Solving repeated optimization problems by Machine Learning

Marc Etheve

HESAM UNIVERSITÉ

le cnam

ÉCOLE DOCTORALE SMI
Laboratoire CEDRIC, CNAM

THÈSE

présentée par : **Marc ETHEVE**
soutenue le : **03 décembre 2021**

pour obtenir le grade de : **Docteur d'HESAM Université**

préparée au : **Conservatoire national des arts et métiers**

Discipline : **Informatique**

Using machine learning to solve repeated optimization problems

THÈSE dirigée par :
Mme KEDAD-SIDHOUM Safia Professeure des universités, Cnam

et co-encadrée par :
M. ALÈS Zacharie Maître de conférences, ENSTA IP Paris
M. BISSUEL Côme Ingénieur de recherche, EDF R&D
M. JUAN Olivier Ingénieur de recherche, EDF R&D

Jury

Mme Claudia D'AMBROSIO	Directrice de recherche, Ecole Polytechnique	Présidente
M. Andrea LODI	Professeur, Jacobs Technion-Cornell Institute	Rapporteur
M. Bruno SCHERRER	Chargé de recherche, INRIA Nancy Grand Est / IECL Université de Lorraine	Rapporteur
M. Axel PARMENTIER	Maître de conférences, Ecole Nationale des Ponts et Chaussées	Examinateur
M. Nicolas THOME	Professeur, Conservatoire national des arts et métiers	Examinateur

THÈSE

Affidavit

Je soussigné / soussignée, Marc ETHEVE, déclare par la présente que le travail présenté dans ce manuscrit est mon propre travail, réalisé sous la direction scientifique de Safia KEDAD-SIDHOUM (directeur / directrice) et de Zacharie ALÈS (co-directeur / co-directrice), dans le respect des principes d'honnêteté, d'intégrité et de responsabilité inhérents à la mission de recherche. Les travaux de recherche et la rédaction de ce manuscrit ont été réalisés dans le respect de la charte nationale de déontologie des métiers de la recherche. Ce travail n'a pas été précédemment soumis en France ou à l'étranger dans une version identique ou similaire à un organisme examinateur.

Fait à Paris, le 31/12/2021

Marc ETHEVE

Affidavit

I, undersigned, Marc ETHEVE, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Safia KEDAD-SIDHOUM (thesis director) and of Zacharie ALÈS (co-thesis director), in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with the French charter for Research Integrity. This work has not been submitted previously either in France or abroad in the same or in a similar version to any other examination body.

Place Paris, date 31/12/2021

Marc ETHEVE

Acknowledgements

First and foremost, I would like to warmly thank all the colleagues at EDF who have made these three years pass so fast. The working environment in PRISME and more particularly P12 has been really valuable, so I really want to thank all of you who made me fill that comfortable in this ecosystem.
Lucky me, this is a place where I can safely give names and thank, here by alphabetical order, Alain, Anne, Bruno, Claire, Côme, Julien, Laura, Loïc, Louis, Luc, Nicolas, Olivier, Pauline, Renaud, Yasmina... But let me be more exhaustive for those who have been the pillars of three years of coffee breaks, either physically or remotely.

Côme, thank you for continuously letting your door open (as well as your computer, sometimes). I always found you ready to answer or at least provide relevant arguments to any question I could have. And we both know that all of them were not that clever. You also greatly improved my knowledge, not only on scientific subjects but also on a wide variety of topics, such as zoology, borderline jokes, bad faith... and more esoteric areas.
Speaking of esoteric topics, I cannot do without thank the oldest child I ever met. Alain, your happiness, enthusiasm but also your "interest" and sharp-mindedness have been of great support. Hope we will have that discussion on quantiles eventually.
Children reveal their full potential when guided by adults. Everybody's entourage should have a Bruno, and I am very grateful for your kindness, patience and humour. "The Dude abides".
Olivier, thank you very much for your expertise and all the drawings and discussions we had... even if it was sometimes difficult to remember their starting point. Know that I really appreciated your Chatou Tuesdays, and not only for professional purposes.

ACKNOWLEDGEMENTS

Drifting from Chatou to Paris, I really want to sincerely thank my academic supervisors as well as every members of the CEDRIC laboratory.
Safia, your genuine and continuous support has been really precious. Thank you very much for helping me to apprehend all these new notions, and also for keeping me close to the concerns of the combinatorial community. Your remarks often helped me to enlarge my way of thinking, and that is of the utmost value.
Zacharie, your remarks and questions were also always relevant. Your willing and capacity to reflect on any kind of question are impressive, and perfectly illustrate the interest of having more than one member in a team.

Due to the Covid situation and the emergence of teleworking, work has inevitably encroached on my personal life. Fortunately enough, I had the chance to share this time with my Petit Babtou, who I believe sincerely supported me without any second thoughts. I will always remember your high-level insights about how making trees grow small. Thank you.

Résumé

Cette thèse a pour but d'utiliser des techniques d'apprentissage automatique pour la résolution de problèmes d'optimisation combinatoire. De par sa position de premier producteur français d'électricité, Electricité de France (EDF) doit continuellement piloter différents sites de production, ce qui se traduit mathématiquement par la résolution de problèmes linéaires en nombres entiers (en anglais Mixed Integer Linear Programming problems). A cet égard, EDF doit régulièrement résoudre des instances issues de ces problèmes, définies par des données stochastiques. Actuellement, ces instances sont résolues par un algorithme de Branch and Bound (B&B), sans tirer profit des potentielles similarités entre l'instance courante et celles résolues par le passé. Afin de conserver la garantie d'optimalité fournie par l'algorithme de B&B, nous nous proposons, pour un problème donné, d'apprendre différentes stratégies au sein de cet algorithme, comme par exemple la stratégie de branchement (sélection de variable) ou de sélection de nœud. Le principal critère utilisé afin d'évaluer la performance des stratégies proposées est la taille de l'arbre de B&B généré.

La principale approche développée dans ce travail est l'utilisation d'apprentissage par renforcement pour découvrir de telles stratégies par essais/erreurs sur les instances historiques. Afin de s'adapter à l'environnement induit par l'algorithme de B&B, nous définissons un nouveau type de transitions au sein de processus de décision markoviens (en anglais Markov Decision Processes), basées sur la structure d'arbre binaire. Par ailleurs, nous étudions différents modèles de coûts. Du point de vue de la minimisation de la taille des arbres de B&B, nous prouvons l'optimalité du modèle de coût unitaire sous le modèle de transition classique ainsi que sous le modèle de transition binaire, dans l'apprentissage non seulement de la stratégie de branchement mais également de la stratégie de sélection de nœud. Pour autant, les expérimentations menées pour la stratégie de branchement suggèrent qu'il peut être préférable d'incorporer un biais dans le modèle de coût afin d'améliorer la stabilité du processus d'apprentissage. En ce qui concerne l'apprentissage de la stratégie de sélection de nœud,

nous démontrons l'optimalité d'une stratégie explicitement définie, qui peut être apprise plus efficacement de manière supervisée.
En plus des approches mentionnées, nous proposons une stratégie de décomposition-coordination afin de potentiellement permettre le passage à l'échelle de l'apprentissage par renforcement sur des problèmes de plus grande dimension. Une heuristique de branchement basée sur une représentation par graphe d'un nœud de l'arbre de B&B est également proposée. Cette représentation peut également être utilisée afin de guider automatiquement la décomposition précédemment mentionnée. Enfin, nous présentons une approche dédiée à l'apprentissage de perturbations de la fonction objectif, afin notamment de briser d'éventuelles sources de symétrie. Les différentes méthodes proposées sont évaluées sur des problèmes réels, fournis par EDF. Pour chaque problème, deux configurations sont envisagées afin de renforcer la robustesse des résultats fournis. Un résumé plus conséquent en français est fourni en annexe.

Mots-clés : Apprentissage, MILP, Problèmes répétés

Abstract

This thesis aims at using machine learning techniques in the context of combinatorial optimization. In its capacity of main french electricity producer, Electricité de France (EDF) has to handle different production sites on a regular basis, mathematically transposed as Mixed Integer Linear Programming problems. In this context, EDF needs to frequently solve instances of these problems, defined by some stochastic data. Currently, these instances are solved using the Branch and Bound algorithm (B&B), without leveraging the potential similarity between one instance and those already solved in the past. To retain the optimality guarantee provided by the B&B algorithm, we propose to learn inner strategies of this algorithm, such as node selection and branching (variable selection), for a given problem. The main criterion chosen to evaluate the efficiency of the designed strategies is the size of the corresponding B&B tree.

The main approach developed in this work is to use reinforcement learning to discover such strategies by trials-and-errors on historical instances. To properly adapt to the B&B environment, we define a new kind of tree-based transitions, and elaborate on different cost models in the corresponding Markov Decision Processes. Regarding the problem of B&B tree size minimization, we prove the optimality of the unitary cost model under both classical and tree-based transitions, either for branching or node selection. However, we experimentally show for variable selection that it may be beneficial to incorporate some bias so as to improve the learning stability. Regarding node selection, we formally exhibit an optimal strategy which can be more efficiently learnt directly by supervised learning.

In addition to these approaches, we put forward a decomposition-coordination methodology to potentially make the learning tractable for large instances. We also propose a branching heuristic based on a graph representation of a B&B node, which may be leveraged for guiding the aforementioned decomposition. Last, we present an approach for learning to disrupt the objective function in order to break potential symmetries. To assess the quality of the different methods, we test them on two real-

world problems provided by EDF. For each problem, two configurations are encompassed to improve the strength of the results.

Keywords : Machine Learning, MILP, Repeated problems

Contents

Part I

Introduction

Chapter 1

General Introduction

This thesis is aimed at using machine learning techniques in the context of combinatorial optimization. Although interactions between the two domains were almost unexplored at the beginning of our work in the late 2010's, this field of research recently received a lot of attention and is now experiencing rapid growth. The work presented here is part of this effervescence, and hopefully may be useful to any reader interested in the topic.

In this introductory chapter, we present in a very general way the context and objectives set for this thesis. Section 1.1 introduces the setting that we consider in general terms. The objective is precised in Section 1.2, as well as the general scientific background. Last, Section 1.3 provides an overview of the document along with insights on the contributions.

1.1 Optimization of repeated problems in an industrial context

Machine learning (ML) has experienced tremendous development over the past decades, and its use in very different areas of the society is now widespread. The question of using machine learning is generally raised as soon as data can be collected or created, and patterns in their generation are to be discovered. In an industrial context, monitoring any production process generates historical and decision-related data. Therefore, it is natural to wonder whether these data can be leveraged to improve such process. As we explain in this section, this is exactly the purpose followed in this thesis in the context of energy production, provided by Electricité de France (EDF), the most important French energy producer.

In the whole document, we consider systems that are repeatedly controlled to perform well with respect to some context-dependent criteria. Formally speaking, let us write $D_t \in \mathcal{D}$ the observation of some exogenous data, or context, at time t. Facing this context, a decision maker must decide of the state of the system, by setting the value of some decision (endogenous) variables x. To ensure its feasibility, x is constrained to belong to a set $\mathcal{X}(D_t) \subseteq \mathcal{X}(\mathcal{D})$, which may be dependent on the exogenous data. The decision maker is endowed with a performance – or cost – evaluation, say $f : \mathcal{X}(\mathcal{D}) \times \mathcal{D} \to \mathbb{R}$, of any feasible point x for a given context D_t. Hence, the problem of the decision

maker can be formalized as

$$\min_{x \in \mathcal{X}(D_t)} f(x, D_t) \tag{1.1}$$

We call *repeated problems* a sequence of such problems, where only the context varies across the sequence. This is exactly the setting EDF faces on a daily (or more frequent) basis. Regularly, operators must decide of systems' states so that they behave well (and are feasible) in a given context. Let us take the simplified example of a mixed electricity plant to illustrate it. Assume that such plant can produce electricity by means of a boiler and a set of photovoltaic panels, and that it must provide a given amount of energy so that the daily demand of the area is satisfied. The setting of the plant, *i.e.* the planning and division of the production among the two sources should be determined for the next 24 hours. Here, the objective of the operator should be to satisfy the demand while minimizing its cost. The context may comprise heterogeneous data, such as the demand level at any moment of the next 24 hours, production costs for each equipment, weather forecasts, etc. The feasibility set varies with the context, as both the demand and the production capacity of the photovoltaic panels fluctuate. Besides, the performance evaluation of a given planning is also dependent on the context as production costs may vary. This situation is illustrated in Chapter 3, introducing the two problems considered in this work.

As we will see in the following, one knows how to solve the optimization problems (1.1) considered in this thesis. Hence, the challenge is not to find a solution of such problem but rather to find it quickly, or efficiently, using learning methods. Schematically, a generic algorithm is used for solving (1.1), which does not take into account that it may have already been run in the past, facing a very similar context. A simple question then arise: can we learn from past experiences to modulate the algorithm in order to make it more efficient in the future?

1.2 Problem statement and objectives

1.2.1 Mixed Integer Linear Programming and Branch and Bound

We focus on a particular subclass of the generic minimization problems formulated as (1.1), namely Mixed Integer Linear Programming problems (MILPs). More precisely, we will only consider binary

integers, so that our problems can be written as

$$p : \begin{cases} \min\limits_{x} & c^\top x \\ s.t. & Ax \leqslant b \ ; \ x \in \{0,1\}^{|\mathcal{J}|} \times \mathbb{R}^{n-|\mathcal{J}|} \end{cases} \tag{1.2}$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, m being the number of constraints, n the number of variables and $\mathcal{J}$ the indices of binary variables.

An immediate observation is that we restrict our work to linear objectives and constraints (except for the binary constraints). Although such choice is a restriction, it still enables to encompass a large variety of real-world problems. Indeed, many non-linear problems are in practice linearized in some way to be more easily solved.

The problem obtained when relaxing the binary constraints is called *linear relaxation*. It is a linear problem (LP), hence convex and solvable in polynomial time. The introduction of the binary (and more generally integer) constraints breaks the convexity of the problem and makes it NP-hard in the general case. As unfortunate as it may be, binary variables allow to model a plethora of real-world problems and thus are widely used in practice. For instance, they make it possible to consider on/off and start/stop variables, and more generally any type of discontinuous phenomena.

Various problems written as (1.2) have been identified in the combinatorial optimization community as being actually solvable in polynomial time, and specific algorithms have been designed accordingly. However, real-world applications are often more complex than problems considered in the literature, leaving the decision maker with non-polynomial generic algorithms. One of the most used of such algorithms is Branch and Bound (B&B) [1], along with its many variants.

