.0
MC-CANADN	Bundle hard	100.0	100.0	100.0	10.0
	Bundle balancing	100.0	100.0	100.0	100.0
	Bundle soft	100.0	100.0	100.0	100.0
	Bundle constant	100.0	100.0	100.0	100.0
	Descent	100.0	100.0	10.0	10.0
	Adam	10.0	1.0	1.0	1.0

Table 5.1: Best average initialization values for the η_0 parameter over the test set for all methods that require a grid search. The values $\{10^4, 10^3, 10^2, 10^1, 10^0, 10^{-1}\}$ are evaluated, and for each fixed maximum number of iterations (10, 25, 50, and 100), the value yielding the lowest average GAP is selected.

Model Architecture - Bundle Network In this case, we consider a first Recurrent Neural Network model (LSTM), presented in Chapter 2 that goes from the features space to predict 6 vectors of size 128. These are the means and variances used to sample three vectors that are the hidden representation of the η parameter, the one for the key for the last found component, and the one for the query for the current iteration.

Each of these three representation are then passed as input to a different decoder (so we have here 3 different decoders), each of which has the same architecture. We only consider a hidden layer of size $8 \cdot 6 \cdot 128$, and we produce a vector of size 128 for the key and the query, and only one scalar for η_t.

Features At iteration t, we perform a human-designed feature extraction, chosen to provide similar information to that used by the heuristics $\eta-$strategies. We add further features to represent the component added to the bundle in the associated iteration, and we add further features to characterize this entry and to compare it with previously added components. For the two approaches, we consider the same features to represent the information gained in the last iteration. They can be divided into three categories.

Features related to iteration t, summarizing the optimization dynamics at the current iteration:

- the last step-size η_{t-1},
- The square norm of the search direction $||\boldsymbol{w}^{(t-1)}||_2^2$,
- The square norm of the search direction weighted with η_t, that is $\eta_{t-1}||\boldsymbol{w}^{(t-1)}||_2^2$. This corresponds to the quadratic part in the DMP objective function,
- The linear part in the DMP: $\sum_{j=1}^{t-1} \alpha_j \theta_j^{(t-1)}$,
- a boolean value to see if the quadratic part is bigger than the linear part $||\boldsymbol{w}^{(t-1)}||_2^2 > \sum_{j=1}^{t-1} \alpha_j \theta_j^{(t-1)}$,
- a boolean to see if the quadratic part, rescaled by a *big* $\eta^* = 10000$, is greater than the linear part: $\eta^* ||\boldsymbol{w}^{(t-1)}||_2^2 > \sum_{j=1}^{t-1} \alpha_j \theta_j^{(t-1)}$,
- The iteration counter t.

Features describing the last trial and stabilization points. These features capture information about the most recent trial point $\boldsymbol{\pi}_t$ and stabilization point $\bar{\boldsymbol{\pi}}_t$:

- The objective value in the last trial point $\phi(\boldsymbol{\pi}_t)$ and in the last stabilization point $\phi(\bar{\boldsymbol{\pi}}_t)$,
- The linearization error in the last stabilization point $\bar{\alpha}_t$ and in the last trial point α_t,
- The square norm of the last trial point $||\boldsymbol{\pi}_t||_2$ of the last stabilization point $||\bar{\boldsymbol{\pi}}_t||_2$ and of the gradient in the stabilization point $||\bar{\boldsymbol{g}}_t||_2$,
- the square norm, the mean, the variance, the minimum, and the maximum of the vector $\boldsymbol{g}_t$,

- the square norm, the mean, the variance, the minimum, and the maximum of the vector $\boldsymbol{\pi}_t$.

Features comparing the last inserted component with the ones already contained in the bundle:

- The minimum and the maximum of the scalar product of the gradients in the bundle $\min_j(\boldsymbol{g}_t^\top \boldsymbol{g}_j)$ and $\max_j(\boldsymbol{g}_t^\top \boldsymbol{g}_j)$,
- The minimum and the maximum of the scalar product of the points in the bundle $\min_j(\boldsymbol{\pi}_t^\top \boldsymbol{\pi}_j)$ and $\max_j(\boldsymbol{\pi}_t^\top \boldsymbol{\pi}_j)$,
- The scalar product of the last inserted gradient and the last search direction $\boldsymbol{g}_t^\top \boldsymbol{w}^{(t-1)}$.

Optimiser Specifications We use Adam as optimizer, with a learning rate 0.00001, a Clip Norm (to 5), and exponential decay 0.9.

GPU specifics For the training on the datasets, we use Nvidia A40 GPUs accelerators with 48GB of RAM. We test performance on the same machine. All the variants, except for Bundle Network, are CPU-only based. The experiments are done on the same machine with QEMU Virtual CPU version 2.5+.

Metrics We use the percentage GAP as a metric to evaluate the quality of the bounds as defined in Chapter 4.

Data for Evaluation We also divide each dataset used in Chapter 4, composed of 2000 instances, into train (1000), validation (500), and test (500) sets. For the newly generated dataset, the validation and training sets each consist of an equal number of instances, while the test set consists of 100 instances. Parameters are learned on the training set, model selection is performed on the validation set, and test proxies for unseen data.

The higher times for ML-η-strategy in Table 5.2 are due to the fact that the predictions of this model are performed on CPU. A better implementation, considering predictions on GPU, will surely be advantageous.

5.2.2 Bound Accuracy

Table 5.2 reports the average performances of different algorithms for the resolution of the Lagrangian Dual on our datasets. For the heuristic strategies that did not use machine learning, we consider a grid search for the initialization of the step-size/regularization parameter. We consider values in between 10^4 and 10^{-1}, after ensuring that the best ones lie strictly in this interval. For each fixed number of iterations ($10,\ 25,\ 50$ and 100), we choose the η_0 associated with lower final GAP.

	Methods	10 iter.		25 iter.		50 iter.		100 iter.	
		GAP	time	GAP	time	GAP	time	GAP	time
MC-SML-40	Bundle h.	20.18	0.044	6.68	0.117	2.84	0.269	1.17	0.767
	Bundle b.	17.68	0.054	6.09	0.145	1.84	0.327	0.41	0.898
	Bundle s.	17.58	0.044	6.10	0.116	1.82	0.271	0.36	0.778
	Bundle c.	11.96	0.048	4.16	0.128	1.30	0.293	0.31	0.816
	Descent	44.66	**0.008**	14.11	**0.021**	4.70	**0.044**	2.24	**0.093**
	Adam	48.87	0.011	12.84	0.027	3.04	0.055	1.89	0.109
	Bundle n.	**9.20**	0.105	**1.75**	0.185	**0.53**	0.321	**0.17**	0.592
	Bundle η-l.	28.3	0.088	5.15	0.219	1.14	0.486	0.34	1.791
MC-SML-VAR	Bundle h.	18.27	0.094	9.55	0.254	4.08	0.564	1.96	1.433
	Bundle b.	15.83	0.124	7.39	0.334	2.88	0.736	0.74	1.749
	Bundle s.	15.82	0.094	6.92	0.256	2.58	0.576	0.67	1.421
	Bundle c.	12.79	0.108	4.33	0.29	1.66	0.635	0.58	1.503
	Descent	62.43	**0.025**	28.64	**0.066**	11.14	**0.142**	4.00	**0.303**
	Adam	51.48	0.034	15.42	0.086	4.74	0.17	2.75	0.341
	Bundle n.	**12.60**	0.138	**3.28**	0.269	**1.24**	0.496	**0.50**	0.952
	Bundle η-l.	23.17	0.206	4.89	0.531	1.61	1.144	0.55	2.635
MCND-BIG-40	Bundle h.	20.66	0.07	8.54	0.185	5.32	0.410	2.88	0.986
	Bundle b.	20.67	0.084	7.37	0.222	3.17	0.485	1.24	1.209
	Bundle s.	20.67	0.07	7.37	0.184	3.17	0.407	1.19	1.042
	Bundle c.	19.34	0.076	5.64	0.203	2.08	0.447	0.78	1.110
	Descent	28.53	**0.023**	14.12	**0.058**	9.91	**0.117**	8.21	**0.241**
	Adam	24.89	0.025	9.38	0.064	7.30	0.129	6.90	0.256
	Bundle n.	**12.70**	0.130	**3.41**	0.243	**1.26**	0.436	**0.47**	0.832
	Bundle η-l.	23.75	0.161	8.56	0.41	2.56	0.837	0.82	1.834
MC-BIG-VAR	Bundle h.	26.23	0.095	9.99	0.248	4.41	0.550	2.23	1.386
	Bundle b.	21.82	0.123	10.01	0.324	3.46	0.695	1.08	1.652
	Bundle s.	21.83	0.094	9.53	0.253	3.43	0.550	1.02	1.353
	Bundle c.	**17.98**	0.112	6.34	0.289	2.66	0.627	0.93	1.477
	Descent	54.11	**0.029**	24.69	**0.074**	10.58	**0.154**	4.85	**0.323**
	Adam	44.21	0.034	13.75	0.086	5.42	0.170	3.84	0.345
	Bundle n.	20.10	0.139	**5.11**	0.271	**2.01**	0.495	**0.70**	0.953
	Bundle η-l.	25.83	0.198	8.22	0.506	2.31	1.084	0.75	2.463
MC-CANADN	Bundle h.	31.16	0.47	16.88	1.219	10.2	2.491	5.65	6.064
	Bundle b.	31.65	0.546	14.45	1.41	7.18	2.961	3.07	6.862
	Bundle s.	31.6	0.473	13.92	1.219	6.82	2.629	**3.01**	6.416
	Bundle c.	**26.74**	0.495	**11.93**	1.263	**6.17**	2.699	3.17	6.504
	Descent	48.09	**0.380**	41.24	**0.977**	33.66	**1.9**	19.44	3.856
	Adam	50.18	0.396	31.88	0.934	16.88	1.902	8.22	**3.853**
	Bundle n.	29.73	0.491	14.34	1.23	8.05	2.474	4.48	4.98
	Bundle η-l.	31.03	5.424	14.93	13.736	7.39	29.102	3.81	63.606