B&B is a tree search algorithm designed to handle the integrity constraints in p, using the fact that one knows how to solve efficiently its linear relaxation. The feasible set is recursively partitioned and explored along the tree, each node being associated to an LP. In the simplest case, this LP is equivalent to the LP of its direct ascendant augmented with an additional *branching constraint* of the form $x_j = k$ with $j \in \mathcal{J}$ a binary variable and $k \in \{0,1\}$. The root node's LP is equivalent to the linear relaxation of p. As an LP is a subproblem of its ascendants in the B&B tree, its value is a valid dual bound which allows for pruning when it is greater (in the minimization case) than the current incumbent's value – or *primal bound*, *i.e.* the best value associated to an integer solution discovered earlier in the tree, if any. Likewise, feasible and infeasible nodes can be fathomed (the subsequent

subtrees are then pruned). An illustration is given in Figure 1.1.

The expansion of the tree is mainly governed by two sequential strategies, namely the *branching strategy* (or *variable selection strategy*) and the *node selection strategy*. Branching refers to the selection of the branching constraints used to create child nodes, e.g. $x_j = 0$ and $x_j = 1$ for binary simple disjunctions, while node selection defines the visiting order of the open nodes – *i.e.* neither visited nor fathomed. The process ends when each node has been visited or fathomed, guaranteeing the incumbent to be an optimal solution if any. Note that in many applications, the decision maker may not be interested in finding an optimal solution, and nodes may be pruned by bound as soon as their LP value is close enough from the incumbent's value. In this document, we only consider the case where instances are solved to optimality.

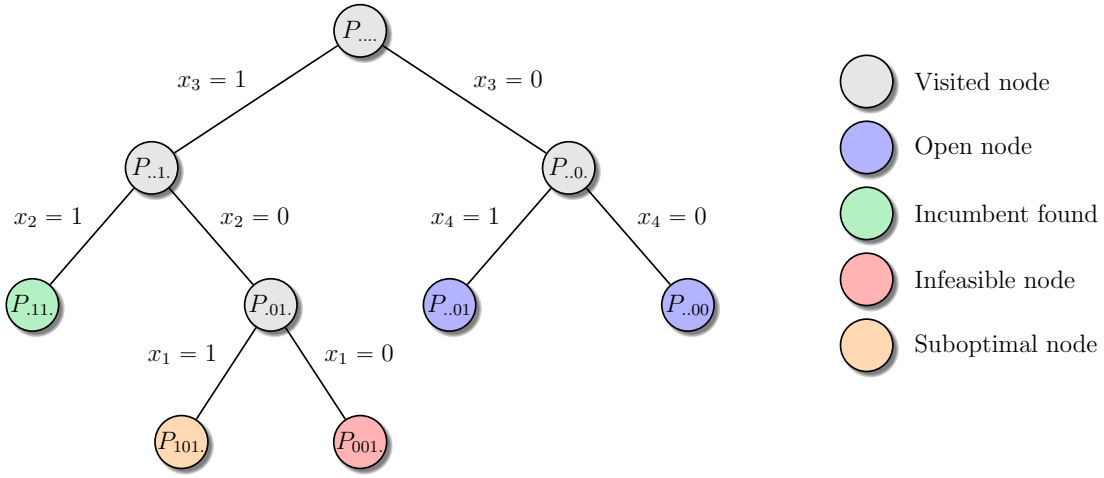


Figure 1.1: Illustration of a B&B tree during expansion with $\{1, \ldots, 4\} \subseteq \mathcal{J}$. At this point, the node selection strategy must select one of the two open nodes. If it is neither infeasible nor provably suboptimal, two child nodes will be created by the branching strategy.

1.2.2 Problem formulation

As B&B is an enumeration procedure, it usually struggles to handle large problems since the number of nodes to explore may grow exponentially with the problem size. Over the past decades, many techniques have been put forward to make the search more efficient. The majority of them are heuristics, empirically tuned on different sorts of problems. The result of all these years of incremental improvements is that current commercial solvers are now very versatile and generic tools, but also abstruse and difficult to tune for specific problems. As a consequence, learning methods may be relevant for adapting the algorithm to the problem at hand. For instance, one may try to find the

most suited solver configuration for the problem at hand. Alternatively, we can wonder whether it is possible to discover new B&B strategies for such problem. The latter option has been chosen in this work, based on the seminal question:

Given a real-world problem, can we leverage historical instances to design from scratch B&B strategies able to solve new instances of such problem while producing the smallest trees?

This formulation already separates our work from the majority of the approaches developed in the literature.

First, the focus is put on the *discovery of strategies* demonstrating high performance for the specific problem considered. In this regard, we differ from the approaches devoted to mimic the behaviour of existing heuristic strategies, initially designed to perform well on heterogeneous problems[1]. Of course, the goal is to develop a learning methodology which may be applied to design specific strategies for any MILP problem leading to repeated instances.

Second, we are interested in designing strategies dedicated to one specific problem, giving rise to *repeated instances*. It means that only the context (D_t in Equation (1.1)) changes across the considered instances, the evaluation function f and the feasible mapping $\mathcal{X}(.)$ being invariant. In the setting of Equation (1.2), it corresponds to the case where the instance data A, b, c vary across instances while keeping a similar structure: the dimensions are invariant, as well as the physical meaning of the constraints (hence the null coefficients, or at least the vast majority of them). Considering the previous example, we wish to discover a strategy designed to perform well on any instance corresponding to the considered plant. Thus, its topology does not evolve with the context.

Last, we mainly aim to develop strategies from scratch, *i.e.* without using any solver's dependent statistic. This choice is not only a practical choice but also a philosophical one. On the one hand, it is appealing for industrial companies to develop proprietary algorithms and free themselves from external contingencies. On the other hand, the objective is to discover strategies based on problem-specific correlations. Therefore, heuristic scores may not be fully appropriate and raw observations hopefully should be sufficient.

Let us consider a problem $\mathcal{P}$ of fixed dimension, for instance a production planning problem

[1]Note that a non-negligible amount of time during the first year was dedicated to the learning of existing strategies. This line of research was later abandoned due to the publication of similar works in the meantime.

for a given horizon and a specific equipment – a fixed number of production units with constant characteristics. Such a problem is perceived as an infinite support for instances $p \in \mathcal{P}$, which differ according to their associated context, *i.e.* their associated data A, b, c such as prices, demand, etc. Instance data can be seen as the outcome of a random variable, distributed according to some unknown joint probability distribution. With no loss of generality, one can directly consider the instances of a given problem as random variables, drawn from a distribution $\mathcal{L}$. From a generic standpoint, we want in this setting to discover some strategy $\pi_{\mathcal{P}}$ so as to obtain the best performance μ in average, that is

$$\pi_{\mathcal{P}} \in \arg\min_{\pi \in \Pi} \mathbb{E}_{p \sim \mathcal{L}} \left[\mu \left(p, \pi \right) \right] \tag{1.3}$$

Here, Π refers to some set of strategies related to the B&B procedure and, as aforementioned, $\mu(p, \pi)$ is the size of the B&B tree produced when solving instance p using the strategy π.

1.2.3 Learning methodology and experimental design

Equation (1.3) casts our problem as a *black-box optimization problem*, where the objective is to find the minimum of the function

$$\mu_{\mathcal{P}} : \begin{cases} \Pi \to \mathbb{R} \\ \pi \mapsto \mathbb{E}_{p \sim \mathcal{L}} \left[\mu \left(p, \pi \right) \right] \end{cases}$$

Such function is referred to as a *black-box function* as its analytic form is unknown.
Addressing directly this black-box optimization problem may appear intractable, due to the computational cost of approximating even a unique evaluation of the black-box function $\mu_{\mathcal{P}}$ – direct evaluation is not possible due to the infinite size of $\mathcal{P}$. Besides, the search space Π might be to large to be explored in this framework.

When considering sequential strategies Π, an alternative to black-box optimization is reinforcement learning (RL), which aims to learn some approximation of $\pi_{\mathcal{P}}$ by trials and errors. A cost model is to be defined to favour actions which are expected to bring the learnt strategy closer to $\pi_{\mathcal{P}}$ and, endowed with such a cost model, a strategy is learnt on training instances for a given problem's support $\mathcal{P}$. The latter approach is the one principally followed in this work.

As learning methods often depend on random factors (such as training instances, initialization, exploration, etc.), results of any experiment involving learning are averaged over at least 25 random seeds. On each seed, training and testing data are randomly sampled from either real-world or simulated data. This technique, known as Monte Carlo Cross Validation [2], is a common practice in machine learning and allows to produce results robust to random factors and data shifts.
For comparable methods, we present in this document results obtained using a unique set of hyperparameters, such as learning rates, buffer sizes, epochs, etc. This set of parameters was obtained after multiple trials, and is not tuned to one specific problem. As a consequence, these results may be improved by fine tuning those parameters for each application. This choice was made to enable a fair comparison between the different approaches. With the same objective of comparing like with like, we perform experiments with a frozen configuration of the `CPLEX`'s B&B, where presolve and cuts are disabled. For the sake of clarity, training curves are smoothed as soon as learning involves an iterative procedure.

1.3 Overview and contributions

We give here a detailed overview of this manuscript and precise the contributions. We highlight the different lines of research and provide some context related to their definition. Note that the outline of the document does not follow any chronological order. From a general standpoint, we focused on the learning of B&B strategies such as branching and node selection so as to keep the advantages provided by the B&B algorithm, especially the guarantee of optimality. These two strategies have been selected due to their core position in the B&B algorithm. Note that one could be interested in learning directly (near-)optimal solutions. However, providing solutions from a black-box model without any guarantee (either of optimality or explainability) may be questioned in an industrial context, which explains our choice.

Chapter 2 provides the background necessary to understand the manuscript. As the subject of the thesis is at the intersection of machine learning and combinatorial optimization, some basic notions from both domains are introduced. A particular emphasis is given to reinforcement learning notions. As for combinatorial optimization, we mainly introduce the reader to the most known B&B strategies.

A brief review of the literature related to the use of learning methods for combinatorial optimization is also presented.

Chapter 3 presents the real-world problems used to perform the experiments displayed in this work. The associated industrial contexts as well as mathematical formulations are given, and some observations are made to help the reader grasp the differences between the considered problems. Characteristics' variations are considered to enrich the experiments.

Part 2: Learning Oracle Strategies in a Branch and Bound Algorithm

This part contains the main axes developed in this document. Although the first idea followed during this thesis was to learn off-line a surrogate model for some expensive heuristic, it was abandoned in favour of a reinforcement learning approach. Indeed, the reinforcement paradigm matches perfectly with our objective, and was completely unexplored at this time.

Chapter 4 proposes a RL methodology aiming at discovering new branching strategies in a Branch and Bound algorithm. A particular care is taken to the selection of the Markov Decision Process, where states are associated with the B&B nodes and actions are the branching decisions. Taking some distance with the classic RL theory, we consider a new kind of tree-based transitions to better suit the structure of the environment. We show that these transitions may exhibit unfortunate theoretical limitations, and provide sufficient conditions on the environment to cancel them. We explain how RL allows to encompass strategies which may otherwise require too much computation. Different cost models are considered and we show that the unitary cost model is the most suited to our objective of tree size minimization, both under classic and tree-based transitions – under the aforementioned conditions.
To improve the performances, we digress from these theoretical recommendations and propose a biased but more robust cost model, coupled with a discount factor. Both numerical and rational justifications for this choice are given.

Chapter 5 transposes the reinforcement methodology presented in Chapter 4 to the learning of the node selection strategy, the actions being the selection of the next open node to visit. Such idea

seems natural, all the more so because the learning task should be easier in this setting. Indeed, we show that both the state space and action space are smaller than the one considered in Chapter 4. However, we explain why such approach may not be the best to discover an optimal strategy. We exhibit sufficient conditions for the definition of an intractable yet known optimal strategy, which can be directly learnt off-line by supervised learning. The question of the sample efficiency is addressed and the reinforcement learning approach is improved using oracle demonstrations.

Chapter 6 proposes different ways of combining the approaches of Chapter 4 and Chapter 5 so as to learn simultaneously branching and node selection strategies.

Part 3: Exploiting the Problems' Structure

This Part comes as a complement of Part 2. It briefly presents different approaches to design strategies by leveraging the problem's structure. Compared with Part 2, we do not aim here to take full control of inner B&B strategies to obtain oracle decisions.

Chapter 7 proposes a branching heuristic based on a graph representation of the problem. Using a proxy for the variables' mutual influence, most influential variables are selected at the root node and used for the first branching decisions. Experiments suggest that such heuristic may decrease the tree size, especially for the hardest instances of the considered problems. The interest in graph embedding for MILPs or B&B nodes in the recent literature echoes our heuristic approach.

Chapter 8 tackles the practical problem of the decomposition of large problems into smaller subproblems. As reinforcement learning suffers from the curse of dimensionality regarding the state-action space, decomposition may allow to improve its scalability. We use a Relax and Fix scheme to only solve subproblems of lower dimension. Unfortunately, Relax and Fix comes at the price of the loss of the optimality guarantee offered by B&B, and may even be unable to solve some instances. We propose to branch on coupling variables to strengthen the Relax and Fix procedure. One possible application of this scheme may be to use the methods developed in Chapter 4, 5 and 6 to learn strategies only on subproblems.

Chapter 9 focuses on adding small disruptions to objective functions so as to break potential symmetries in the problem. We use a black-box optimization framework to explore potential perturbations in an iterative fashion.

Chapter 2

Background

Content

As the work presented in this thesis is at the intersection of machine learning and combinatorial optimization, we briefly introduce in this chapter different notions from these two fields. Section 2.1 provides basic notions in supervised learning, with a focus put on imitation learning and neural networks. In Section 2.2, we introduce the reinforcement learning paradigm, which has a central role in this work. Last, Section 2.3 briefly presents Branch and Bound strategies and recent attempts of using machine learning in this context.

2.1 Supervised learning

The field of machine learning is extremely vast, with many subtleties and paradigms. We refer to the book of C. Bishop [3] for a comprehensive introduction to machine learning, with a nice statistical learning coloration. In this section, we try to keep the introduction concise and focus only on the main notions necessary to read this document.

2.1.1 A general overview

Supervised Learning (SL) is the name given to the problem of estimating a mapping between two random variables using observations from a limited number of training samples. The realizations of the (almost always) multivariate input random variable are called *features* while those of the output random variable are designated as *labels*. Denoting respectively X and Y the features and labels random variables, we write $\mathcal{X}$ and $\mathcal{Y}$ the sets where their realizations lie and P their unknown joint probability distribution. Then, the aim of SL is to find a mapping $f : \mathcal{X} \to \mathcal{Y}$ following some criterion. To do so, one is equipped with a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, where $l(y_1, y_2)$ quantifies the cost of predicting y_2 when y_1 occurs. For instance, one of the most commonly used loss is the squared error $l(y_1, y_2) = ||y_1 - y_2||_2^2$.
Considering a set of prediction functions $\mathcal{F}$, a classic objective in SL is to find a predictor with minimal risk under the unknown probability distribution P:

$$f^* \in \arg\min_{f \in \mathcal{F}} R_P(f) = \mathbb{E}_{(Y,X)\sim P}\left[l(Y, f(X))\right] \tag{2.1}$$

As the probability distribution P is unknown, such optimal predictor cannot be found by solving directly (2.1), and a common practice is then to solve its empirical counterpart

$$\hat{f} \in \arg\min_{f \in \mathcal{F}} \hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} l\left(Y_i, f(X_i)\right) \tag{2.2}$$

where $(X_i, Y_i)_{i=1,\dots,n}$ are the observed training samples. Such procedure is called *Empirical Risk Minimization* (ERM). If one assumes that the samples $(X_i, Y_i)_{i=1,\dots,n}$ are drawn independently from one another (the samples are then called *i.i.d.* for independent and identically distributed) and that $\mathbb{E}_{(Y,X)\sim P}\left[|l\left(Y, f(X)\right)|\right] < +\infty$, then the Strong Law of Large Numbers gives the almost sure convergence

$$\hat{R}_n(f) \xrightarrow[n\to\infty]{a.s.} R_P(f)$$

This convergence justifies the use of the ERM paradigm and, often, the need for a large amount of data to yield an efficient predictor.

We call *classification* (resp. *multivariate classification*) the task of finding such predictor in this supervised setting when $\mathcal{Y} = \{0, 1\}$ (resp. $\mathcal{Y} = \{0, 1\}^K$ with $K > 1$) and *regression* (resp. *multivariate regression*) when $\mathcal{Y} = \mathbb{R}$ (resp. $\mathcal{Y} = \mathbb{R}^K$ with $K > 1$).

Without diving into too many details, some remarks can be made according to the degrees of freedom left to anyone practicing SL. Of course, one of the main levers is designing and selecting the pertinent features X. This is not trivial in general and demands a good understanding of the studied phenomenon – this point is discussed a little further in Section (2.1.3). The next point is the choice of a function space $\mathcal{F}$, with a plethora of possibilities – linear, non-linear, kernel-based, with or without differentiable parameters, etc. Last but not least, the relevance of a given loss function l may depend on the considered application and should transcript the final goal at stake. However, such goal is often not well defined and the squared distance is a common default choice - R_P is called in this case the Mean Squared Error (MSE).

All these considerations often determine the ability of the predictor to generalize over unseen data or, on the contrary, its tendency to overfit the training set, *i.e.* to specialize on training instances at the cost of lower performances on unseen data. This can be illustrated through the bias-variance decomposition of the MSE, as one can show under standard assumptions that

$$\begin{aligned}\mathbb{E}_{(Y,X)\sim P}\left[\left(Y - f(X)\right)^2\right] = \mathbb{E}_{(Y,X)\sim P}\left[\left(Y - f(X)\right)\right]^2 \\ + \mathbb{V}_{(Y,X)\sim P}\left(f(X)\right) \\ + \sigma^2\end{aligned} \tag{2.3}$$

where the three terms are respectively (i) the squared bias of the predictor, (ii) the variance of the predictor and (iii) the irreducible error, that is the variance of some random noise around observations.

The more richer the features and the function space $\mathcal{F}$, the lower the bias. However, a low bias implies that the predictor represents well the oscillations in the training set, which generally comes with a high variance and a poor generalization ability (overfitting).

2.1.2 Imitation learning

Imitation learning (IL) is a specific case of SL dedicated to learning sequential control strategies from expert demonstrations, and potentially through interactions with an environment. The two main differences between IL and vanilla SL as presented above is the structure of the set $\mathcal{Y}$, which is now a structured set of trajectories (sequences of observations), and the violation of the *i.i.d.* hypothesis. We introduce here three of the main contributions in this field of research. For the sake of simplicity and conciseness, we do not focus on the convergence properties of these approaches.

Let us change a bit the terminology introduced above and consider a set of states $\mathcal{S}$ at which actions from set $\mathcal{A}$ should be taken. We call environment the whole system which samples states in $\mathcal{S}$ conditionally to some previous states and taken actions. An agent can then build trajectories through interactions with this environment by producing a sequence of actions when facing consecutive sampled states. We write $\pi_\theta : \mathcal{S} \to \mathcal{A}$ a policy parametrized by $\theta \in \Theta$ a function which selects an action $\pi_\theta(s)$ when facing state s.
As an example, one can think of a robot (agent) who can move in four directions (actions) in a maze (environment). The consequence of an action (*transition*) is governed by the environment (the robot does not progress if it tries to go through a wall and move otherwise).

In this setting, Behavioral Cloning (BC) [4] is undoubtedly the simplest form of IL as it reduces the structured prediction problem to vanilla SL techniques. Assuming that one can sample trajectories from some expert on the considered environment, these trajectories are treated as *i.i.d.* samples and BC then performs a classification task through SL by estimating

$$\theta^* \in \arg\min_{\theta \in \Theta} R_{P*}(\theta) = \mathbb{E}_{s \sim P*}\left[l(\pi^*(s), \pi_\theta(s))\right] \tag{2.4}$$

with $\pi^*(s)$ the expert's action facing state s and P^* the probability distribution for states when following the expert.

Although BC is expected to perform well when one is able to decently approach π^* by π_θ, two linked concerns may legitimately be raised at this point. First, learning from the expert's distribution does not tell anything about the actions that should be taken outside of the induced trajectories. Hence this approach may suffer from overfitting: what if deviations from P^* lead to unexplored states? Second, BC focuses on mimicking the actions of the expert, without having the possibility to take into account the cost associated to the trajectories produced by the agent. The following approaches seek to solve these drawbacks inherent to BC.

In essence, the limitations of BC can be summed up to the question of controlling the agent's risk $R_{P_\theta}(\theta)$ while learning from $R_{P^*}(\theta)$, where P_θ denotes the states distribution when following policy π_θ. This is typically handled, if possible, by interacting directly with the environment so as to gather information on $R_{P_\theta}(\theta)$. Many strategies have been designed to take into account the potential misalignment between P_θ and P^*, as well as the cost associated to the actions selected by the agent (see for example [5, 6, 7, 8, 9]). For the sake of conciseness, we only present here the two most pertinent algorithms according to the work presented in this thesis.

Dataset Aggregation, or DAgger [7], is a simple algorithm designed to learn the expert's policy on the agent's distribution. As described in Algorithm 1 (some slight modifications are brought to line up with notations), the method assumes that the expert can be invoked at any visited state. In an iterative manner, trajectories are sampled using the current agent's policy and $(s, \pi^*(s))$ pairs are collected from these samples. Stacking those samples in a growing dataset, a classifier is learnt at each iteration on the whole set of experiences collected from the first iteration. Thus, at iteration i of Algorithm 1, the agent is calibrated by estimating the solution of the theoretical program

$$\theta_i \in \arg\min_{\theta\in\Theta} \sum_{j=0}^{i-1} \mathbb{E}_{s\sim P_{\theta_j}} \left[l(\pi^*(s), \pi_\theta(s)) \right] \tag{2.5}$$

Algorithm 1 DAgger Algorithm

Initialization:
Initialize randomly the parameter θ_0
Set $\mathcal{D} \leftarrow \varnothing$
Procedure:
for $i = 1$ to N **do**:
Sample trajectories using $\pi_{\theta_{i-1}}$
Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_{\theta_{i-1}}$ and actions given by the expert
Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
Learn the parameter θ_i on $\mathcal{D}$ following the empirical counterpart of program (2.5)
end for
Output:
Best θ_i on validation

Although DAgger handles the problem of discrepancy between P_θ and P^*, it still reduces to iterative classification without taking into account the potential cost associated to non-expert trajectories. This limitation is the principal motivation behind AggreVate [9], presented in Algorithm 2. Rather than learning expert's actions through a vanilla classification task, they are learnt using a cost-sensitive classifier which solves the problem

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathbb{E}_{s \sim P} \left[Q\left(s, \pi_\theta(s)\right) \right] \tag{2.6}$$

with $Q(s, a)$ the expected cost of taking action a at s and then following the expert policy. Program (2.6) can be reduced to classic convex optimization problems, for instance by considering an argmax regression scheme. Since Q is not known in advance, we consider $\hat{Q}(.; \theta) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ a predictor for the cost-to-go Q and policies of the form

$$\pi_\theta(s) = \arg\min_{a \in \mathcal{A}} \hat{Q}(s, a; \theta) \tag{2.7}$$

The predictor is naturally trained using SL at iteration i of Algorithm 2 by solving the theoretical (regression) problem

$$\theta_i \in \arg\min_{\theta \in \Theta} \sum_{j=0}^{i-1} \mathbb{E}_{s \sim P_{\theta_j}} \left[l\left(Q(s, \pi_{\theta_j}(s)), \hat{Q}(s, \pi_{\theta_j}(s); \theta) \right) \right] \tag{2.8}$$

with l being usually a convex and differentiable loss function, *e.g.* MSE.

Algorithm 2 AggreVate Algorithm

Initialization:
  Initialize randomly the parameter θ_0
  Set $\mathcal{D} \leftarrow \varnothing$
Procedure:
  for $i = 1$ to N **do**:
    for $j = 1$ to m **do**:
      Sample uniformly $t \in \{1, \dots, T\}$ with T the maximum length of trajectories.
      Start a new trajectory in some initial state drawn from initial state distribution
      Execute current policy $\pi_{\theta_{i-1}}$ defined by Equation (2.7) up to time $t-1$
      Execute (randomly or not) some exploration action a_t in current state s_t at time t
      Execute expert from time $t+1$ to T and observe estimate of cost-to-go $\hat{Q}$
      starting at time t
    end for
    Get dataset $\mathcal{D}_i = \left\{(s, a, \hat{Q})\right\}$ of states, times, actions with expert's cost-to-go
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
    Learn the parameter θ_i on $\mathcal{D}$ following the empirical counterpart of program (2.8)
  end for
Output:
  best θ_i on validation

All the methods aforementioned undergo a common limitation. As they essentially learn to mimic the expert's behaviour on some given distribution, they are not expected to achieve better results than such expert. As a consequence, they should only be used when one has at its disposal a well-performing expert. Alternatively, the discovering of new policies, which potentially outperform that of the expert, is the object of Reinforcement Learning, another learning paradigm introduced in Section 2.2.

2.1.3 A particular function space: neural networks

As mentioned earlier, both the chosen function space $\mathcal{F}$ and the design of pertinent features are key components in SL in order to make meaningful predictions. The growing enthusiasm observed over the past few decades around neural networks is mainly due to their ability to blur the frontier between these two components.

Taking its origins in biological systems modeling, a neural network is a non-linear function $f_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by a weight vector θ, where $\mathcal{X} \subset \mathbb{R}^q$ and $\mathcal{Y} \subset \mathbb{R}^K$ with q (resp. K) the input (resp. output) dimension. The versatility of neural networks lies in the possibility to create an arbitrary

complex non-linear link between inputs and outputs while still allowing to use efficient optimization techniques to learn their weights. We introduce here the reader to some basic notions related to neural networks.

MultiLayer Perceptron (MLP)

An MLP is an acyclic (referred to as feedforward) directed graph (V, E) where a path exists from any input nodes $X \subset V$ to at least one of the output nodes $f(X) \subset V$. Conversely, any output nodes is linked to at least one input node by some path. Nodes of such graph are organized in consecutive layers as illustrated in Figure 2.1 and are called artificial neurons, with edges only linking nodes of a layer to nodes of the next one. Writing $|L|$ the number of neurons in the L-th layer, the latter can be seen as a function $f^{(L)} : \mathbb{R}^{|L-1|} \to \mathbb{R}^{|L|}$ for $1 \leqslant L \leqslant N$ and $f^{(0)} = Id_{\mathbb{R}^q}$ with N the number of layers (except from the input layer). Thus, the neural network output can be rewritten

$$f_\theta(X) = f^{(N)} \circ f^{(N-1)} \circ \cdots \circ f^{(1)} \circ f^{(0)}(X) \tag{2.9}$$

with θ a weight vector made explicit later on.

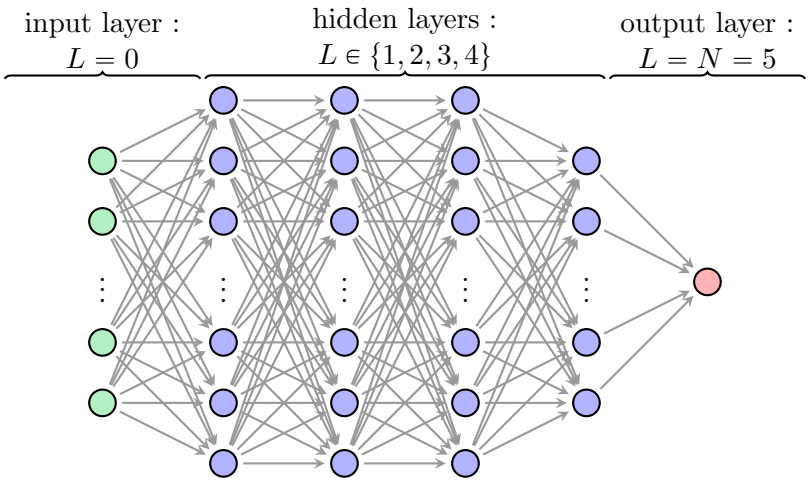


Figure 2.1: MLP illustration with 4 hidden layers with a single output node (univariate regression).

The interest of neural networks lies in the non-linearity of each function $f^{(L)}$. Using the previous notations, the value of the j-th neuron of the L-th layer $V_j^{(L)}$ is the j-th component of $f^{(L)} \circ f^{(L-1)} \circ$

$\cdots \circ f^{(1)}(X)$. Denoting $o_j^{(L)}$ such value given some input X, it is defined for $1 \leqslant L \leqslant N$ as

$$o_j^{(L)} = \phi_j^{(L)} \left(\sum_{k=1}^{|L-1|} w_{jk}^{(L)} o_k^{(L-1)} + b_j^{(L)} \right) \tag{2.10}$$

with $o_k^{(0)} \equiv X_k$ for $k \in \{1, \ldots, p\}$, $w_{jk}^{(L)} \in \mathbb{R}$ the weight of the edge from $V_k^{(L-1)}$ to $V_j^{(L)}$, $b_j^{(L)} \in \mathbb{R}$ the *bias parameter* for $V_j^{(L)}$ and $\phi_j^{(L)}$ a non-linear *activation function* (such as the sigmoid, tanh, ReLu functions and so on). In this setting, the parameter θ is the concatenation of all the weights and biases $\left(w_{jk}^{(L)}, b_j^{(L)}\right)_{\substack{L=1,\ldots,N \\ k=1,\ldots,|L-1| \\ j=1,\ldots,|L|}}$.