Table 5.2: Comparison between our approach and several baseline methods. For the baselines, the initial regularization parameter or step size is selected via grid search, whereas our method requires no hyperparameter tuning and is trained for only 10 iterations. The Methods column lists various Bundle implementations (abbreviated by their initial letters). Specifically, Bundle hard, Bundle soft, Bundle balancing, and Bundle constant use grid search to tune the initial η parameter. In contrast, Bundle network and Bundle η-learning require model training prior to use and correspond to the two approaches introduced in this chapter.

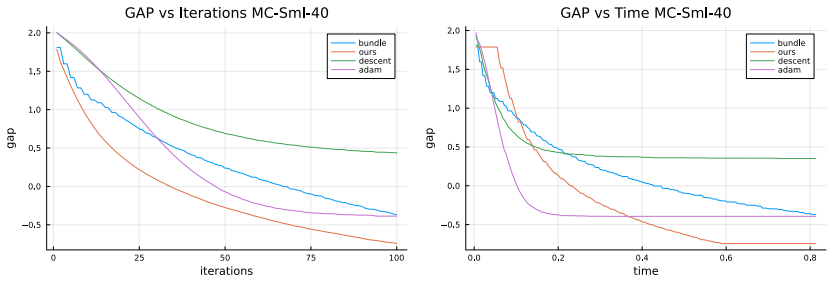

Figure 5.3: Comparison plots of the GAP (in logarithmic scale) during the execution of our trained neural network and different baselines in terms of iterations (in the left) and in terms of time (in the right) for MC-SML-40 dataset.

From Table 5.2 we can see that Bundle Network outperforms other techniques in terms of GAP for a fixed number of iterations in different datasets. Even in the dataset MC-CANADN where it does not outperform classic Bundle methods (in terms of GAP), it performs similarly to the Bundle method and lies between them and Adam/Descent methods.

The model learning only η is less advantageous in the first iterations, but provides performances near to the optimal η-strategy found by a grid search while considering more iterations.

By Figures 5.3-5.6, we can see that Bundle Network needs an initialization time that slows down the approach. These times can also depend on passages of CPU vectors to the GPU and vice versa, and possibly they can be improved. For MC-CANADN the performance are a bit poorer than the exact Bundle method, but still not so fair. Nevertheless, Bundle networks generalizes well to a larger number of iterations than those used during training (10) providing the best gaps for almost every dataset and for a larger number of iterations the initialization times become less relevant.

From Table 5.2 we can also see that the constant $\eta-$strategy provides almost every time the best performance, compared to the other *deterministic* $\eta-$strategies. This is a well-known characteristic of the Bundle method, as 100 iterations is a *relatively* small number of iterations for an Aggregated Bundle method, making η tuning less performative than a constant strategy obtained after a grid-search. This is because the Bundle method operates in distinct phases: initially, its goal is to identify promising directions for improvement, and later it focuses on collecting high-quality subgradients near the optimum to certify optimality This raises an interesting question about using multiple models to capture this Bundle behavior, which is beyond the scope of this chapter.

5.2.3 Ablation Study

In this subsection, we compare the Bundle network trained with and without the sampling strategy for the hidden representation discussed earlier. We consider also the two alternatives to provide an element of the simplex: softmax and sparsemax. In the case

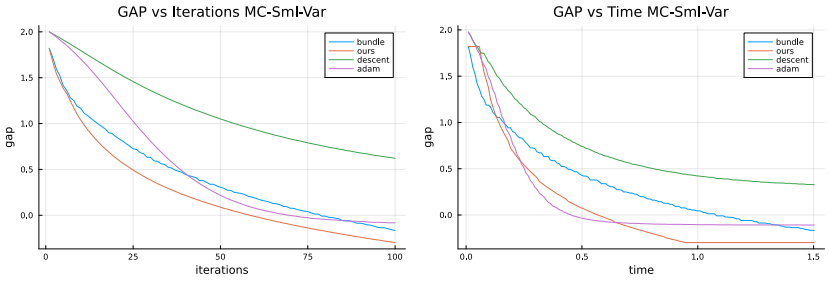

Figure 5.4: Comparison plots of the GAP (in logarithmic scale) during the execution of our trained neural network and different baselines in terms of iterations (in the left) and in terms of time (in the right) for MC-SML-VAR dataset.

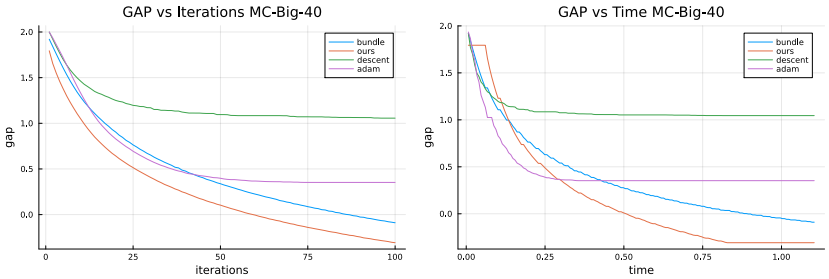

Figure 5.5: Comparison plots of the GAP (in logarithmic scale) during the execution of our trained neural network and different baselines in terms of iterations (in the left) and in terms of time (in the right) for MC-BIG-40 dataset.

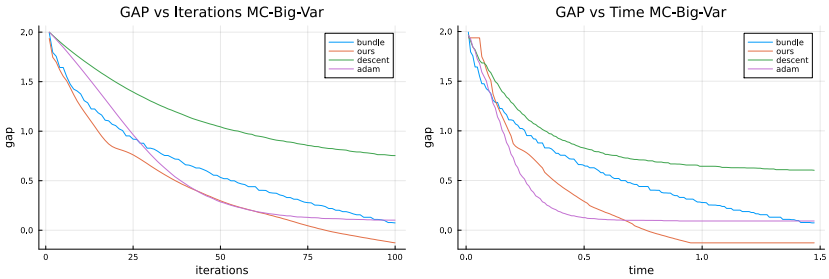

Figure 5.6: Comparison plots of the GAP (in logarithmic scale) during the execution of our trained neural network and different baselines in terms of iterations (in the left) and in terms of time (in the right) for MC-BIG-VAR dataset.

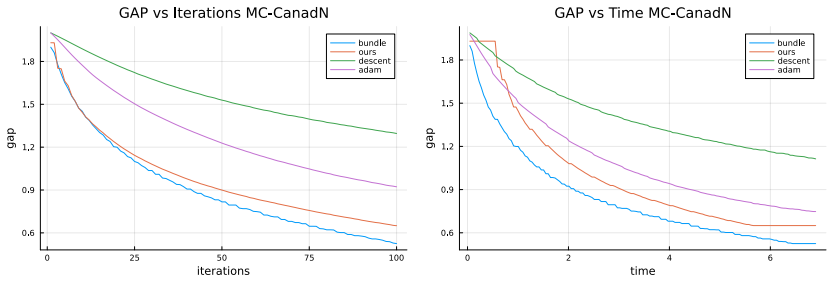

Figure 5.7: Comparison plots of the GAP (in logarithmic scale) during the execution of our trained neural network and different baselines in terms of iterations (in the left) and in terms of time (in the right) for MC-CANADN dataset.

in which we do not use the sampling mechanism, the hidden representations for the queries, the key, and the step size are not predicted by sampling them through a Gaussian distribution. Indeed, we learn the mean and the variance. Instead, it is predicted directly by the encoder.

In the test phase, the hidden representation is not sampled for either approach. Instead, we take the mean directly when the model predicts both the mean and the variance.

Table 5.3: Comparison of softmax and sparsemax, with or without using the sample strategy, or using the sample strategy only for t on MC-SML-40

Activation	Sample		10 iter.		25 iter.		50 iter.		100 iter.	
	t	q k	GAP	time	GAP	time	GAP	time	GAP	time
softmax			**9.20**	0.105	**1.75**	0.185	**0.53**	0.321	**0.17**	0.592
softmax	x		9.78	0.104	1.85	0.187	0.56	0.321	**0.17**	0.592
softmax	x	x	10.41	0.105	1.99	0.185	0.60	0.318	0.18	0.588
sparsemax			58.3	0.125	57.93	0.214	57.47	0.361	57.29	0.661
sparsemax	x		13.16	0.124	2.61	0.212	0.80	0.359	0.24	0.655
sparsemax	x	x	10.51	0.126	2.07	0.214	0.60	0.363	0.18	0.662

Tables 5.3-5.7 shows that, the sampling mechanism seems to be advantageous while using sparsemax, meanwhile performances suggest to do not use it when softmax is considered, even if the performance difference is smaller while using softmax. One possible advantage of combining sampling with sparsemax is that this approach may help the model escape regions where sparsemax has a zero derivative.