The term *architecture* is used to describe the number and types of links between nodes, and MLPs is a specific class among a vast amount of different architectures (residual, recurrent, convolutional, graph, auto-encoders, generative adversarial neural networks, etc.). We refer to [10] for a non-exhaustive introduction to the bestiary of most-used neural networks.

As briefly mentioned earlier, one of the greatest strengths of neural networks is that their construction as a composition of non-linear functions allows them to build complex and abstract representations of the inputs in a relevant way for the considered problem. Hidden layers, and more generally deep architectures (with numerous hidden layers), act as "useful, multistage, feature extractors with little prior knowledge" [11]. Thus, if the architecture is well chosen for the task at hand, one may provide raw inputs (*e.g.* images) and let the neural network build its own features.

Universal approximation theorem and beyond

Although many complex architectures have been created to handle different kinds of problems, the most powerful theoretical result related to neural networks concerns a basic MLP with a single hidden layer. This result, known as the universal approximation theorem, has been shown in 1991 [12] and generalized a similar result from [13]. Presented in Theorem 2.1.1, it states that any continuous function on a real compact set can be approximated by a single-layer MLP at any given precision (the more precise, the more neurons needed). This result has a direct implication: for any SL problem, one can reduce the empirical error on training set to 0 using such a neural network. Of course, this would lead the model to overfit and show poor generalization performance. This bias-variance trade-off, previously introduced, sheds light on the common practice for trying to minimize the bias while using the simplest neural network so as to reduce the model variance. Different strategies have been

developed to reduce the variance of neural networks, such as weight regularization (for instance $L1$ or $L2$ regularization), dropout [14], early stopping [15], or simply downsizing the architecture.

Theorem 2.1.1. *Let $\mathcal{C}(K_q)$ be the set of continuous functions on a given compact set $K_q \subset \mathbb{R}^q$, $f \in \mathcal{C}(K_q)$ and $\phi : \mathbb{R} \to \mathbb{R}$ an activation function which is bounded, continuous and non-decreasing.*
Then for any $\varepsilon > 0$, there exist $H \in \mathbb{N}$, $\left(w_{ki}^{(1)}\right)_{\substack{k=1,\dots,H \\ i=1,\dots,q}} \in \mathbb{R}^{Hq}$, $\left(w_{1k}^{(2)}\right)_{k=1,\dots,H} \in \mathbb{R}^{H}$, $\left(b_k^{(1)}\right)_{k=1,\dots,H} \in \mathbb{R}^{H}$ such that

$$\forall x \in K_q, \quad \left| \left(\sum_{k=1}^{H} w_{1k}^{(2)} \phi \left(\sum_{i=1}^{q} w_{ki}^{(1)} x_i + b_k^{(1)} \right) \right) - f(x) \right| \leqslant \varepsilon$$

Note here that, in Theorem 2.1.1, $f_\theta(x) = \sum_{k=1}^{H} w_{1k}^{(2)} \phi \left(\sum_{i=1}^{q} w_{ki}^{(1)} x_i + b_k^{(1)} \right) = Id \left(\sum_{k=1}^{H} w_{1k}^{(2)} o_k^{(1)} + 0 \right)$ is the output of an MLP as defined above with $N = 2$, ϕ the activation function for any unit in the hidden layer, H hidden units, no bias and the identity activation for the output layer.

Parameters estimation: backpropagation

In addition to their ability to approximate any arbitrary complex function, neural networks have gained a huge popularity due to the efficiency of existing calibration methods. Although many variants have been proposed, they are mainly built upon Stochastic Gradient Descent (SGD) originated in [16], which performs the following update

$$\theta \leftarrow \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta l \left(Y_i, f_\theta(X_i) \right) \tag{2.11}$$

where α is the learning rate, l is a differentiable loss function and $(X_i, Y_i)_{i=1}^m$ is a random batch sampled from training data.

SGD is particularly well suited for neural networks as the derivative of the loss function with respect to the weights of the network $\nabla_\theta l$ is efficiently computed by the backpropagation algorithm [17] as presented below.

Backpropagation lies on the fact that any derivative $\frac{\partial l(Y_i, X_i)}{\partial w_{jk}^{(L)}}$ can be computed using quantities derived in the calculation of the next layer's derivatives. Therefore, backpropagation works by computing first last layer's derivatives and iterating backward to progressively calculate those in previous layers. We detail here the derivatives without any bias for the output layer and for an arbitrary layer L as a function of quantities calculated when deriving those of layer $L + 1$. To alleviate the notations, let us write $l_i \equiv l \left(Y_i, f_\theta(X_i) \right)$, $a_j^{(L)} \equiv \sum_{k=1}^{|L-1|} w_{jk}^{(L)} o_k^{(L-1)}$ and consider a unique differentiable activation

function by layer, so as we have

$$o_j^{(L)} = \phi^{(L)}\left(a_j^{(L)}\right) \tag{2.12}$$

Using the chain rule, we have for the output layer

$$\frac{\partial l_i}{\partial w_{1k}^{(N)}} = \frac{\partial l_i}{\partial o_1^{(N)}}\frac{\partial o_1^{(N)}}{\partial w_{1k}^{(N)}} = \frac{\partial l_i}{\partial o_1^{(N)}}\frac{\partial o_1^{(N)}}{\partial a_1^{(N)}}\frac{\partial a_1^{(N)}}{\partial w_{1k}^{(N)}} = \frac{\partial l_i}{\partial o_1^{(N)}}\phi^{(N)'}\left(a_1^{(N)}\right)o_k^{(N-1)}$$

where $o_1^{(N)} \equiv f_\theta(X_i)$.

Considering now an inner layer L, let us compute the derivative of the loss with respect to a weight in such layer, assuming that we have computed the terms $\frac{\partial l_i}{\partial o_k^{(L+1)}}$ and $\phi^{(L+1)'}\left(a_k^{(L+1)}\right)$ for any $k \in \{1, \dots, |L+1|\}$. Still by using the chain rule, we have

$$\frac{\partial l_i}{\partial w_{ml}^{(L)}} = \frac{\partial l_i}{\partial o_m^{(L)}}\frac{\partial o_m^{(L)}}{\partial a_m^{(L)}}\frac{\partial a_m^{(L)}}{\partial w_{ml}^{(L)}} = \frac{\partial l_i}{\partial o_m^{(L)}}\phi^{(L)'}\left(a_m^{(L)}\right)o_l^{(L-1)}$$

Recalling the expression (2.9), l_i can be expressed as a function of the independent terms $\left(o_k^{(L+1)}\right)_{k=1,\dots,|L+1|}$. Then, applying again the chain rule gives

$$\frac{\partial l_i}{\partial o_m^{(L)}} = \sum_{k=1}^{|L+1|}\frac{\partial l_i}{\partial o_k^{(L+1)}}\frac{\partial o_k^{(L+1)}}{\partial a_k^{(L+1)}}\frac{\partial a_k^{(L+1)}}{\partial o_m^{(L)}} = \sum_{k=1}^{|L+1|}\frac{\partial l_i}{\partial o_k^{(L+1)}}\phi^{(L+1)'}\left(a_k^{(L+1)}\right)w_{km}^{(L+1)}$$

which yields

$$\frac{\partial l_i}{\partial w_{ml}^{(L)}} = \phi^{(L)'}\left(a_m^{(L)}\right)o_l^{(L-1)}\sum_{k=1}^{|L+1|}\frac{\partial l_i}{\partial o_k^{(L+1)}}\phi^{(L+1)'}\left(a_k^{(L+1)}\right)w_{km}^{(L+1)}$$

where the terms in the sum have already been computed by assumption. Thus, the terms $\frac{\partial l_i}{\partial o_m^{(L)}}$ and $\phi^{(L)'}\left(a_m^{(L)}\right)$ are now available for calculations in layer $L-1$.

Although SGD is convenient for calibrating neural networks as backpropagation provides an efficient way of computing derivatives, it is important to note that l is a highly non-convex function of θ. As a consequence, the procedure may (and *a priori* does) get stuck in the potentially vast amount of saddle points or local minima. However, the analysis in [18] suggests that the probability of falling into a bad local minima with respect to learning performances quickly decreases with the neural network size.

2.2 Reinforcement learning

As briefly discussed in the introduction, reinforcement learning (RL) deals with finding good sequential control strategies. In this regard, it shares the same purpose as IL. However, contrary to the supervised setting, RL aims at discovering such controls without any labeled samples. Instead, policies are learnt through trials and errors and the learner has to discover how to map situations to actions in order to minimize some numerical cost – although the RL community often reasons in terms of maximizing a reward, we rather use costs in this document as it is more suited with our purpose. Of course, as policies may have long-lasting impacts, actions may affect the immediate cost signal but also any subsequent ones. For instance, in an attempt to learn to play Draughts, one may consider to assign a reward (resp. a cost) of 1 (resp. -1) for each capture of an adverse piece and -1 (resp. 1) for a loss of a piece.
RL is at the intersection between psychology of animal learning, optimal control (and thus dynamic programming) and machine learning. It has emerged in the late 1980s especially with the works of Richard Sutton [19] and Chris Watkins [20] and has been since then a really active and prolific field of research, with many applications in various domains such as games, robotics, finance, etc. The interested reader will find a nice introduction to RL in [21], which is a classic reference in this field. In the current section, we introduce some fundamental notions, necessary to grasp the specifics of some later developments.

2.2.1 Formal definition of the RL problem

As previously mentioned, RL is about learning policies in order to minimize some numerical cost. It is common to formalize this problem in terms of optimal control of a Markov Decision Process (MDP), originally introduced by Bellman in 1957 [22].

An RL system is defined by an *agent* (or controller) and an *environment* (or controlled system). The agent interacts at discrete time steps with the environment by performing *actions* in given *states* and, subsequently, the environment provides the agent with a new state through a *transition*. In addition, the environment may give *costs* (or *rewards*) as a (in)direct consequences of state-action pairs. The well-known Figure 2.2, adapted from [21], illustrates such system. At time t, the agent

faces state s_t and selects an action a_t through a possibly stochastic policy π_t where $\pi_t(a|s)$ is the probability of performing action a at state s. At next state, the agent receives a cost C_{t+1} in pair with a new state s_{t+1}. We then define a generic setting where the objective of the agent is to minimize the total discounted amount of costs received during a complete sequence of actions. In the following, we assume that the immediate cost is deterministic with respect to a state-action pair.

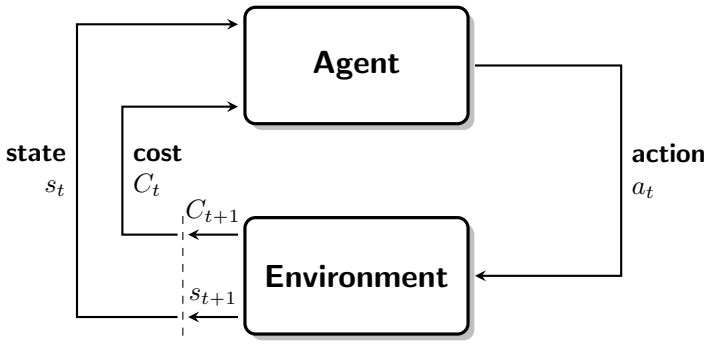


Figure 2.2: Illustration of agent-environment interactions in RL at time t

RL often assumes the environment to be markovian which gives rise to the notion of MDP, defined by its state and action sets and by the transitions between states along with the cost model. One may also include a discount factor in the definition of an MDP.
A process is said to be markovian if the probability of observing a state at time $t+1$ only depends on the state-action pair at time t, that is

$$T(s'|s,a) \equiv \mathbb{P}\left(s_{t+1} = s'|s_t = s, a_t = a\right) = \mathbb{P}\left(s_{t+1} = s'|s_t = s, a_t = a, s_{t-1}, \dots s_1, a_{t-1}, \dots, a_1\right) \quad (2.13)$$

Consider now an MDP $< \mathcal{S}, \mathcal{A}, T, c, \gamma >$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, T the transition probabilities, c a bounded cost model and $\gamma \in [0,1]$ a discount factor. A *value function* V^π associated to a policy π is defined as the expected sum of discounted costs that the agent will collect if it follows policy π from the current state s_t:

$$V^\pi(s_t) = \mathbb{E}_{\Delta^\pi}\left[\sum_{k=0}^{\infty} \gamma^k C_{t+k+1}|s_t\right] \quad (2.14)$$

where Δ^π is the state distribution when following policy π and C_t is the cost received at time t – by assumption it is a random variable if and only if the policy or the transitions are stochastic. In pair with this value function, the *action-value function*, or *Q-value function*, is defined as

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\Delta^\pi}\left[\sum_{k=0}^{\infty} \gamma^k C_{t+k+1}|s_t, a_t\right] \quad (2.15)$$

These functions satisfy recursive relationships called the *Bellman equations*:

$$V^{\pi}(s_t) = \sum_{a \in \mathcal{A}} \pi(a|s_t) \sum_{s \in \mathcal{S}} T(s|s_t, a) \left[c(s_t, a) + \gamma V^{\pi}(s) \right] \tag{2.16a}$$

$$Q^{\pi}(s_t, a_t) = \sum_{s \in \mathcal{S}} T(s|s_t, a_t) \left[c(s_t, a_t) + \gamma \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s, a) \right] \tag{2.16b}$$

with $c(s, a)$ the cost associated with the state-action pair (s, a).
In this setting, we can formally define the objective of the agent as finding an *optimal policy* which is better than any other in any state, *i.e.* which achieves a lower value for any state. Thus, writing π^* such optimal policy and $V^* \equiv V^{\pi^*}$ the associated value function, the RL task is to find

$$V^*(s) = \min_{\pi \in \Pi} V^{\pi}(s) \tag{2.17}$$

for some given set of policies Π, and such optimal value function also satisfies a Bellman recursion

$$V^*(s_t) = \sum_{a \in \mathcal{A}} \pi(a|s_t) \sum_{s \in \mathcal{S}} T(s|s_t, a) \left[c(s_t, a) + \gamma V^*(s) \right] \tag{2.18}$$

Likewise, one can define the optimal Q-function Q^*, which satisfies a similar Bellman equation. Searching for an optimal policy or an optimal value function is then equivalent, as a greedy policy with respect to an optimal value function is an optimal policy by definition.

In the following, we introduce some exact methods to derive such optimal policies (or value functions). However, in most applications, those exact methods are not tractable and one must leave it to approximations.

2.2.2 Exact methods

A common way of introducing RL methods is to begin with dynamic programming, as it allows for understanding the nature of learning through trials and errors and the need for approximation methods. Dynamic programming is a methodology designed to solve optimal control problems developed since the late 1950s, assuming perfect knowledge on the environment (transitions and costs). The two most referred dynamic programming methods are *policy iteration* and *value iteration* (VI). We introduce here only VI, which is a special case of policy iteration. The reader may refer to [21] for a more complete introduction to dynamic programming methods in this setting.

Turning the Bellman equations (2.16a) into an update rule, the dynamic programming operator is built as in Definition 2.2.1.

Definition 2.2.1. *Dynamic programming operator*

Let $V : \mathcal{S} \to \mathbb{R}$ be a value function and $\mathcal{V}$ the set of all value functions. The dynamic programming operator $\mathcal{B} : \mathcal{V} \to \mathcal{V}$ is defined by setting

$$\mathcal{B}V(s) = \min_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T(s'|s,a) \left[c(s,a) + \gamma V(s') \right] \tag{2.19}$$

Note that one may define a similar dynamic programming operator for the Q-function. This operator is a maximum-norm contraction for $\gamma \in [0,1)$ as stated in Lemma 2.2.1 and VI is the algorithm defined in Theorem 2.2.1.

Lemma 2.2.1. *The dynamic programming operator $\mathcal{B}$ is a contraction for the L_∞ norm as soon as $\gamma \in [0,1)$.*

Proof of Lemma 2.2.1. *Let us show that $||\mathcal{B}V_1 - \mathcal{B}V_2||_\infty \leqslant \gamma ||V_1 - V_2||_\infty$ for two arbitrary functions V_1 and V_2. For any $s \in \mathcal{S}$ we have*

$$\begin{aligned}
|\mathcal{B}V_1(s) - \mathcal{B}V_2(s)| = \Bigg| & \min_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} T(s'|s,a) \Big[c\left(s,a\right) + \gamma V_1\left(s'\right) \Big] \right\} \\
& \qquad - \min_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} T(s'|s,a) \Big[c\left(s,a\right) + \gamma V_2\left(s'\right) \Big] \right\} \Bigg| \\
& \leqslant \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} T(s'|s,a) \Big[\left(c\left(s,a\right) + \gamma V_1\left(s'\right) \right) - \left(c\left(s,a\right) + \gamma V_2\left(s'\right) \right) \Big] \right| \\
& \leqslant \gamma \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} T(s'|s,a) \left(V_1\left(s'\right) - V_2\left(s'\right) \right) \right| \\
& \leqslant \gamma \max_{s' \in \mathcal{S}} \left| V_1\left(s'\right) - V_2\left(s'\right) \right| \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} T(s'|s,a) \right| \\
& \leqslant \gamma \max_{s' \in \mathcal{S}} \left| V_1\left(s'\right) - V_2\left(s'\right) \right| \quad \textit{as } T \textit{ is a probability distribution}
\end{aligned}$$

□

Theorem 2.2.1. *V^* is the unique fixed point of the dynamic programming operator $\mathcal{B}$ and can be found as $V^* = \lim_{k \to +\infty} \mathcal{B}^k V_0$ with V_0 any value function and $\gamma \in [0,1)$.*

Proof of Theorem 2.2.1. *V^* is a fixed point for $\mathcal{B}$ by definition (see Equation (2.17)), and the uniqueness is induced by the contraction property of operator $\mathcal{B}$.*

Let us now consider the sequence $(V_k)_{k\in\mathbb{N}}$ *defined by* $V_{k+1} = \mathcal{B}V_k$ *with* V_0 *any value function, perceived here as a vector in* $\mathbb{R}^{|\mathcal{S}|}$.

Such sequence (V_k) *is bounded since* c *is bounded and* $\gamma \in [0, 1)$ *gives:*

$$\begin{aligned}
||V_k||_\infty &= \max_{s\in\mathcal{S}} \left| \min_{a\in\mathcal{A}} \left\{ c(s,a) + \gamma \sum_{s'\in\mathcal{S}} T(s'|s,a) V_{k-1}(s') \right\} \right| \\
&\leqslant ||c||_\infty + \gamma ||V_{k-1}||_\infty \max_{s\in\mathcal{S}} \left| \min_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} T(s'|s,a) \right| \\
&\leqslant ||c||_\infty + \gamma ||V_{k-1}||_\infty \\
&\leqslant \gamma^k ||V_0||_\infty + ||c||_\infty \sum_{i=0}^{k-1} \gamma^i \\
&\leqslant ||V_0||_\infty + \frac{||c||_\infty}{1-\gamma}
\end{aligned}$$

Besides, the sequence (V_k) *is Cauchy for the* L_∞ *norm as, for any* $k \geqslant p$*:*

$$||V_k - V_p||_\infty \leqslant \gamma ||V_{k-1} - V_{p-1}||_\infty \leqslant \cdots \leqslant \gamma^p ||V_{k-p} - V_0||_\infty \underset{k,p\to+\infty}{\longrightarrow} 0$$

since (V_k) *is bounded and* $\mathcal{B}$ *is a contraction for* $\gamma \in [0, 1)$.

As the space $\mathbb{R}^{|\mathcal{S}|}$ *equipped with the* L_∞ *norm is a Banach space, the sequence* (V_k) *converges. Let us write* V_∞ *its limit. By passage to the limit in* $V_{k+1} = \mathcal{B}V_k$ *we have* $V_\infty = \mathcal{B}V_\infty$*. Thus, by uniqueness of the fixed point for* $\mathcal{B}$*, we have* $\lim_{k\to+\infty} V_k = \lim_{k\to+\infty} (\mathcal{B})^k V_0 = V^*$. □

Starting from an arbitrary value function, value estimations are updated for each state by selecting the lowest evaluated action through the Bellman recursion. VI is guaranteed to converge at a geometric rate and may, in practice, obtain good performances after only few iterations. However, it suffers heavily from the curse of dimensionality, as one step of the algorithm requires $|\mathcal{S}|^2 |\mathcal{A}|$ evaluations. Indeed, a single update operates on each state and scans over each state-action pair from each point. In addition, applying the dynamic programming operator requires to know the transitions and cost model, which is a requirement rarely met in practice.

Another way of finding the optimal value function is to solve directly the Bellman equation at optimality if the environment's dynamics are known. As the optimal value function is the unique fixed point of the dynamic programming operator, it satisfies the equation

$$V^*(s) = \min_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} T(s'|s,a) \left[c(s,a) + \gamma V^*(s') \right] \tag{2.20}$$

for any state s. This is a non-linear system of $|\mathcal{S}|$ equation with $|\mathcal{S}|$ unknowns and thus can be solved

using any methods for solving such systems as soon as $\mathcal{S}$ is finite. Such approach does not rely on the discount factor but turns out to be intractable in practice for much smaller state spaces than dynamic programming methods. Likewise, one can cast the search of V^* into a linear system with $|\mathcal{S}|$ unknowns and $|\mathcal{S}|\,|\mathcal{A}|$ constraints (see for instance [23], page 20). The linear programming approach thus has the same limitations as the former, which lets dynamic programming be the only tractable approach in practice.

2.2.3 Approximations

In many applications, approximations may be needed for two main reasons. First, the environment may not be known, hence preventing from applying exact methods such as the aforementioned approaches. In this case, the learner should make use of approximations for value estimations, as example using Monte Carlo methods or Temporal-Difference learning (TD-learning [19]), learning from raw experiences to estimate the expectation in the VI update. Second, when the state and action sets are too large to be maintained and fully estimated by VI or approximated tabular methods (such as TD-learning), one may also use function approximation to learn a surrogate mapping, *e.g.* for the Q-value function Q^π. In both cases, the convergence speed depends on the ability of the learner to search efficiently the state space, which comes by estimating properly the according values. To achieve this in the most efficient way, a balance has to be found between exploration of new states and exploitation of states with a high estimated value. This trade-off is called the *exploration/exploitation dilemma* and exclusively arises in RL by opposition with other forms of learning. A common way of dealing with this trade-off is to perform ε-greedy exploration, which consists in selecting random actions with probability ε to explore the state-action space, with potentially a decreasing exploration probability ε over time.

We introduce here the tabular versions of Sarsa (originally named MCQ-L [24]) and Q-learning [20] as well as an approximated counterpart using function approximation, necessary to understand further developments.

TD methods are iterative procedures with updates based on a difference between two value estimates at different times, *e.g.* of the form $V(s_t) - V(s_{t+1})$. Instead of updating the values for the entire state space, iterations are performed only on explored states where value estimates are available. Updates

are said to be *on-policy* when they exclusively depend on explored state-action pairs, and *off-policy* when they depend on estimates of non-explored pairs.
Sarsa is an on-policy TD algorithm, whose updates are

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \overbrace{\left[c(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]}^{\text{TD-error}} \tag{2.21a}$$

$$\iff Q(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha \left[c(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})\right] \tag{2.21b}$$

The learnt Q-function here approximates at each step the value corresponding to the current policy guiding the exploration. On the contrary, Q-learning aims at learning directly the optimal Q-function Q^* by performing the updates

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \overbrace{\left[c(s_t, a_t) + \gamma \min_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\right]}^{\text{TD-error}} \tag{2.22a}$$

$$\iff Q(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha \left[c(s_t, a_t) + \gamma \min_{a \in \mathcal{A}} Q(s_{t+1}, a)\right] \tag{2.22b}$$

Sarsa and Q-learning are examples of model-free methods, where the updates are done using a sample-based estimate either of the Bellman or the dynamic programming operator image without trying to model the environment dynamics. We see here the importance of the exploration/exploitation trade-off as, in opposition with exact methods, the updates are made only on visited state-action pairs. As made explicit in Equations (2.21b) and (2.22b), the *learning rate* α governs how far the updates should bring the Q-function towards its images by the corresponding operator applied on visited state-action pairs. The calibration of this parameter is crucial as it may have a huge impact on the convergence of the learning procedure.

When the state space is too large, this procedure may turn out to be ineffective and one may intent to recognize patterns in the Q-function instead of trying to estimate each visited point independently. Rather than using a tabular representation of the Q-function, the state space is embedded in a feature space $\phi(\mathcal{S})$ and the Q-function is approximated by a surrogate Q-function $\hat{Q}(.,.;\theta) : \phi(\mathcal{S}) \times \mathcal{A} \to \mathbb{R}$ whose weights θ are learnt through interactions with the environment. This general framework is called *fitted Q-iteration* and was proposed in [25], later adapted to the use of neural networks in the Neural Fitted Q-iteration approach (NFQ, [26]). As mentioned in the latter, using neural networks to approximate Q-functions (sometimes referred to as *Q-networks*) may be risky as a weight update from

a specific location in the state space may have an influence on the estimations in any other regions due to the global representation mechanism in an MLP. To overcome this issue, the authors used batch learning and experience replay [27], *i.e.* sampling from previously collected experiences rather than performing on-line learning. As pointed out in the Deep Q-Network (DQN) seminal paper [28], experience replay also allows to break the samples' correlation induced by control problems, hence getting closer from the *i.i.d.* hypothesis usually made in SL (see Section 2.1.1). DQN, synthesized in Algorithm 3, is one of the most famous RL algorithms using neural networks, the main differences with NFQ being the use of a replay buffer of fixed size, the use of convolutional layers and the iterative training of a single neural network without restarting from scratch its calibration during the learning process. To encourage the independence of learning samples, DQN also introduces the use of a temporarily fixed parameter for building the Q-learning updates. As an approximation for the exact Q-learning updates (2.22a), the theoretical loss used for computing neural network updates at any iteration i of the M learning steps is

$$L_i^{DQN}(\theta_i) = \mathbb{E}_{(s,a,c,s')\sim\Delta^i}\left[\left(c + \gamma\Big[\min_{a'}\hat{Q}(s',a';\theta_i^-)\Big] - \hat{Q}(s,a;\theta_i)\right)^2\right] \tag{2.23}$$

where c is the cost obtained from state-action pair (s,a), θ_i^- is the fixed parameter periodically updated and Δ^i is a probability distribution over the experiences in the replay buffer.

Algorithm 3 DQN Algorithm

Initialization:
  Initialize replay buffer B
  Initialize a Q-network $\hat{Q}(.,.;\theta_0)$ and set $\theta_0^- = \theta_0$
Procedure:
  for $e = 1$ to M **do**:
    Draw an initial state s_1
    for $t = 1$ to T **do**:
      With probability ε select a random action a_t
      otherwise select $a_t = \arg\min_a \hat{Q}(s_t, a; \theta_t)$
      Observe the transition and store (s_t, a_t, c_t, s_{t+1}) in B
      Sample a random minibatch of transitions (s_j, a_j, c_j, s_{j+1}) from B
      Update θ_i to θ_{i+1} following the gradient derived from Equation (4.11) using the sampled minibatch
      Periodically set θ_{i+1}^- to either θ_{i+1} or θ_i^-
    end for
  end for
Output:
  Final parameter θ_N

As the optimal value function is unlikely to be found in real applications, one may ask what guarantee do we have on a policy which is greedy with respect to a suboptimal value function Q. Corollary 2 in [29] showed that the following bound holds

$$V^{\pi}(s) \leqslant V^{*}(s) + \frac{2}{1-\gamma}||Q - Q^{*}||_{\infty} \ , \ \forall s \in \mathcal{S} \tag{2.24}$$

where $V^{\pi}(s)$ is the value of state s when following the greedy policy π with respect to Q. In words, a greedy policy with respect to some approximated Q-function will perform well provided that the approximation is good enough. However, one has usually no control over the term $||Q - Q^{*}||_{\infty}$, especially when facing large state spaces.

2.2.4 Some challenges in RL

We briefly discuss here some open challenges in the RL field as they will appear in different ways in this document.

Regardless the tuning of hyper-parameters associated with the elected learning model, the choice of the discount factor γ and that of the learning rate α have often more to do with art than science. Historically introduced with respect to common practice in economics, a discount factor γ set in $(0, 1)$ is also theoretically justified as it constitutes a sufficient condition for the value functions to be defined in infinite-horizon MDPs (*cf.* the infinite sum in Equations (2.14) and (2.15)) and as it enables to obtain the maximum-norm contraction property of the dynamic programming operator (Lemma 2.2.1). However, few is known about how to fix its value. Some studies suggest that using on purpose a low discount factor allows to reduce the variance of the targets by tightening the approximate error bound [30] and may act as a regularizer to the loss function [31]. It even may increase the performance of dynamic programming methods when the reward (or cost) signal is sparse [30]. Even in episodic MDPs where the natural discount factor is 1, one may thus find it useful to set a lower value. In [32], the authors suggest that coupling an increasing discount factor with a decreasing learning rate stabilizes the DQN algorithm and allows for better performances.