5.2.4 Testing on another dataset

Table 5.8 shows several interesting generalization properties of Bundle network. For MC-SML-40 training the model on MC-SML-VAR provides better performances than the model trained on MC-SML-40 itself. There are no other cases in which train the

Table 5.4: Comparison of softmax and sparsemax, with or without using the sample strategy, or using the sample strategy only for t on MC-BIG-40

Activation	Sample		10 iter.		25 iter.		50 iter.		100 iter.	
	t	q k	GAP	time	GAP	time	GAP	time	GAP	time
softmax			**12.70**	0.130	3.41	0.243	1.26	0.436	0.47	0.832
softmax	x		12.85	0.131	**3.34**	0.244	**1.22**	0.436	**0.45**	0.831
softmax	x	x	12.88	0.131	3.36	0.244	1.26	0.436	0.48	0.834
sparsemax			38.65	0.150	24.00	0.270	10.34	0.475	3.39	0.898
sparsemax	x		39.40	0.150	32.37	0.270	17.76	0.474	1.58	0.896
sparsemax	x	x	13.27	0.150	3.50	0.272	1.40	0.479	0.51	0.905

Table 5.5: Comparison of softmax and sparsemax, with or without using the sample strategy or using the sample strategy only for t on MC-SML-VAR

Activation	Sample			10 iter.	25 iter.		50 iter.		100 iter.	
	t	q k	GAP	time	GAP	time	GAP	time	GAP	time
softmax			**12.60**	0.138	3.28	0.269	**1.24**	0.496	**0.50**	0.952
softmax	x		13.03	0.137	**3.27**	0.268	1.26	0.492	0.51	0.948
softmax	x	x	14.68	0.139	3.94	0.269	1.55	0.494	0.62	0.957
sparsemax			30.23	0.157	15.96	0.293	12.45	0.527	11.05	1.006
sparsemax	x		56.83	0.160	56.03	0.300	56.00	0.538	56.00	1.033
sparsemax	x	x	14.02	0.158	3.80	0.298	1.56	0.531	0.64	1.007

model on another dataset provides better performance that the model trained on the dataset itself. Anyway there are several cases in which, in particular for 50 and 100 iterations, the model trained on a different dataset performs better than the Bundle method obtained by the grid search.

5.3 Conclusions

In this chapter, we explore two approaches based on unrolling and amortization. The first is a machine learning-based η-strategy that predicts the value of the parameter η at each iteration. We demonstrate how this approach can be differentiated to maximize the objective function in an unsupervised manner. However, the gradient computation in this case is only approximate. As confirmed by the numerical experiments, this method performs less effectively than the second approach, which jointly predicts both the η parameter and the new search direction. By bypassing the need to solve the Dual Master problem at each iteration, this second method leads to a smoother optimization process and yields more informative gradients. Moreover, it exhibits strong generalization capabilities, maintaining robust performance even beyond the training horizon and across different datasets for the same problem.

Table 5.6: Comparison of softmax and sparsemax, with or without using the sample strategy or using the sample strategy only for t on MC-BIG-VAR

Activation	Sample		10 iter.		25 iter.		50 iter.		100 iter.	
	t	q k	GAP	time	GAP	time	GAP	time	GAP	time
softmax			20.10	0.139	5.11	0.271	2.01	0.495	0.70	0.953
softmax	x		16.79	0.141	4.78	0.273	1.68	0.499	0.63	0.956
softmax	x	x	**16.66**	0.139	4.63	0.270	1.80	0.495	0.67	0.955
sparsemax			42.84	0.160	32.39	0.301	29.68	0.541	23.25	1.031
sparsemax	x		30.20	0.162	12.33	0.305	6.25	0.542	3.78	1.033
sparsemax	x	x	18.94	0.162	**4.24**	0.305	**1.47**	0.548	**0.61**	1.045

Table 5.7: Comparison of softmax and sparsemax, with or without using the sample strategy, or using the sample strategy only for t on MC-CANADN

Activation	Sample		10 iter.		25 iter.		50 iter.		100 iter.	
	t	q k	GAP	time	GAP	time	GAP	time	GAP	time
softmax			**29.73**	0.491	14.34	1.23	8.05	2.474	**4.48**	4.98
softmax	x		29.74	0.462	14.22	1.171	8.0	2.358	4.41	4.75
softmax	x	x	31.97	0.498	17.07	1.259	10.04	2.533	5.77	5.1
sparsemax			41.21	0.474	21.93	1.198	15.58	2.413	11.94	4.857
sparsemax	x		29.89	0.493	**13.23**	1.248	**7.51**	2.51	4.58	5.055
sparsemax	x	x	58.91	0.463	55.04	1.156	53.94	2.313	53.64	4.65

Dataset		GAP %			
Test	Train	10 iter.	25 iter.	50 iter.	100 iter.
MC-SML-40	MC-SML-40	9.2	1.75	0.53	**0.17**
	MC-SML-VAR	**8.02**	**1.69**	**0.51**	**0.17**
	MC-BIG-40	18.02	4.14	1.49	0.47
	MC-BIG-VAR	10.77	2.26	0.7	0.2
	MC-CANADN	13.5	3.18	0.89	0.25
best baseline		11.96	4.16	1.30	0.31
MC-SML-VAR	MC-SML-40	19.87	5.72	2.32	1.0
	MC-SML-VAR	**12.6**	**3.28**	**1.24**	**0.5**
	MC-BIG-40	33.83	14.03	7.54	3.9
	MC-BIG-VAR	16.21	3.83	1.39	0.52
	MC-CANADN	16.4	4.74	1.56	0.53
best baseline		12.79	4.33	1.66	0.58
MC-BIG-40	MC-SML-40	35.09	5.78	1.63	0.62
	MC-SML-VAR	56.68	25.62	8.88	3.06
	MC-BIG-40	**12.7**	**3.41**	**1.26**	**0.47**
	MC-BIG-VAR	37.57	10.74	4.89	1.49
	MC-CANADN	62.79	35.18	16.83	4.82
best baseline		19.34	5.64	2.08	0.78
MC-BIG-VAR	MC-SML-40	21.50	5.48	2.07	0.85
	MC-SML-VAR	21.03	7.64	2.67	0.95
	MC-BIG-40	28.39	11.15	5.87	2.94
	MC-BIG-VAR	20.10	**5.11**	**2.01**	**0.70**
	MC-CANADN	26.51	10.07	4.58	1.44
best baseline		**17.98**	6.34	2.66	0.93
MC-CANADN	MC-SML-40	44.29	24.95	15.26	9.0
	MC-SML-VAR	34.39	17.07	9.34	5.02
	MC-BIG-40	56.38	35.08	23.79	15.33
	MC-BIG-VAR	33.66	16.16	8.84	4.91
	MC-CANADN	29.73	14.34	8.05	4.48
best baseline		**26.74**	**11.93**	**6.17**	**3.17**

Table 5.8: Comparison of the network with and without the sampling strategy on different datasets. Here, we also consider the generalization properties of networks trained on one dataset and tested on another dataset. The rows *best baseline* shows the mean value (in the test set) of the Bundle variant associated with the lower gap.

Chapter 6

Conclusions and Future Research Directions

This chapter summarizes the main results achieved in this thesis and presents some directions for future research. Section 6.1 is dedicated to the overall conclusions drawn from our research, whereas Section 6.2 outlines research avenues that have emerged during this scientific exploration.

6.1 Conclusions

In this thesis, we focused on the domain of machine learning for optimization. The thesis proposed a novel integration of machine learning techniques with classical optimization frameworks, specifically targeting the prediction of Lagrangian multipliers. The first contribution, discussed in Chapter 4, focuses on the prediction of Lagrangian multipliers. This requires encoding an optimization instance in a form suitable for input to a machine learning model.

Accordingly, we employ the bipartite graph representation, originally introduced by Gasse et al. [90] and described in detail in Section 4.1.2. This representation allows for encoding various optimization problems and handling instances of different sizes.

Starting from a simple feature representation for each node, we use a Graph Neural Network (GNN) encoder to iteratively refine node embeddings through message-passing. Final embeddings are retained only for the dualized constraints. Each embedding parametrizes the mean and the variance of a Gaussian distribution from which we sample a latent representation of each dualized constraint. These representations are fed into a decoder, composed of a parallel Multi-Layer Perceptron, each predicting a scalar corresponding to a predicted Lagrangian multiplier for each dualized constraint.

This model can be cast as an Energy-Based Model, as shown in Chapter 4. Numerical experiments on Multi-Commodity Network Design and Generalized Assignment problems demonstrate that our model delivers tight bounds efficiently and generates Lagrangian multipliers that can be used to initialize exact solvers for the Lagrangian

Dual problem, such as the Bundle method. Thus, our approach belongs to the family of amortized optimization approaches.