A long-lasting challenge in RL is the so-called *credit assignment problem*, already highlighted by

Minsky [33] in the early 60s. It corresponds to the difficulty of identifying and thus crediting more the important decisions in a given sequence. A typical situation where the credit assignment problem arises is the case of sparse (rare) rewards, for instance in a long-lasting game as chess or go. When the only reward is given at the end of the game as a binary signal "won" or "lost", how can the agent identify the important choices among the great number of made decisions during the play? The common strategy to soften this issue is to break the task into multiple sub-tasks, thus producing more frequent reward or cost signals.

Last but not least, RL techniques often face a low sample efficiency, in the sense that they require a lot of training data before achieving good performances in the considered task. As an example, AlphaZero [34], which is one of the most famous RL achievements, played 29 million games of go before defeating the earlier state-of-the-art program AlphaGo Master (from [35]). In the Atari video game benchmark, DQN's results [28] were obtained using 50 million frames for training.
To increase the sample efficiency, one may focus on two components: the exploration policy and the learning strategy from collected experiences. Improving from ε-greedy exploration, different lines of research (see for instance [36, 37]) suggest that guiding the exploration toward promising and/or uncertain locations of the state space might increase the sample efficiency of deep reinforcement learning methods, following the *optimism in the face of uncertainty* principle of Upper Confidence Bounds [38]. Random methods have also been proposed to improve exploration, for instance in [39, 40]. Regarding the use of collected experiences, we can mention here prioritized experience replay [41], which is an extension of the previously mentioned experience replay [27]. In the former setting, learning occurs on previously collected samples using a weighting scheme, favouring the samples with high TD-errors. Finally, expert knowledge may, if possible, be leveraged to improve the sample efficiency by modifying the process to acquire training data (*e.g.* exploration) and the way to use it. Originally used only for pre-training an agent [42] (see also [35]), expert's demonstrations have also been used throughout the whole training process, for instance to shape rewards [43] or add a supervised loss on demonstrations [44]. In such approaches, demonstration data are collected once and for all before interacting with the environment, and the expert is not re-invoked during training. By contrast, [45] combines RL with a behavioral cloning loss on interactive expert's demonstrations, observed on the agent's state distribution. In [46], the authors actively invoke the expert at exploration time.

2.3 Learning in the Context of Branch and Bound

As mentioned in the introduction, our aim is to leverage learning techniques as introduced above in the context of repeated MILPs, in particular using B&B. In this section, we first overview some of the basic components or notions related to B&B. Second, we present some of the recent attempts to leverage machine learning for strategies in the B&B algorithm. Last, we widen the scope to mention some other learning approaches, related in some ways with B&B.

2.3.1 Branch and Bound strategies

As presented in the introduction (see Section 1.2.1, page 23), B&B is an algorithm designed to handle the non-convexity of MILPs induced by the integrity constraints through a divide and conquer approach, relying on the exploration of successive partitions of the feasible set. Such exploration is controlled by different kinds of strategies, such as variable selection (also known as branching), node selection, bounding, cutting, etc. Due to a lack of mathematical understanding of the dynamic nature of such strategies, state-of-the-art solvers use hand-made strategies, empirically tuned on classic benchmarks from literature (e.g. [47]). Hence, they are called *heuristics*, in the sense that they do not guarantee any optimality with respect to the size of the tree produced by the B&B procedure. The plethora of existing heuristics reflects the difficulty of the task, and their efficiency heavily depends on the problem to be solved.
We give here an overview of existing methods designed over the past few years to improve the efficiency of the B&B algorithm. It is not meant at all to be exhaustive, simply to introduce notions that may have some relevance in the understanding of the present document. A more in-depth review can be found in [48].

Presolve

Presolve is one of the most efficient techniques used in modern solvers to speed up B&B procedures and increase their ability to solve MILPs [49]. It consists in transforming the considered problem into a different but equivalent one, hopefully easier to solve. Presolve may for instance imply reformulation, bound strengthening and information extraction for later use, depending on the considered solver. As our aim is to develop techniques independent of software considerations, presolve will not be encompassed in the present work.

Node selection

Node selection selects at each iteration of a B&B algorithm one node to visit amongst the set of current open nodes. This strategy holds the responsibility for finding early good feasible solutions, that allows to prune other nodes by bound. The complexity of such strategy is its need for arbitrating between regions of the search space which are expected to contain feasible solutions and nodes with good dual bounds. In other words, it must find some balance between the probability of a node to contain a feasible solution and the expected value of such solution. Different kinds of node selection strategies are available in modern solvers, and many implement hybrid versions of the three presented below.

Depth-First Search (DFS) always selects (one of) the deepest open node(s) at each iteration. The two main advantages of DFS are to allow warm starting with low memory usage (the current optimal basis can be used to warm-start subsequent LP solves using the simplex algorithm) and to find hopefully quickly a feasible solution so as to be in a position to prune early by bound. On the other hand, DFS may spend a lot of time in regions of the search space while encountering only poor primal bounds.

Breadth-First Search (BrFS) proceeds in opposition with DFS by exploring a node at some depth only if all the nodes at a lower depth have been visited. Contrary to DFS, it is designed to handle imbalanced search spaces. However, it generally produces large trees as feasible solutions generally lies in deep nodes.

Best-First Search (BFS) is perhaps more relevant and used. It selects the open node which has the best "quality" in some sense, one of the most common quality measure being the dual bound.

Variable selection (branching)

Branching is about finding shortest paths. Conditionally to a given node selection strategy, branching governs how quick trajectories in a B&B tree will end, either by finding feasible solutions, sub-optimal or infeasible nodes. The difficulty here is double. First, a single branching choice impacts any trajectories running from the current node, and a choice that is good for one of them is not ensured to be efficient for the rest. Hence, branching decisions have to find a balance between all the trajectories rooted in the current node. Second, the three ways of ending a trajectory make the characterization of branching efficiency difficult. As suggested in [49], variable branching may be one of the most important strategies in B&B algorithms. We only present here some of the most used binary branching

rules, *i.e.* rules which partition a feasible set into two mutually exclusive sets.

Most-Fractional Branching (or Most-Infeasible Branching) is undoubtedly one of simplest used branching rule, as it consists in selecting the variable whose fractional value (in the corresponding LP solution) is closest to 0.5.

Pseudo-Cost Branching (PCB) is introduced in [50] and refers to a heuristic measure of the importance of branching candidates, by trying to assess the change in the LP value consecutively to the branching decision. This score is based on historic observations made in the B&B tree.

Strong Branching (SB), first referred without either definition or citation in [51] and first used in `CPLEX` 7.5, follows the same goal as PCB but in a more brute-force manner, by actually computing the change in the objective value. Let us write z the LP value at the current node and z_j^0 (resp. z_j^1) the LP values at child nodes when branching on $x_j = 0$ (resp. $x_j = 1$). The SB rule then selects a binary variable in $\arg\max_{j \in \mathcal{J}} \text{score}\left(z_j^0 - z, z_j^1 - z\right)$. Different scoring functions can be considered, the product being of common practice, supported by the results in [52].

Various other heuristics have been proposed over the past few years, and the most used are hybrid versions of those aforementioned (see for instance reliability branching [53] and PCB with SB initialization [54]). Additionally, *Hybrid Branching* [55] incorporates conflict clauses' lengths and values in the branching decision, which can be seen as estimates for the probability to yield infeasible nodes after branching. We shall also cite here [56] which, rather than estimating the change in the objective value, focuses on the number of active constraints.

Cutting

In LP-based B&B, the procedure lies on the bounds provided by LP relaxations to prune nodes. Cutting strategies intend to strengthen these linear relaxations by solving a separation problem, *i.e.* finding inequalities (cutting planes) valid for the convex hull of all feasible integer points but violated by the optimal solution of the current LP relaxation. These cuts may be either local – valid only for the current subtree, or global – valid for any feasible integer solution of the LP associated to root node.

Cutting strategies are not encompassed in the present work, but an interested reader may look into Gomory cuts [57] amongst others, which has the advantage of being available at low cost.

Decomposition methods

In practice, many problems arise with a very large number of variables. Facing such problems, the number of nodes necessary to obtain an optimality guarantee may be prohibitively large. However, one may take advantage of the specific structure of the problem. From a generic standpoint, decomposition methods aim at solving a Master Problem (MP), which is a lower constrained version of the original problem but which may be significantly easier to solve. The solution of such problem is then refined in an iterative manner by solving one (or many) subproblem(s) until some optimality criterion is met. The two most known techniques in this regard are column generation [58] and Benders decomposition [59].

Metrics

Different metrics may be encompassed to compare any of the aforementioned strategies. The following metrics are among the most used in the literature. They are briefly presented with some explanation of our interest with one of them particularly.

The most widespread metric used to compare B&B methods is undoubtedly the time needed to complete the procedure (*i.e.* proving infeasibility or finding an optimal solution and proving its optimality). It is used for instance in `CPLEX` reports, but also in the vast majority of integer programming articles. Duration time is particularly well suited for comparing different configurations of a single solver, or again to compare the performances of different solvers. Indeed, those are designed to have a high performance with respect to this metric, since it is often what clients are looking for. Hence it is fair to compare them using this criterion.

History of combinatorial optimization has been built by the ability of solving more and more challenging problems. Hence, a classic metric is the number of instances of a given problem to be solved optimally in a specific amount of time. This metric is out of place in this work as we will focus on problems which are already solved in reasonable time with existing B&B implementations.

In the same spirit, one can also use the number of visited nodes during the B&B procedure to compare different strategies. It is also a commonly used metrics, and one of its main advantages is to be independent on hardware or implementation specifics. As the number of nodes will be our metric of interest, it deserves a little further discussion. On the one hand, it can be a deceiving metric by concealing the computation time. A famous example of such phenomenon is SB, which performs generally quite well with respect to the number of nodes but turns out to be affected with a high processing

time. On the other hand, it truly reflects the ability of a strategy to partition and search effectively the feasible set of a given problem. In addition, it allows for a fair comparison against commercial solvers' strategies as they accumulate a long history of computational improvements, out of reach from individual works.
Last but not least in some industrial settings, the evolution of the bounds (primal and dual) throughout the search can also be used to compare strategies. Although the number of nodes is our main metric of interest, we may refer to gap metrics as it is often an interesting criterion in an industrial context, where obtaining early good solutions may be relevant.

Let us define specifically the metrics and related notions used in this work. The root node of a B&B tree is considered to be at depth 0, and a node is naturally at depth $d+1$ of the tree if its parent is at depth d. We write $|\mathcal{T}|$ the size (number of nodes) of a B&B tree $\mathcal{T}$, defined as the number of expanded or fathomed nodes during the search, *i.e.* inner and leaf nodes. The size of a dive (or trajectory) toward some node ζ is the number of nodes visited from the root node to ζ. Writing $d(\zeta)$ the depth of node ζ, the trajectory towards ζ is therefore of size $d(\zeta)+1$.
The primal integral [60] is defined, when discretizing time by the iterations of the B&B algorithm, as

$$\sum_{t=0}^{T} \Gamma(\tilde{x}_t) \tag{2.25}$$

where T is the number of iterations at the considered time of evaluation, $\tilde{x}_t$ is the best integer solution found at iteration t and

$$\Gamma(\tilde{x}_t) = \begin{cases} 1 & \text{if } c^\top x^* . c^\top \tilde{x}_t < 0 \text{ or if no integer solution has been found} \\ \dfrac{\left| c^\top x^* - c^\top \tilde{x}_t \right|}{|c^\top \tilde{x}_t|} & \text{else} \end{cases} \tag{2.26}$$

with x^* an optimal solution for the considered MILP. A common related measure is the primal-dual integral defined as

$$\sum_{t=0}^{T} c^\top \tilde{x}_t - z_t^* \tag{2.27}$$

where z_t^* is the minimum over all the dual bounds found at iteration t.

2.3.2 Machine learning and inner Branch and Bound strategies

As presented above, many components of B&B solvers are empirically-tuned heuristics. In a context where one needs to solve many combinatorial problems on a regular basis, ML appears as a

natural tool to improve upon such heuristics. The literature on this topic is quite limited as interactions between ML and combinatorial optimization is an emerging field of research, and the reader may be interested in two relevant surveys [61, 62], introducing both methodological concepts and literature overviews.

Here, we present a non-exhaustive list of approaches related to B&B inner strategies (namely branching, node selection and cutting), focusing on their goals and learning methodology.

Variable selection

We start by introducing the lines of research regarding the learning of a branching strategy, as this is the most used approach in the literature – it is also often considered as one of the most important components in B&B solvers [49, 63].

As referred in [62] from [64], learning a branching (or other) heuristic seems natural as both learning and heuristically-based decisions aim at mapping some characterization of a state (features, node description) to either take a decision (classification, variable selection). The vast majority of the approaches regarding learning to branch use imitation learning, and more specifically BC (see Section 2.1.2). In [65], the authors learn by BC to rank the candidate variables at each node using SVM^{rank} [66], considering SB as an expert. Still using BC, [63] learns the SB score (regression) using extremely randomized trees [67]. Simplifying the decision task by casting it into a classification scheme, [68] uses BC to directly learn SB decisions by means of a Graph Neural Network (see [69] for a recent review on the use of GNNs in combinatorial optimization). A similar scheme is followed in [70] with respect to the SCIP [71] default branching rule. All these approaches learn off-line the branching rule, in the sense that they first collect data using the expert's strategy, learn a surrogate decision rule on this collected data and then use it at test time on new instances. On the contrary, [72] learns online SB scores through linear regression.

We can easily understand why IL is that much leveraged for learning the variable selection strategy. First, one does not know how to solve the problem of finding an efficient branching scheme, for the simple reason that one does not know how to set this problem – define a proper tractable objective. Therefore, it is natural to mimic the empirically efficient heuristics. Second, it happens that one of the best performing branching heuristics SB, see Section 2.3.1 comes with an extremely high computational cost. Machine learning is then a way to pay this cost off-line during training rather than

online, at testing time.

Despite that, one may try to discover new strategies, or at least parametrize existing ones. This is the optic followed in [73], where the branching decision is a convex combination of two SB-like heuristic branching schemes. In this setting, the authors propose to learn the optimal weights for a given problem. In the same spirit, [74] does not propose to follow a combination of scores, but rather to dynamically (potentially at each node) select the heuristic to follow by clustering the B&B nodes – each cluster has been associated to a best performing heuristic.

Node selection

Learning node selection strategies has turned out to be less attractive, undoubtedly due to the belief that branching has more impact on the tree size. Again, IL has been leveraged to learn such strategies. In contrast with the previous paragraph, the selected experts are not aforementioned heuristics anymore. In [75], the authors leverage BC to learn a node selection and pruning strategy which dives towards a good solution and discard unpromising nodes. DAgger is used in [76] to learn a strategy which also couples both node selection and pruning, thus loosing the guarantee of optimality provided by the B&B procedure. According to the authors, the observed speedup gains mainly come from their pruning strategy.

Cut selection

This work will not treat cut selection, but cutting strategies have been learnt in a relevant fashion and thus deserve to be mentioned here. Again with the aim of assuming off-line the cost of an expensive heuristic, SL is used in [77] to learn on randomly generated data scores for cuts based on objective improvement. More related with our aim, [78] uses RL to select among Gomory cuts. The authors take as a reward the gap between objective values with and without the selected cut. The RL framework allows them to take into account ahead consequences of cuts due to the recursive nature of value functions (see Section 2.2). Their agent takes the form of a policy network in some latent space trained by Evolutionary Strategies [79], the probabilities of candidate cuts being derived from the outputs in this space. This latent space allows them to handle the varying size of their action set.

2.3.3 Widening the scope

Of course, many other ways of leveraging machine learning for combinatorial optimization have been explored. A large part of the literature has focused on learning specific algorithms dedicated to classic combinatorial problems, such as the traveling salesman problem [80, 81, 82]. Even when considering only B&B methods, plenty of decisions might be learnt without having to take into account the dynamics inherent to inner strategies. For instance, [83] proposes to learn the probability of an instance to be solved before some time limit. In [84], the authors propose to learn to tune the MILP solver's best configuration for each instance of a given problem. In a more generic way, [85] predicts the run-time of different combinatorial algorithms, which allows for example to select online among a pool of algorithms to solve a specific instance. With a similar aim, [86] uses classification to decide which (if any) decomposition should be applied to best exploit the model structure. Perhaps more aggressively, some studies suggest to directly predict the integer solution through classification [87, 88], which may turn out to be intractable for complex problems.

Chapter 3

Use Cases

Content

As explained in the introduction, our aim is to learn strategies for a given problem, in order to solve efficiently future instances of such problem. To evaluate our methods, we will focus on two different problems arising in EDF activities. This chapter is dedicated to the introduction of these two problems, described and mathematically defined. The mixed integer linear programming formulations presented here are those used in the experiments and are representative of EDF formulations. Furthermore, we introduce some variations in the characteristics of these problems in order to allow more comprehensive experiments.

3.1 Microgrid

3.1.1 Problem description

The first problem used in our experiments, schematically displayed in Figure 3.1, consists in the optimization of a gas microgrid. This is a *Unit Commitment* problem [89], where the objective is to coordinate the production units to meet a heat demand for a given discrete time horizon $\{1, \dots, T\}$ while minimizing the costs. To this end, the system considered is composed of two parallel asymmetric boilers, which directly produce heat from gas acquired at a non-stationary cost, and a cogeneration unit, producing both heat and electricity. The heat produced by the three different units can be stored before being used to satisfy the demand, at the cost of linear losses through time. As for the electricity produced by the cogeneration unit, it is to be sold on the electricity market, with varying prices through time. All the units have different and fixed characteristics, such as energy efficiency (affine here, sometimes piecewise linear in practice), production capacities and starting costs. The cogeneration unit has an additional *max-up* constraint, meaning that it cannot be used continually for too many time steps.

This problem will be referred to as the `microgrid` problem in this document. For a given configuration (which will be explicitly identified by its naming), the context or varying data across instances consist in gas prices, electricity prices, and heat demand for each time step of the discrete planning horizon. These three sequences fully identify an instance for a given problem, any other characteristic being fixed as part of the problem definition.

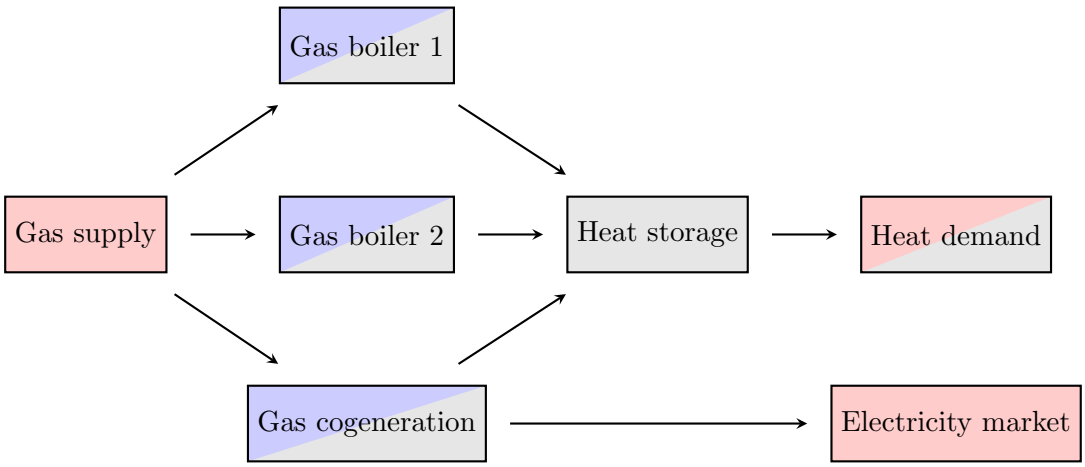


Figure 3.1: Schematic of a configuration for the `microgrid` problem.
Red cells represent the varying data, grey indicates constraints and blue the production units.

Note that many variations of the unit commitment problem have been studied in the literature, due to its importance in the production industry. The complexity of such problems depend on their structure, but its has been shown that dynamic coupling constraints, similar to *max-up* constraints, can be sufficient to prove NP-hardness [90].

3.1.2 Problem formulation

We present here the MILP formulation of the `microgrid` problem used in the experiments. We write it keeping unspecified the number of units for the sake of clarity, denoting $\mathcal{I}$ the set of units and $\mathcal{E} \subset \mathcal{I}$ that of cogenerators.

Varying data (across instances of a same problem), noted in uppercase font and indexed by the time step, are historical gas prices G_t, electricity prices E_t and heat demand D_t for each time step $t \in \{1, \dots, T\}$ with T the planning horizon.

Decision variables, identified by lowercase roman letters, are binary or continuous. The binary variables consist in the indicators $x_{i,t}$ of units' state (1 if it is on and 0 otherwise) at a given time t with $i \in \mathcal{I}$ the unit identifier, and the starting and stopping indicators $s_{i,t}$ and $z_{i,t}$ respectively, with $s_{i,t} = 1$ (resp. $z_{i,t} = 1$) if and only if $x_{i,t} = 1$ and $x_{i,t-1} = 0$ (resp. $x_{i,t} = 0$ and $x_{i,t-1} = 1$). As for the continuous variables, $y_{i,t}$ refers to the production level of unit i and r_t to the stock level at time t.

The characteristics, held constant for every instances of the corresponding problem, are referred to with greek letters – except for the number of periods, naturally written T. The terms ρ_i and α_i are

the linear and affine components of a unit efficiency, μ is the quantity of electricity by unit of heat produced by the cogenerators, β_i its starting cost, $\underline{\pi}_i, \overline{\pi}_i$ the lower and upper production capacities, and γ the inventory value at the beginning of the period. The stock loses a proportion Δ of its value by time unit, with a constant maximal capacity of ζ.

Given these characteristics, an instance defined by the data $\{G_t, E_t, D_t\}_{t=1,\dots,T}$ is then formulated as

$$\min_{x,s,z,y,r} \quad \sum_{t=1}^{T} G_t \left(\sum_{i \in \mathcal{I}} \frac{y_{i,t}}{\rho_i} + \alpha x_{i,t} \right) - E_t \sum_{i \in \mathcal{E}} y_{i,t} + \sum_{i \in \mathcal{I}} \beta_i s_{i,t}$$

$$\begin{aligned}
\text{s.t.} \quad & s_{i,t} - x_{i,t} \leqslant 0 && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c1)\\
& x_{i,t} - x_{i,t-1} - s_{i,t} \leqslant 0 && \forall i \in \mathcal{I},\ t \in \{2,\dots,T\} && (c2)\\
& x_{i,t} + s_{i,t+1} \leqslant 1 && \forall i \in \mathcal{I},\ t \in \{1,\dots,T-1\} && (c3)\\
& z_{i,t+1} - x_{i,t} \leqslant 0 && \forall i \in \mathcal{I},\ t \in \{1,\dots,T-1\} && (c4)\\
& x_{i,t} - x_{i,t+1} - z_{i,t+1} \leqslant 0 && \forall i \in \mathcal{I},\ t \in \{1,\dots,T-1\} && (c5)\\
& x_{i,t} + z_{i,t} \leqslant 1 && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c6)\\
& \underline{\pi}_i x_{i,t} \leqslant y_{i,t} && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c7)\\
& \overline{\pi}_i x_{i,t} \geqslant y_{i,t} && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c8)\\
& (1-\Delta) r_t - r_{t+1} + \sum_{i \in \mathcal{I}} y_{i,t} = D_t && \forall t \in \{1,\dots,T-1\} && (c9)\\
& r_t \leqslant \zeta && \forall t \in \{1,\dots,T\} && (c10)\\
& r_1 = \gamma && && (c11)\\
& (1-\Delta) r_T + \sum_{i \in \mathcal{I}} y_{i,T} - D_T = \gamma && && (c12)\\
& s_{i,t} - z_{i,t+1} - z_{i,t+2} \leqslant 0 && \forall i \in \mathcal{E},\ t \in \{1,\dots,T-2\} && (c13)\\
& x_{i,1} + x_{i,2} + x_{i,3} \leqslant 2 && \forall i \in \mathcal{E} && (c14)\\
& x_{i,t},\ s_{i,t},\ z_{i,t} \in \{0,1\} && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c15)\\
& y_{i,t},\ r_t \in \mathbb{R}^+ && \forall i \in \mathcal{I},\ t \in \{1,\dots,T\} && (c16)
\end{aligned}$$

Constraints $(c1-3)$ define the start variables. Due to $(c1)$, a unit i can be at a starting state only if it is on, when constraints $(c2)$ imply that it is necessarily at starting state at time t if it is on at time t and off at time $t-1$. Besides, $(c3)$ prevent an on state from preceding a start state. Constraints $(c4-6)$ define the stop variables in a similar manner. Constraints $(c7)$ (resp. $(c8)$) prevent the production level of some unit to exceed (resp. be lower than) its capacity if the unit is at on state. If the unit is

off, these two constraints force the production level to be null. The demand satisfaction is enforced by constraints ($c9$), equalizing the demand level to the production level plus or minus the stock flow. This stock has a maximal capacity defined by constraint ($c10$) and an initial value given in ($c11$). The constraint ($c12$) enforces a constant terminal value for the inventory. The max-up constraints ($c13$) state that if a cogenerator starts a time t, it must either be stopping at time $t+1$ or $t+2$. These constraints are completed by ($c14$) at the beginning of the period, stating that a cogenerator cannot be up during the first three time steps.

3.1.3 Configurations

In order to make more comprehensive experiments, we consider different configurations of the `microgrid` problem. First, we make the time horizon vary to induce different problem dimensions. Concretely, we will perform experiments with $T \in \{6, 8, 12\}$. Second, we tune the efficiency characteristics of the two parallel boilers, so as to handle different degrees of symmetries in the formulation. This setting gives rise to two problems, the one with asymmetric boilers being referred to as `micro_asym` and the more balanced one, *i.e.* with more alike boilers' characteristics, referred to as `micro_bal`. Apart from this difference, the two problems are identical and the instances are built from the same data. Table 3.1 displays the problems' dimensions as well as the average tree size of `CPLEX` on each configuration on 500 instances. As mentioned in the introduction (page 28), the experiments are performed with a restrained version of `CPLEX`'s branch and bound so as to enable fair comparisons of specific strategies.

Problem	# constraints	# variables	# binary	average tree size
`micro_asym_T6`	186	120	54	45.9
`micro_bal_T6`	186	120	54	52.0
`micro_asym_T8`	254	160	72	113.5
`micro_bal_T8`	254	160	72	95.5
`micro_asym_T12`	390	240	108	447.6
`micro_bal_T12`	390	240	108	337.5

Table 3.1: Dimensions of the different `microgrid` problems

To give some insights about the computational difficulty of solving these problems under these conditions, Figure 3.2 displays some statistics observed on the configurations encompassed in this

work.

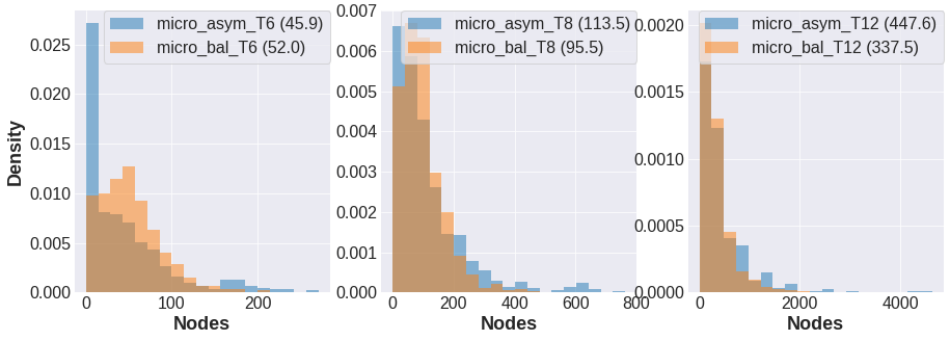
Naturally, we observe in Figure 3.2a that the tree sizes are positively correlated with the problem's dimension. For a given time horizon, both `micro_asym` and `micro_bal` are intrinsically of comparable difficulty, the main difference being the ability of `CPLEX` to find early good solutions in the asymmetric configuration – see Figure 3.2b.