Next, we investigate ways to incorporate learning into the Bundle method itself. While initialization is important, it is not the only factor affecting the performance of the Bundle method. In Chapter 5, we start by exploring a complementary direction: learning the regularization parameter in the Bundle method.

By rewriting the Bundle method in a sub-differentiable form, we enable gradient approximation via unrolling techniques and dynamically learn the parameter. However, this approach still relies on solving a quadratic problem at each iteration to determine a search direction and includes non-differentiable operations for updating the stabilization point.

To address these limitations, Chapter 5 introduces a smoother variant of this method, called *Bundle Network*. This variant replaces *hard updates* for the stabilization point with *soft updates* and uses a learning model, based on recurrent and attention layers, to predict the search direction at each iteration. This results in a smoother optimization process, which achieves better performance in numerical experiments than the one that only tunes the regularization parameter. In particular, it lowers the gaps in many datasets for the Multi-Commodity Network Design problem. Even when the predicted solutions provide a worse gap than the ones obtained with state-of-the-art methods, their quality is still close. Bundle Network removes the necessity of performing a grid search to choose the best initialization for the regularization parameter and to determine the best strategy to handle increments or decrements of that parameter, thus providing a significant advantage.

Moreover, this method can be used for larger instances for which the convolution on the bipartite graph representation becomes intractable. The second approach from Chapter 5 can be viewed as a learned optimizer. We demonstrate that the approach has good generalization capabilities: when trained on datasets composed only of small instances, the model still performs well on larger ones.

6.2 Future Directions

Although Chapters 4 and 5 explore different approaches, they are also complementary: advances in one area may benefit the other. For instance, the method proposed in Chapter 4 could serve as the initial Lagrangian multipliervector for the second approach from Chapter 5, thereby facilitating joint training of the models. This section outlines several open research directions that emerged during my academic journey.

Section 6.2.1 discusses the representations considered as inputs to the machine learning model. In Chapter 4 and Chapter 5, we use radically different ways to represent information related to an optimization instance, each of which offers distinct advantages and limitations. Here, we discuss the limitations of both approaches and explore possible strategies for their improvement. Section 6.2.2 stems from the explorations carried out in Chapter 5 and more particularly the second approach. Here we discuss some problems that emerged during this research and further extensions opened up by this exploration.

These subsections should not be regarded as comprehensive research plans but as preliminary research notes.

6.2.1 Instance Representation

In Chapters 4 and 5, we explore different strategies to represent an optimization instance such that it can serve as input to a neural network. Throughout, we focus on representations that support datasets of varying sizes and are adaptable to different types of optimization problems.

The bipartite graph representation considered in Chapter 4 has proven to be effective for modeling optimization instances of diverse sizes and providing them as input to neural networks. In Chapter 5, we instead take advantage of inserting our prediction inside an iterative resolution process, where, at each step, *hand-crafted* feature representations are computed for each newly inserted component, exploiting task-specific knowledge.

Problematic of the Bipartite Graph Representation

The bipartite graph representation automates feature extraction by relying on simple characteristics defining the variables and constraints of the optimization problem.

However, it also presents significant limitations. In particular, because the graph contains one node per variable and one node per constraint, message-passing operations become computationally infeasible for large-scale instances.

Solutions for General Graphs The Graph Neural Network (GNN) community has proposed several strategies to address scalability issues. These methods include subgraph-based training techniques [89], dropout mechanisms [195], convergence-guaranteed sampling methods [254], as well as dimensionality reduction via vector quantization [101, 68] and Sobolev-norm-based approaches to sparse convolutions [97].

Architectural innovations have also been introduced, such as filters with efficient precomputation [217], and sparse attention mechanisms with L_0 regularization in GATs [273]. Other research directions explore transfer learning for GNNs [141], and the impact of sparse matrix storage formats [211].

Domain-Specific Strategies for Optimization Instances General-purpose GNN techniques may not be optimal for our setting. Our application involves bipartite graphs where predictions are required only on a subset of nodes. Since the graph encodes an optimization instance, we may leverage domain-specific structures to compress the representation while preserving the quality of computed features.

Many optimization problems exhibit a block diagonal structure, at least partially. For instance, in Lagrangian relaxation, families of constraints often involve only subsets of variables, interconnected by other families of constraints. This observation motivates a two-level strategy: first, performing convolutions within these smaller blocks, and then aggregating information across blocks via a higher-level graph.

However, exploiting sparsity at the adjacency matrix level could be insufficient. The real challenge lies in identifying structures within the bipartite graph that allow us

to *condense* information in some part of the graph, useless for direct predictions, for example, inspired by Hierarchical GNN [231].

Thus, an important question is how to define and exploit such structures to enable effective message-passing on substantially smaller graphs.

Dependency on the Continuous Relaxation Another limitation of the approach presented in Chapter 4 is its dependence on Continuous relaxation (CR).

Our method enhances the bounds obtained from CR by feeding solution information into a neural network. However, solving the CR itself may be computationally prohibitive or may yield the same bounds as a Lagrangian relaxation. Anyway, our experiments suggest that features related only to the instance are insufficient for strong predictive performance.

Solving the Lagrangian subproblem for a single vector of Lagrangian multipliers is often faster than the resolution of the CR. However, the CR is just one way of deriving Lagrangian multipliers. A natural question thus arises: how can we enhance feature representations by leveraging solution information obtained through alternative approaches?

This is a non-trivial challenge, as solving this task could become the task itself. In the sense that the approach presented in Chapter 4 already provides a way to predict a Lagrangian multiplier vector. Nonetheless, this prompts the idea of considering some kind of generative process that utilizes information gained from the Lagrangian subproblem to enhance feature representations. Yet, this also establishes a clear connection with the second approach discussed in Chapter 5, which further highlights the significance of how we compute the features for this methodology.

Challenges with Hand-Crafted Features

Automating feature extraction is highly desirable. First, it would eliminate the need for human expertise, and second, it could lead to representations better tailored and more efficient for particular tasks.

To illustrate a key limitation of manually designed feature logic, we examine a scenario that initially motivated our investigation into feature quality. In particular, we suggest modifying our approach to add new components to the bundle only when necessary.

For instance, let us consider the case where a newly visited point lies on the same line as a previously visited one, as illustrated in Figure 6.1. This case is interesting because it highlights how two points can share identical gradient information. This means that they have the same component in the quadratic part of the Dual Master problem 1.20, but one point will have a higher coefficient in the linear part, which results in higher linearization error. This implies that in the optimal solution of the Dual Master problem, the associated coordinate will always be equal to zero. Therefore, it is sufficient to keep only the point with the lower linearization error, since the solution of the Dual Master problem, and consequently, the search direction, will remain unchanged.

We attempted to integrate this idea, but the resulting framework of the Bundle Network provides poorer performance. This may be attributed to the feature encoding

being too weak to achieve previous performance levels. This underscores the importance of a richer and dynamically updated feature representation.

6.2.2 Bundle Network Extensions

In this section, we discuss modifications to the Bundle Network for different purposes. First, we present some general-purpose extensions inspired by the *classical* Proximal Bundle method. These may require a more appropriate representation of the visited points, as preliminary experiments suggest that the current representation is insufficient. We will then conclude by examining potential achievements aimed at extending our approach to meta-learning. This may involve introducing modifications inspired by Bundle methods for non-convex optimization problems.

Extension to Constrained Problems

Our Bundle Network implementation is limited to unconstrained concave or convex optimization problems.

From the perspective of Lagrangian relaxation, this restriction corresponds to dualizing only equality constraints. However, many practical problems involve inequalities, and extending the Bundle Network to handle them is an interesting next step. Concretely, if we introduce only inequality constraints, the Lagrangian Dual inherits non-negativity requirements on its multipliers. In other words, we move from solving the Dual Master problem in Equation (1.20) to the non-negative multiplier variant in Equation (1.19) (as presented in Chapter 1).

To maintain the *smooth* multiplier updates that motivate the Bundle Network, we can assure non-negativity simply by enforcing the new trial point to be non-negative component-wise. In practice, one can achieve this by applying a non-negative activation function to each tentative multiplier update. This mean that line 10 of Algorithm 3 become:

$$\boldsymbol{\pi}_{t+1} \leftarrow \sigma(\bar{\boldsymbol{\pi}}_t + \eta_t \boldsymbol{w}^{(t)})$$

where σ is an activation function, assuming non-negative values, as `ReLU` or `Softplus`, applied component-wise to the vector $\bar{\boldsymbol{\pi}}_t + \eta_t \boldsymbol{w}^{(t)}$.

When the feasible set involves more complex constraints, as general polyhedra or nonlinear regions, a direct extension is less straightforward. One promising idea is to incorporate a projection operator onto the feasible multiplier set at each iteration, thus ensuring validity of the dual variables without sacrificing smoothness. We leave the detailed development and analysis of such projections to future work.

Removing Outdated Components

In classical Bundle methods, some techniques are considered to avoid adding redundant points and to regularly remove outdated points. Components that no longer influence the search direction after many consecutive iterations are removed. Even the bundle size could be fixed, forcing the removal of the more useless components. This prevents the bundle from growing uncontrollably. Points that describe the function far from the current stabilization point are typically discarded after being inactive for many iterations.

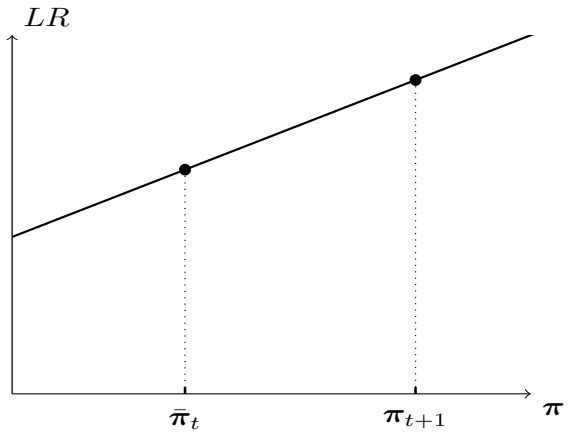

Figure 6.1: This figure represents a case in which we do not want to add a new component to the bundle, as $\bar{\boldsymbol{\pi}}_t$ and $\boldsymbol{\pi}_{t+1}$ share the same sub-gradient. Instead, we would like to update the information by changing the stabilization point and updating the function value and the linearization error with the information in $\boldsymbol{\pi}_{t+1}$, which should also become the new stabilization point.

By dynamically maintaining a compact bundle with an enhanced feature representation, we could further reduce the prediction space and improve scalability. Let us observe that this is easy to implement. At inference time, it is possible to simply remove the associated components from the bundle. At training time, all the information should be maintained to properly perform the back-propagation. Anyway, this is easy to implement as we can use an adaptive mask to choose which component to consider and which to remove.

Bundle Network for non-convex Optimization and Meta Learning

There is a growing interest in applying the Bundle method for optimization problems arising from the machine learning community [150]. Variants of Proximal Bundle methods [139] are used to optimize complex, non-differentiable loss functions often encountered in machine learning [196].

Recent research on Bundle methods for non-convex problems [63, 62], opens the possibility to use the Bundle variants for machine learning problems. Bundle methods have been applied to improve the training of SVMs [56], especially for large-scale problems [126, 243]. Extensions of Bundle methods have also been proposed for multiclass SVMs, where Bundle methods can handle a larger number of constraints efficiently [78].

Bundle Network for non-convex Optimization A *natural* question that arises is: how to adapt the Bundle Network for non-convex functions? In this context, several modifications should be considered. First, since the loss function may be non-convex,

it would be necessary to implement specific techniques to prevent the model from becoming trapped in a local optimum that is not globally optimal.

Figure 6.2 illustrates a simple example of a non-smooth and non-convex objective function. We start from the stabilization point shown in green at iteration t, and we assume we already have the information related to the red line in the bundle. Moving rightward in the following iteration, two different settings can be considered. If we find the new trial point in blue, we *escape* from the *bad* local optimum, and the information provided by the red line becomes obsolete and should be discarded. When the blue point becomes the new stabilization point, the linearization error associated with the point in the red line will be negative, whereas, for convex functions, linearization errors are always non-negative. This negative error provides a simple criterion for identifying outdated information, related to other locally concave parts of the function, possibly leading to a worse local optimum.

If the new trial point is the red one, since its objective value is lower than that of the green point, we do not update the stabilization point immediately. At this stage, we cannot determine whether the red point leads to a better local optimum. In this case, additional exploration techniques might be necessary to avoid becoming trapped in a *bad* optimum.

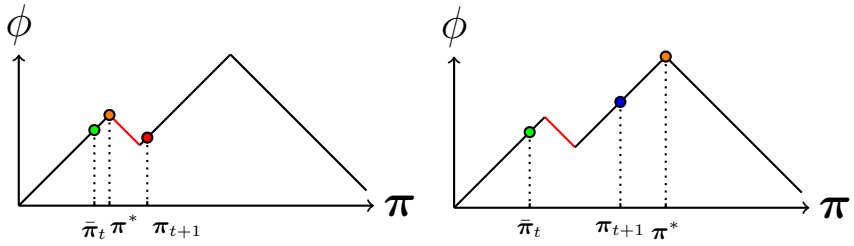

Figure 6.2: Example of a non-convex function with two possibilities in which the Bundle could escape or not through a local optimum. The green point is the current stabilization point. If at the next iteration we find the blue point and we change the stabilization point, we have to suppress the information related to the green point to converge to the global optimum. If we encounter the red point, the stabilization point remains unchanged, and it becomes necessary to implement exploration techniques to escape from the local optimum.

Bundle Network for Meta Learning A clear and compelling application in this context is training a neural network using the Bundle Network framework. In other words, we want to solve, using Bundle Networks, the optimization problem of selecting the best parameter configuration for a neural network that minimizes a specific loss function. In this case, instead of maximizing the Lagrangian subproblem value, we minimize the mean loss function value in a given dataset. The solution will not be Lagrangian multiplier vectors, but the parameter configuration of the neural network. Our meta-learning task, then, is to determine the parameters of the optimizer in such a way that it provides effective updates for solving this learning problem.

Let us observe that, in general, learning problems do not require additional constraints. This highlights a complementary research direction to the one already discussed: here, we aim to apply our approach to more elaborate objective functions without introducing additional constraints to the feasible region.

Making Predictions in the *Dual* Space Andrychowicz et al. [10] directly make predictions in the solution space, eliminating the need for explicit feature extraction. This is made possible by considering the input component-wise in the space Π and using the obtained subgradient information. This approach can be problematic in some cases. For the Lagrangian relaxation, the input can be extremely large, as it depends on the number of dualized constraints. Moreover, in Meta-Learning settings, the input would have the same size as the number of parameters in the network being trained, which makes it impractical for large architectures.

An advantage of our approach is that we make predictions in the *dual* space rather than directly in the parameter space. We assert that, denoting by $\boldsymbol{d}^{(t)}$ the solution of the Master problem in the *classic* Bundle method at iteration t, the size of $\boldsymbol{d}^{(t)}$ is the same as that of the solution $\boldsymbol{\pi} \in \Pi$. The vector $\boldsymbol{d}^{(t)}$ is linked to the solution of the Dual Master problem by the formula $\boldsymbol{d}^{(t)} = \eta_t \boldsymbol{w}^{(t)} = \eta_t \sum_{i=1}^{|\beta_t|} \theta_i^{(t)} \boldsymbol{g}_i$ where $\boldsymbol{\theta}^{(t)} \in [0,1]^{|\beta_t|}$ is the solution of the Dual Master problem. The dimension of $|\beta_t|$ is generally significantly smaller than the dimension of Π. Indeed, $|\beta_t|$ can be bounded by the number of training epochs in a typical implementation. Furthermore, we do not need to call our model for each component in β_t since we store the information of previously visited components.

This gives us reason to believe that our approach can be significantly beneficial for neural network training, as it may enable better scalability for larger networks. This could still require exploiting some block structure to encode and share information between the different parameters.

Given that the points in the space Π correspond to parameter configurations of a neural network, ad-hoc techniques can be developed, for instance, to handle parameters and gradients specific to different layers. Andrychowicz et al. [10] used a parallel model on all the components; meanwhile, we need to carefully choose the feature representation to obtain good performance. A useful task, therefore, will provide valuable information without operating directly on the high-dimensional parameter space.

This could also prompt further modifications to the Bundle framework, potentially bringing it closer to the Disaggregated Bundle method, introduced in Chapter 1. Anyway, this connection is not immediate. On one hand, we would like to be able to consider different gradient aggregations to propose ad-hoc updates for the layers in the network. On the other hand, the Disaggregated Bundle is generally related to a decomposable structure with independent functions, which are only related to the choice of a common trial point. In the case of learning problems, we only have a single function: the value of the loss in the learning problem.

Bibliography

[1] A. Abbas and P. Swoboda. DOGE-Train: Discrete Optimization on GPU with End-to-end Training, May 2022.

[2] A. Abbas and P. Swoboda. Fastdog: Fast discrete optimization on gpu. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 439–449, June 2022.

[3] M. R. Akhavan Kazemzadeh, T. Bektaş, T. G. Crainic, A. Frangioni, B. Gendron, and E. Gorgone. Node-based lagrangian relaxations for multicommodity capacitated fixed-charge network design. *Discrete Applied Mathematics*, 308:255–275, 2022. Combinatorial Optimization ISCO 2018.

[4] A. M. Alvarez, Q. Louveaux, and L. Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. Orbi, 2014.

[5] A. M. Alvarez, Q. Louveaux, and L. Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[6] S. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, 3:299–307, 1967.

[7] B. Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *CoRR*, 2022.

[8] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 06–11 Aug 2017.

[9] E. D. Andersen and K. D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm*, pages 197–232. Springer US, Boston, MA, 2000.

[10] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[11] J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

[13] B. Babaki and S. D. Jena. COIL: A deep architecture for column generation. Technical report, Cirrelt, 2022.

[14] V. Balachandran. An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks. *Operations Research*, 24(4):742–759, Aug. 1976.

[15] R. Baltean-Lugojan, P. Bonami, R. Misener, and A. Tramontani. Selecting cutting planes for quadratic semidefinite outer-approximation via trained neural networks. In *optimization-online*, 2018.

[16] D. Bank, N. Koenigstein, and R. Giryes. *Autoencoders*, pages 353–374. Springer International Publishing, Cham, 2023.

[17] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, May 2000.

[18] S. Basso, A. Ceselli, and A. Tettamanzi. Random sampling and machine learning to understand good decompositions. *Annals of Operations Research*, 284(2):501–526, Jan. 2020.

[19] R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*, volume 45. Springer Science & Business Media, 2008.

[20] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[21] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[22] D. Belanger and A. McCallum. Structured prediction energy networks. *ArXiv*, abs/1511.06350, 2015.

[23] D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 429–439. PMLR, 2017.

[24] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, Dec 1962.

[25] S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for anns. *Neural Processing Letters*, 2:26–30, 07 1995.

[26] Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 969 vol.2–, 1991.

[27] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *CoRR*, abs/1811.06128, 2018.

[28] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[29] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J. Vert, and F. R. Bach. Learning with differentiable perturbed optimizers. *CoRR*, abs/2002.08676, 2020.

[30] T. Berthold, M. Francobaldi, and G. Hendel. Learning to use local cuts. *arXiv preprint arXiv:2206.11618*, 2022.

[31] C. Bertocchi, E. Chouzenoux, M.-C. Corbineau, J.-C. Pesquet, and M. Prato. Deep unfolding of a proximal interior point method for image restoration, 2020.

[32] D. P. Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[33] M. Blondel, A. F. Martins, and V. Niculae. Learning with fenchel-young losses. *Journal of Machine Learning Research*, 21(35):1–69, 2020.

[34] S. Borozan, S. Giannelos, P. Falugi, A. Moreira, and G. Strbac. Machine learning-enhanced benders decomposition approach for the multi-stage stochastic transmission expansion planning problem. *Electric Power Systems Research*, 237:110985, 2024.

[35] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[36] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of Bundle and Classical Column Generation. *Mathematical Programming*, 113(2):299–344, June 2008.

[37] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[38] R. E. Burkard, S. E. Karisch, and F. Rendl. Qaplib – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, Jun 1997.

[39] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks, 2022.

[40] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[41] D. G. Cattrysse and L. N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European journal of operational research*, 60(3):260–272, 1992.

[42] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, M. Schuster, N. Shazeer, N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, Z. Chen, Y. Wu, and M. Hughes. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[43] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[44] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[45] T. Chen, W. Zhang, Z. Jingyang, S. Chang, S. Liu, L. Amini, and Z. Wang. Training stronger baselines for learning to optimize. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7332–7343. Curran Associates, Inc., 2020.

[46] Z. Chen, J. Liu, X. Wang, J. Lu, and W. Yin. On representing linear programs by graph neural networks, 2023.

[47] C. Chi, A. Aboussalah, E. Khalil, J. Wang, and Z. Sherkat-Masoumi. A deep reinforcement learning framework for column generation. *Advances in Neural Information Processing Systems*, 35:9633–9644, 2022.

[48] J. M. Child. The manuscripts of leibniz on his discovery of the differential calculus. part ii (continued). *The Monist*, 27(3):411–454, 1917.

[49] A. Chmiela, E. B. Khalil, A. Gleixner, A. Lodi, and S. Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

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

[51] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[52] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[53] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Society for Industrial and Applied Mathematics, 1990.

[54] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer, New York, 2014.

[55] M.-C. Corbineau, C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet. Learned Image Deblurring by Unfolding a Proximal Interior Point Algorithm. In *ICIP 2019 - 26th IEEE International Conference on Image Processing*, Taipei, Taiwan, Sept. 2019. IEEE.

[56] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[57] T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1):73–99, 2001.

[58] B. C. Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[59] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[60] G. Dalle, L. Baty, L. Bouvier, and A. Parmentier. Learning with combinatorial optimization layers: a probabilistic approach. *arXiv preprint arXiv:2207.13513*, 2022.

[61] G. B. Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[62] W. de Oliveira and M. Solodov. A doubly stabilized bundle method for nonsmooth convex optimization. *Mathematical programming*, 156(1):125–159, 2016.

[63] W. L. de Oliveira and C. A. Sagastizábal. Level bundle methods for oracles with on-demand accuracy. *Optimization Methods and Software*, 29:1180 – 1209, 2014.

[64] F. Demelas, J. Le Roux, M. Lacroix, and A. Parmentier. Predicting lagrangian multipliers for mixed integer linear programs. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 10368–10384. PMLR, 21–27 Jul 2024.

[65] J. Desrosiers and M. E. Lübbecke. Branch-price-and-cut algorithms. *Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Chichester*, pages 109–131, 2011.

[66] S. S. Dey and M. Molinaro. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170(1):237–266, July 2018.

[67] J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. Accelerating primal solution findings for mixed integer programs based on solution prediction, 2019.

[68] M. Ding, K. Kong, J. Li, C. Zhu, J. P. Dickerson, F. Huang, and T. Goldstein. VQ-GNN: A universal framework to scale up graph neural networks using vector quantization. *CoRR*, abs/2110.14363, 2021.

[69] J. Djolonga and A. Krause. Differentiable learning of submodular models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[70] J. Domke. Implicit differentiation by perturbation. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[71] J. Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326. PMLR, 2012.

[72] P. L. Donti, B. Amos, and J. Z. Kolter. Task-based end-to-end model learning. *CoRR*, abs/1703.04529, 2017.

[73] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

[74] Y. Du, S. Li, J. Tenenbaum, and I. Mordatch. Learning iterative reasoning through energy minimization. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5570–5582. PMLR, 17–23 Jul 2022.

[75] Y. Du, J. Mao, and J. B. Tenenbaum. Learning iterative reasoning through energy diffusion, 2024.

[76] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. A comprehensive survey and performance analysis of activation functions in deep learning. *CoRR*, abs/2109.14545, 2021.

[77] Y. Eryoldaş and A. Durmusoglu. A literature survey on instance specific algorithm configuration methods. In *Annual International Conference on Industrial Engineering and Operations Management*, 03 2021.

[78] T. Finley and T. Joachims. Training structural svms when exact inference is intractable. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 304–311, New York, NY, USA, 2008. Association for Computing Machinery.

[79] F. Fioretto, T. W. Mak, and P. Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 630–637, 2020.

[80] R. A. Fisher. Theory of statistical estimation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 22(5):700–725, 1925.

[81] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum separable assignment problems. *Mathematics of Operations Research*, 36(3):416–431, 2011.

[82] A. Frangioni. Solving semidefinite quadratic problems within nonsmooth optimization algorithms. *Computers & Operations Research*, 23(11):1099–1118, 1996.

[83] A. Frangioni. *Dual-ascent methods and multicommodity flow problems*. Università degli studi di Pisa. Dipartimento di informatica, 1997.

[84] A. Frangioni. Generalized bundle methods. *SIAM Journal on Optimization*, 13(1):117–156, 2002.

[85] A. Frangioni and B. Gendron. A stabilized structured dantzig–wolfe decomposition method. *Mathematical Programming*, 140:45–76, 2013.

[86] A. Frangioni, B. Gendron, and E. Gorgone. On the computational efficiency of subgradient methods: a case study with lagrangian bounds. *Mathematical Programming Computation*, 9(4):573–604, Dec 2017.

[87] A. Frangioni, B. Gendron, and E. Gorgone. Dynamic smoothness parameter for fast gradient methods. *Optimization Letters*, 12(1):43–53, Jan 2018.

[88] A. Frangioni, N. Iardella, and R. Durbano Lobato. SMS++, 2023.

[89] H. Gao, Z. Wang, and S. Ji. Large-scale learnable graph convolutional networks. *CoRR*, abs/1808.03965, 2018.

[90] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[91] B. Gendron, T. G. Crainic, and A. Frangioni. *Multicommodity Capacitated Network Design*, pages 1–19. Springer US, Boston, MA, 1999.

[92] Z. Geng, X. Li, J. Wang, X. Li, Y. Zhang, and F. Wu. A deep instance generative framework for milp solvers under limited data availability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 26025–26047. Curran Associates, Inc., 2023.

[93] I. Gent, L. Kotthoff, I. Miguel, and P. Nightingale. Machine learning for constraint solver design–a case study for the alldifferent constraint. *arXiv preprint arXiv:1008.4326*, 2010.

[94] A. M. Geoffrion. Lagrangean relaxation for integer programming. In M. L. Balinski, editor, *Approaches to Integer Programming*, pages 82–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 1974.

[95] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[96] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[97] J. H. Giraldo, S. Javed, A. Mahmood, F. D. Malliaros, and T. Bouwmans. Higher-order sparse convolutions in graph neural networks, 2023.

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

[99] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization, 2016.

[100] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[101] R. Gray and D. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998.

[102] A. Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, 389400, 2012.

[103] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, USA, second edition, 2008.

[104] W. Hare, C. Planiden, and C. Sagastizábal. The chain rule for vu-decompositions of nonsmooth functions, 2019.

[105] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[106] T. Hazan, J. Keshet, and D. McAllester. Direct loss minimization for structured prediction. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[107] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[108] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[109] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pages 770–778. IEEE, June 2016.

[110] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[111] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, Dec 1971.

[112] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms II: Advance Theory and Bundle Methods*, volume 305. Springer science & business media, 1996.

[113] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[114] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[115] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks — ICANN 2001*, pages 87–94, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[116] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[117] T. Huang, Y. Ma, Y. Zhou, H. Huang, D. Chen, Z. Gong, and Y. Liu. A review of combinatorial optimization with graph neural networks. In *2019 5th International Conference on Big Data and Information Analytics (BigDIA)*, pages 72–77, 2019.

[118] Z. Huang, K. Wang, F. Liu, H.-L. Zhen, W. Zhang, M. Yuan, J. Hao, Y. Yu, and J. Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353, 2022.

[119] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[120] G. Iommazzo, C. D'Ambrosio, A. Frangioni, and L. Liberti. *Algorithm Configuration Problem*, page 1–8. Springer International Publishing, Sept. 2022.

[121] A. Ivakhnenko, V. Lapa, and P. U. L. I. S. O. E. ENGINEERING. *Cybernetic Predicting Devices*. JPRS 37, 803. Joint Publications Research Service [available from the Clearinghouse for Federal Scientific and Technical Information], 1965.

[122] A. J. Izenman. Introduction to manifold learning. *WIREs Computational Statistics*, 4(5):439–446, 2012.

[123] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax, 2017.

[124] J. Ji, X. Chen, Q. Wang, L. Yu, and P. Li. Learning to learn gradient aggregation by gradient descent. In *IJCAI*, pages 2614–2620, 2019.

[125] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, and T. Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1):12, Feb 2021.

[126] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77:27–59, 10 2009.

[127] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, Nov 1999.

[128] K. Jörnsten and M. Näsberg. A new lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research*, 27(3):313–323, 1986.

[129] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[130] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *CoRR*, abs/1811.11597, 2018.

[131] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016.

[132] E. B. Khalil, B. N. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *International Joint Conference on Artificial Intelligence*, 2017.

[133] E. B. Khalil, C. Morris, and A. Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):10219–10227, Jun. 2022.

[134] D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[135] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[136] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[137] E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

[138] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[139] K. C. Kiwiel. A proximal bundle method with approximate subgradient linearizations. *SIAM Journal on Optimization*, 16(4):1007–1023, 2006.

[140] L. Kong, J. Cui, Y. Zhuang, R. Feng, B. A. Prakash, and C. Zhang. End-to-end stochastic optimization with energy-based model. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11341–11354. Curran Associates, Inc., 2022.

[141] N. Kooverjee, S. James, and T. van Zyl. Investigating transfer learning in graph neural networks. *Electronics*, 11(8), 2022.

[142] J. Kotary, F. Fioretto, P. V. Hentenryck, and B. Wilder. End-to-end constrained optimization learning: A survey. *CoRR*, abs/2103.16378, 2021.

[143] L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. *CoRR*, abs/1210.7959, 2012.

[144] M. Kramer. Autoassociative neural networks. *Computers & Chemical Engineering*, 16(4):313–328, 1992. Neutral network applications in chemical engineering.

[145] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[146] S. Kraul, M. Seizinger, and J. O. Brunner. Machine learning–supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation. *INFORMS Journal on Computing*, 35(3):692–709, 2023.

[147] M. Kruber, M. E. Lübbecke, and A. Parmentier. Learning when to use a decomposition. In D. Salvagnin and M. Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 202–210. Springer, 2017.

[148] A. G. Labassi, D. Chételat, and A. Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in Neural Information Processing Systems*, 35:32000–32010, 2022.

[149] A. H. Land and A. G. Doig. *An automatic method for solving discrete programming problems*. Springer, 2010.

[150] Q. Le, A. Smola, and S. Vishwanathan. Bundle methods for machine learning. *Advances in neural information processing systems*, 20, 2007.

[151] Y. Le Cun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[152] M. Lee, N. Ma, G. Yu, and H. Dai. Accelerating generalized benders decomposition for wireless resource allocation. *IEEE Transactions on Wireless Communications*, 20(2):1233–1247, 2021.

[153] C. Lemaréchal. *Lagrangian Relaxation*, pages 112–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[154] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Mathematical programming*, 69:111–147, 1995.

[155] C. Lemarechal, J.-J. Strodiot, and A. Bihain. On a bundle algorithm for nonsmooth optimization. In *Nonlinear programming 4*, pages 245–282. Elsevier, 1981.

[156] C. Lemaréchal and C. Sagastizabal. Variable metric bundle methods: From conceptual to implementable forms. *Math. Program.*, 76:393–410, 01 1996.

[157] K. Li and J. Malik. Learning to optimize. *ArXiv*, abs/1606.01885, 2016.

[158] M. Li, J. Liu, and W. Yin. Learning to combine quasi-newton methods. *arXiv preprint arXiv:2210.06171*, 2023.

[159] X. Li, Q. Qu, F. Zhu, J. Zeng, M. Yuan, K. Mao, and J. Wang. Learning to reformulate for linear programming, 2022.

[160] H. Liang, S. Wang, H. Li, L. Zhou, X. Zhang, and S. Wang. Bignn: Bipartite graph neural network with attention mechanism for solving multiple traveling salesman problems in urban logistics. *International Journal of Applied Earth Observation and Geoinformation*, 129:103863, 2024.

[161] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655, 2021. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[162] D. Liu, K. Sun, Z. Wang, R. Liu, and Z.-J. Zha. Frank-wolfe network: An interpretable deep structure for non-sparse coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):3068–3080, Sept. 2020.

[163] J. Liu, X. Chen, Z. Wang, W. Yin, and H. Cai. Towards constituting mathematical structures for learning to optimize. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21426–21449. PMLR, 23–29 Jul 2023.

[164] M. Lombardi and S. Gualandi. A lagrangian propagator for artificial neural networks in constraint programming. *Constraints*, 21(4):435–462, Oct 2016.

[165] Z. Long, Y. Lu, and B. Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, Dec. 2019.

[166] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning pdes from data, 2018.

[167] A. Lucena and J. E. Beasley. Branch and cut algorithms. *Advances in linear and integer programming*, 4:187–221, 1996.

[168] D. Lunga, S. Prasad, M. M. Crawford, and O. Ersoy. Manifold-learning-based feature extraction for classification of hyperspectral data: A review of advances in manifold learning. *IEEE Signal Processing Magazine*, 31(1):55–66, 2014.

[169] G. Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5, 05 2016.

[170] K. Lv, S. Jiang, and J. Li. Learning gradient descent: Better generalization and longer horizons, 2017.

[171] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[172] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transp. Sci.*, 18(1):1–55, 1984.

[173] J. Mandi and T. Guns. Interior point solving for lp-based prediction+optimisation. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7272–7282. Curran Associates, Inc., 2020.

[174] P. MarinVlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolinek. Differentiation of blackbox combinatorial solvers. *ArXiv*, abs/1912.02175, 2019.

[175] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[176] A. F. T. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068, 2016.

[177] R. Matai, S. P. Singh, and M. L. Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25, 2010.

[178] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[179] S. Mei, A. Montanari, and P.-M. Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[180] L. Metz, N. Maheswaranathan, C. D. Freeman, B. Poole, and J. Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *CoRR*, abs/2009.11243, 2020.

[181] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4556–4565. PMLR, 09–15 Jun 2019.

[182] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022.

[183] I. Mitrai and P. Daoutidis. Machine learning-based initialization of generalized benders decomposition for mixed integer model predictive control. In *2024 American Control Conference (ACC)*, pages 4460–4465, 2024.

[184] M. Mladenović, T. Delot, G. Laporte, and C. Wilbaut. The parking allocation problem for connected vehicles. *Journal of Heuristics*, 26(3):377–399, June 2020.

[185] L. G. Mona Holmqvist and A. Wernberg. Generative learning: learning beyond the learning situation. *Educational Action Research*, 15(2):181–208, 2007.

[186] V. Monga, Y. Li, and Y. C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing, 2020.

[187] M. Morabit, G. Desaulniers, and A. Lodi. Machine-Learning–Based Column Selection for Column Generation. *Transportation Science*, 55(4):815–831, July 2021.

[188] K. P. Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[189] V. Nair, S. Bartunov, F. Gimeno, I. Von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[190] V. Nair, D. Dvijotham, I. Dunning, and O. Vinyals. Learning fast optimizers for contextual stochastic integer programs. In *UAI*, pages 591–600, 2018.

[191] M. G. Narciso and L. A. N. Lorena. Lagrangean/surrogate relaxation for generalized assignment problems. *European Journal of Operational Research*, 114(1):165–177, 1999.

[192] R. Neamatian Monemi, S. Gelareh, and N. Maculan. A machine learning based branch-cut-and-benders for dock assignment and truck scheduling problem in cross-docks. *Transportation Research Part E: Logistics and Transportation Review*, 178:103263, 2023.

[193] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, Jun 1996.

[194] T. Öncan. A survey of the generalized assignment problem and its applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141, 2007.

[195] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *CoRR*, abs/2111.06283, 2021.

[196] A. Paren, L. Berrada, R. P. K. Poudel, and M. P. Kumar. A stochastic bundle method for interpolating networks. *J. Mach. Learn. Res.*, 23(1), Jan. 2022.

[197] A. Parjadis, Q. Cappart, B. Dilkina, A. Ferber, and L.-M. Rousseau. Learning lagrangian multipliers for the travelling salesman problem, 2023.

[198] A. Paulus, G. Martius, and V. Musil. LPGD: A general framework for backpropagation through embedded optimization layers. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[199] A. Paulus, M. Rolinek, V. Musil, B. Amos, and G. Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8443–8453. PMLR, 18–24 Jul 2021.

[200] R. Pauwels, E. Tsiligianni, and N. Deligiannis. Hcgm-net: A deep unfolding network for financial index tracking. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3910–3914, 2021.

[201] R. Pavón, F. Díaz, R. Laza, and V. Luzón. Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *Expert Systems with Applications*, 36(2, Part 2):3407–3420, 2009.

[202] H. Peng, S. Thomson, and N. A. Smith. Backpropagating through structured argmax using a SPIGOT. *CoRR*, abs/1805.04658, 2018.

[203] Y. Peng, B. Choi, and J. Xu. Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Science and Engineering*, 6(2):119–141, Jun 2021.

[204] L. Pinheiro Cinelli, M. Araújo Marins, E. A. Barros da Silva, and S. Lima Netto. *Variational Autoencoder*, pages 111–149. Springer International Publishing, Cham, 2021.

[205] D. Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999.

[206] M. D. Plummer. Well-covered graphs: a survey. *Quaestiones Mathematicae*, 16(3):253–287, 1993.

[207] B. Polyak. *Introduction to Optimization*. Optimization Software, New York, 1987.

[208] A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V.-M. Le, K. Pilsl, M. Enzelsberger, and T. N. T. Tran. An overview of machine learning techniques in constraint solving. *Journal of Intelligent Information Systems*, 58(1):91–118, Feb 2022.

[209] S. Požgaj, A. S. Kurdija, M. Šilić, G. Delač, and K. Vladimir. An overview of the state-of-the-art machine learning methods for traveling salesman problem. In *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, pages 844–849, 2024.

[210] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[211] S. Qiu, Y. Liang, and Z. Wang. Optimizing sparse matrix multiplications for graph neural networks. *CoRR*, abs/2111.00352, 2021.

[212] Q. Qu, X. Li, Y. Zhou, J. Zeng, M. Yuan, J. Wang, J. Lv, K. Liu, and K. Mao. An improved reinforcement learning algorithm for learning to branch. *CoRR*, abs/2201.06213, 2022.

[213] P. A. Rey and C. Sagastizábal. Dynamical adjustment of the prox-parameter in bundle methods. *Optimization*, 51(2):423–447, 2002.

[214] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[215] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[216] G. T. Ross and R. M. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical programming*, 8(1):91–103, 1975.

[217] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. M. Bronstein, and F. Monti. SIGN: scalable inception graph neural networks. *CoRR*, abs/2004.11198, 2020.

[218] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[219] T. Runarsson and M. Jonsson. Evolution and design of distributed learning rules. In *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No.00*, pages 59–63, 2000.

[220] O. Rybkin, K. Daniilidis, and S. Levine. Simple and effective vae training with calibrated decoders. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9179–9189. PMLR, 18–24 Jul 2021.

[221] S. S. Sahoo, A. Paulus, M. Vlastelica, V. Musil, V. Kuleshov, and G. Martius. Backpropagation through combinatorial algorithms: Identity with projection works. In *The Eleventh International Conference on Learning Representations*, 2023.

[222] R. Sambharya, G. Hall, B. Amos, and B. Stellato. End-to-end learning to warm-start for real-time quadratic optimization, 2022.

[223] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. A survey of methods for automated algorithm configuration. *Journal of Artificial Intelligence Research*, 75:425–487, 2022.

[224] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[225] A. Schrijver et al. On cutting planes. *Combinatorics*, 79:291–296, 1980.

[226] J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[227] R. E. Shawi, M. Maher, and S. Sakr. Automated machine learning: State-of-the-art and open challenges. *CoRR*, abs/1906.02287, 2019.

[228] J. Shen, X. Chen, H. Heaton, T. Chen, J. Liu, W. Yin, and Z. Wang. Learning a minimax optimizer: A pilot study. In *International Conference on Learning Representations*, 2021.

[229] Y. Shen, Y. Sun, X. Li, Z. Cao, A. Eberhard, and G. Zhang. Adaptive stabilization based on machine learning for column generation, 2024.

[230] K. Smagulova and A. P. James. A survey on lstm memristive neural network architectures and applications. *The European Physical Journal Special Topics*, 228(10):2313–2324, Oct 2019.

[231] S. Sobolevsky. Hierarchical graph neural networks. *CoRR*, abs/2105.03388, 2021.

[232] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. Mit Press, 2011.

[233] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[234] N. Sugishita, A. Grothey, and K. McKinnon. Use of Machine Learning Models to Warmstart Column Generation for Unit Commitment. *INFORMS Journal on Computing*, page ijoc.2022.0140, Jan. 2024.

[235] C. Sutton and A. McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[236] R. Sutton and A. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[237] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[238] R. R. T. *Convex Analysis*. Princeton University Press, 2015.

[239] S. Takabe and T. Wadayama. Theoretical interpretation of learned step size in deep-unfolded gradient descent, 2020.

[240] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 13–18 Jul 2020.

[241] M. Tanneau and P. Van Hentenryck. Dual lagrangian learning for conic optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 55538–55561. Curran Associates, Inc., 2024.

[242] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003.

[243] C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11(10):311–365, 2010.

[244] S. Thrun and L. Pratt. *Learning to Learn*. Springer US, 2012.

[245] J. Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[246] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[247] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[248] A. Velloso and P. Van Hentenryck. Combining deep learning and optimization for preventive security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(4):3618–3628, 2021.

[249] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

[250] C. Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

[251] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305, 2008.

[252] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[253] F. Wang, Q. He, and S. Li. Solving combinatorial optimization problems with deep neural network: A survey. *Tsinghua Science and Technology*, 29(5):1266–1282, 2024.

[254] J. Wang, Z. Shi, X. Liang, D. Lian, S. Ji, B. Li, E. Chen, and F. Wu. Provably convergent subgraph-wise sampling for fast gnn training, 2024.

[255] P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 09–15 Jun 2019.

[256] Q. Wang and C. Tang. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, 233:107526, 2021.

[257] X. Wang, S. Yuan, C. Wu, and R. Ge. Guarantees for tuning the step size using a learning-to-learn approach, 2021.

[258] Z. Wang, X. Li, J. Wang, Y. Kuang, M. Yuan, J. Zeng, Y. Zhang, and F. Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023.

[259] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 499–515, Cham, 2016. Springer International Publishing.

[260] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3751–3760. PMLR, 06–11 Aug 2017.

[261] B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *CoRR*, abs/1809.05504, 2018.

[262] L. A. Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[263] L. A. Wolsey. *Integer Programming*. Wiley, Hoboken, NJ, second edition edition, 2021.

[264] Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018.

[265] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[266] X. Xiong and F. De la Torre. Supervised descent method and its applications to face alignment. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539, 2013.

[267] X. Xu, P.-F. Hung, and Y. Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62(1):151–171, 1996.

[268] M. Yagiura, T. Yamaguchi, and T. Ibaraki. *A Variable Depth Search Algorithm for the Generalized Assignment Problem*, pages 459–471. Springer US, Boston, MA, 1999.

[269] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[270] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *CoRR*, abs/2007.15745, 2020.

[271] T. Yang, H. Ye, and H. Xu. Learning to generate scalable milp instances. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '24 Companion, page 159–162, New York, NY, USA, 2024. Association for Computing Machinery.

[272] Y. Yang and L. wu. Machine learning approaches to the unit commitment problem: Current trends, emerging challenges, and new strategies. *The Electricity Journal*, 34, 11 2020.

[273] Y. Ye and S. Ji. Sparse graph attention networks. *CoRR*, abs/1912.00552, 2019.

[274] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin. Understanding straight-through estimator in training activation quantized neural nets, 2019.

[275] T. Yoon. Confidence threshold neural diving, 2022.

[276] A. S. Younger, S. Hochreiter, and P. R. Conwell. Meta-learning with back-propagation. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, 3:2001–2006 vol.3, 2001.

[277] T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689, 2020.

[278] Y. Yu, X. Si, C. Hu, and J. Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019.

[279] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph transformer networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[280] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio. Parameterizing branch-and-bound search trees to learn branching policies, 2021.

[281] X. Zhang, Y. Lu, J. Liu, and B. Dong. Dynamically unfolding recurrent restorer: A moving endpoint control method for image restoration, 2018.

[282] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li. Improving deep neural networks using softplus units. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–4. IEEE, 2015.

[283] Y. Zhu, F. Lyu, C. Hu, X. Chen, and X. Liu. Encoder-decoder architecture for supervised dynamic graph learning: A survey, 2022.

[284] M.-A. Zöller and M. F. Huber. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, 70:409–472, 2021.