Although these configurations may be similar for the considered solver, learning may be harder to perform on `micro_bal` due to the higher degree of variability in the solutions induced by its more symmetrical structure. To illustrate this point, we consider the stability score σ_j for a binary variable $j \in \mathcal{J}$ defined as

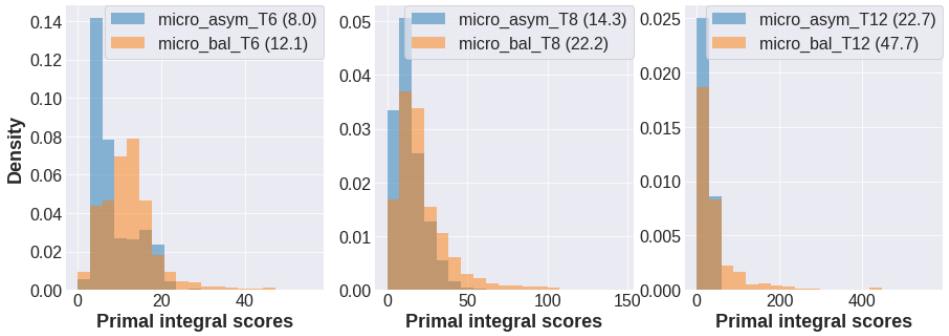
$$\sigma_j = \frac{2}{N} \left| \sum_{i=1}^{N} x^*_{i,j} - 0.5 \right| \tag{3.1}$$

where N is the number of considered instances and x^*_i the optimal solution found by `CPLEX` on instance i. This score is defined such that σ_j equals one if the variable has always the same value in the found solutions and gets closer to zero as the variability increases. Figure 3.2d represents the empirical cumulative distribution of this score across the binary variables. We see for example that, for `micro_bal_T12`, around 40% of the variables have a score under 0.5 (cf. the black dot), which means that 40% of the variables takes some value at optimality for at most 75% of the instances, and the other value for the other 25%. In the case of `micro_asym_T12`, only 15% exhibits such level of variability. Thus, the higher the curve, the more variability, and we see that `micro_bal` solutions vary more.

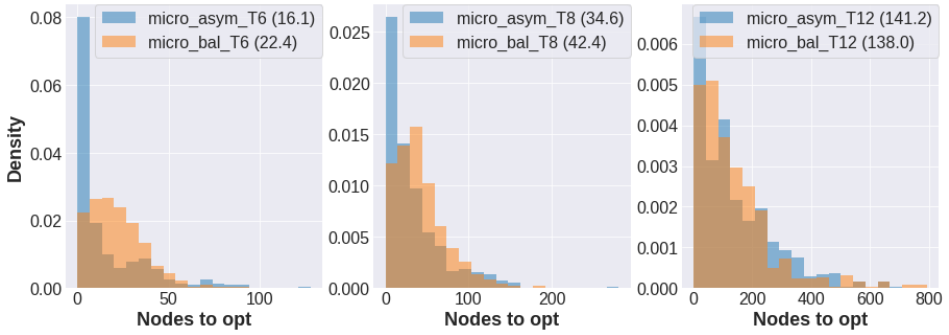
This score does not fully reflect the variability of the solutions, as 100% of the variables could have a null score in the case of only two distinct integer solutions x^* and $1 - x^*$, each one being found 50% of the time. Thus, Table 3.2 completes this score by giving the ratio of the number of distinct near-optimal integer solutions found over the number of considered instances. These solutions are obtained by setting the relative tolerance gap to 1% when collecting the pool of solutions for each instance. Here again, we see that `micro_bal` offers much more variability.



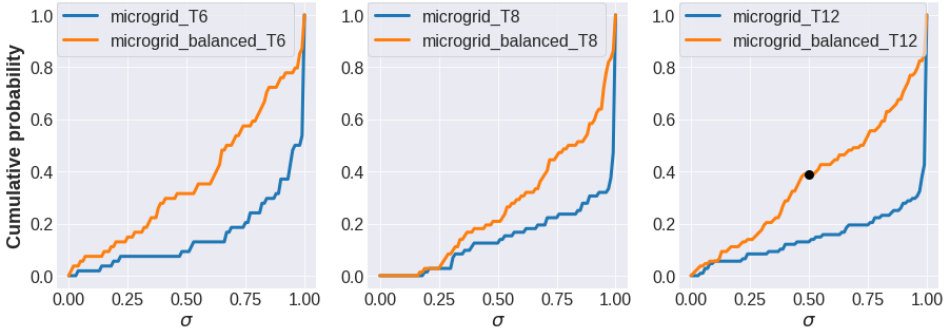
(a) Histogram of tree sizes.



(b) Histogram of primal integral scores.



(c) Histogram of nodes processed before finding an optimal solution.



(d) Empirical cumulative probability of the stability score (3.1). This Figure is to be read as follows: on `micro_bal_12`, the black dot indicates that 40% of the variables has a score lower than 0.5.

Figure 3.2: Statistics on the solving of instances from the different configurations of `microgrid`. The numbers in parentheses are mean values for the corresponding statistics.

Problem	Percentage (%)
micro_asym_T6	5.17
micro_bal_T6	17.17
micro_asym_T8	14.07
micro_bal_T8	31.36
micro_asym_T12	44.32
micro_bal_T12	99.84

Table 3.2: Percentages of distinct near-optimal integer solutions found by CPLEX on microgrid problems.

3.2 Hydroelectric valley

3.2.1 Problem description

The second use case, presented in Figure 3.3, is also a unit commitment problem, this time applied to the management of a hydroelectric valley. The objective is to minimize the costs on a discrete time horizon, selling the energy produced by hydroelectric plants. A valley consists of reservoirs and power units, where the units are powered by the flow incoming from the upstream reservoir. The flow may then continue to the next (downstream) reservoir, creating a chain of interconnected units and reservoirs. Note that in the literature, this problem may be referred to as the hydro-chain scheduling problem or again hydro valley problem.

Many valley topologies can be encompassed, from different flow sources to pumping facilities allowing for water circuits. Here, we consider a simple linear valley, to be optimized for 12 time steps. The units generate power by controlled water flow from the upstream reservoir through an ordered set of turbines, referred to as unit's components. These components also determine the level of primary and secondary spinning reserves, encouraged by the regulator. Flow variations are constrained, and some management constraints are imposed on the components' pilotage. Last, the reservoirs' volumes are constrained to fit a mid-term and end-of-period water level policy.

This problem will be referred to as the hydro problem in this document. For a given configuration (which will be explicitly identified by its naming), the context or varying data across instances are the reservoirs' policy volume constraints and electricity prices (power, primary and secondary reserves).

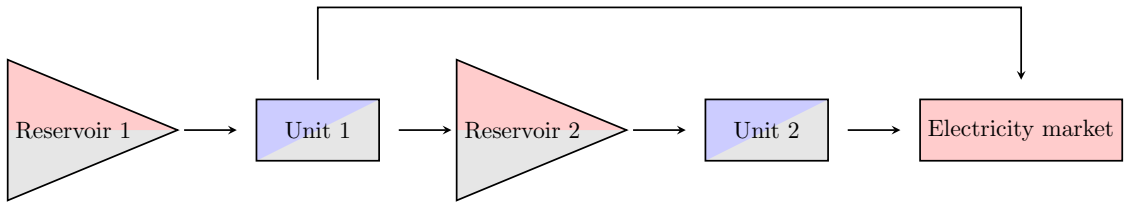


Figure 3.3: Schematic of the `hydro` problem.
Red cells represent the varying data, grey indicates constraints and blue the production units.

3.2.2 Problem formulation

We present here the MILP formulation of the `hydro` problem used in the experiments. N refers to the number of units.

Varying data, again noted in uppercase font and indexed by the time step, are simulated market prices P_t, primary and secondary prices P'_t, P''_t, and volume bounds at mid-period $\underline{V}_{i,T/2}, \overline{V}_{i,T/2}$ and at the end of the period $\underline{V}_{i,T}, \overline{V}_{i,T}$ for each unit $i \in \{1, \ldots, N\}$.

The binary variables consist in the state (on or off) of each individual unit component $x_{i,j,t}$ for unit i at time t and their starting indicators $s_{i,j,t}$, $j \in \{1, \ldots, M_i\}$ being here the identifiers of the M_i components of unit i. The continuous variables are the power, primary and secondary reserves, respectively $y_{i,t}$, $p_{i,t}$ and $z_{i,t}$, produced by unit i at time t, the flow going through units $f_{i,t}$ and reservoirs' volume $v_{i,t}$.

The characteristics of the valley, denoted with Greek letters – except for the time horizon, comprise component efficiencies for power generation (ρ_j), primary reserves (ρ'_j) and secondary reserves (ρ''_j). If component j of unit i is at on state, a flow $\alpha_{i,j}$ runs through it, and the variations of this flow across to consecutive time steps are lower (resp. upper) bounded by $\underline{\eta}_i$ (resp. $\overline{\eta}_i$). Reservoir i has limited capacities $\underline{\nu}_i$ and $\overline{\nu}_i$, its initial volume is denoted $\nu_{i,0}$, the time between two time steps is Δ and δ_i is the constant natural inflow at unit i.

An instance defined by the varying data $\left\{P_t, P'_t, P''_t, \underline{V}_{i,T/2}, \overline{V}_{i,T/2}, \underline{V}_{i,T}, \overline{V}_{i,T}\right\}_{t=1,\dots,T}^{i=1,\dots,N}$ is then formulated as

$$\min_{x,s,v,f,y,p,z} \quad -\sum_{i=1}^{N}\sum_{t=1}^{T}\left(P_t y_{i,t} + P'_t p_{i,t} + P''_t z_{i,t}\right)$$

$$\text{s.t.} \quad y_{i,t} = \sum_{j=1}^{M_i} \rho_j x_{i,j,t} \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c1)$$

$$p_{i,t} = \sum_{j=1}^{M_i} \rho'_j x_{i,j,t} \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c2)$$

$$z_{i,t} = \sum_{j=1}^{M_i} \rho''_j x_{i,j,t} \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c3)$$

$$f_{i,t} = \sum_{j=1}^{M_i} \alpha_{i,j} x_{i,j,t} \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c4)$$

$$\underline{\eta}_i \leqslant f_{i,t} - f_{i,t-1} \leqslant \overline{\eta}_i \qquad \forall i \in \{1,\dots,N\},\ t \in \{2,\dots,T\} \qquad (c5)$$

$$x_{i,j,t} \geqslant x_{i,j+1,t} \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i-1\},\ t \in \{1,\dots,T\} \qquad (c6)$$

$$x_{i,j,t} - x_{i,j,t+1} - x_{i,j,t-1} \geqslant -1 \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i\},\ t \in \{2,\dots,T-1\} \qquad (c7)$$

$$x_{i,j,t} - x_{i,j,t+1} - x_{i,j,t-1} \leqslant 0 \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i\},\ t \in \{2,\dots,T-1\} \qquad (c8)$$

$$s_{i,j,0} \geqslant x_{i,j,0} \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i\} \qquad (c9)$$

$$s_{i,j,t} \geqslant x_{i,j,t} - x_{i,j,t-1} \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i\},\ t \in \{2,\dots,T\} \qquad (c10)$$

$$v_{1,1} = \nu_{1,0} + \Delta\left(\delta_1 - f_{1,1}\right) \qquad (c11)$$

$$v_{i,1} = \nu_{i,0} + \Delta\left(\delta_i + f_{i-1,1} - f_{i,1}\right) \qquad \forall i \in \{2,\dots,N\} \qquad (c12)$$

$$v_{1,t} = v_{1,t-1} + \Delta\left(\delta_1 - f_{1,t}\right) \qquad \forall t \in \{2,\dots,T\} \qquad (c13)$$

$$v_{i,t} = v_{i,t-1}\Delta\left(\delta_i + f_{i-1,t} - f_{i,t}\right) \qquad \forall i \in \{2,\dots,N\},\ t \in \{2,\dots,T\} \qquad (c14)$$

$$\underline{\nu}_i \leqslant v_{i,t} \leqslant \overline{\nu}_i \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c15)$$

$$\underline{V}_{i,T/2} \leqslant v_{i,T/2} \leqslant \overline{V}_{i,T/2} \qquad \forall i \in \{1,\dots,N\} \qquad (c16)$$

$$\underline{V}_{i,T} \leqslant v_{i,T} \leqslant \overline{V}_{i,T} \qquad \forall i \in \{1,\dots,N\} \qquad (c17)$$

$$x_{i,j,t},\ s_{i,j,t} \in \{0,1\} \qquad \forall i \in \{1,\dots,N\},\ j \in \{1,\dots,M_i\},\ t \in \{1,\dots,T\} \qquad (c18)$$

$$v_{i,t},\ f_{i,t},\ y_{i,t},\ p_{i,t},\ z_{i,t} \in \mathbb{R}^+ \qquad \forall i \in \{1,\dots,N\},\ t \in \{1,\dots,T\} \qquad (c19)$$

Constraints $(c1-4)$ define the impact of components on production, reserves and flow, while constraints $(c5)$ limit the flow variations. Constraints $(c6)$ impose an order on the set of a unit's components,

and constraints ($c7 - 8$) forbid frequent start-ups and stops of the turbines. The start variables are defined by constraints ($c9 - 10$). Reservoirs' volume is defined in constraints ($c11 - 14$) by inflows and outflows, while the mid-period and end-of-period policies are defined in ($c16 - 17$).

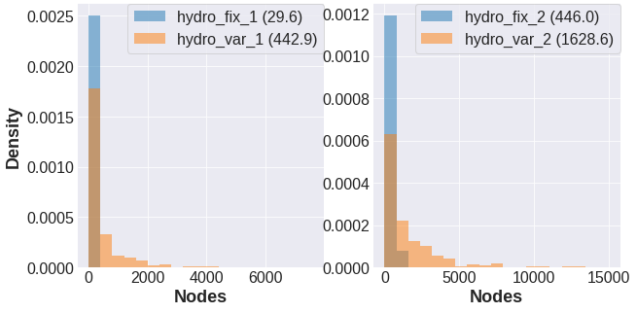
3.2.3 Configurations

Unlike `microgrid`, the main lever considered for controlling the dimension of the `hydro` problem is the number of hydroelectric power plants. We mainly perform experiments with one and two plants, that is $N \in \{1, 2\}$. The second source of heterogeneity introduced is that we consider a problem with a fixed (resp. variable) volume policy, *i.e.* volume bounds, referred to as `hydro_fix` (resp. `hydro_var`). Note that the number of turbines per unit is not the same between these two problems. Table 3.3 summarizes the problem's dimensions.

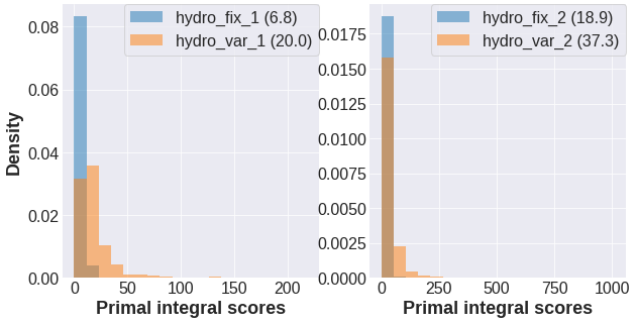
As observed in Figure 3.4, `hydro_var` is not only more intrinsically difficult to solve, both in terms of B&B nodes and ease of finding good solutions, but also more heterogeneous regarding the optimal solutions found by `CPLEX`. This is true at the variable level (Figure 3.4d) but also at the solution level (Table 3.4). Thus, learning is expected to be more difficult on `hydro_var` problems. Mathematically, the feasible set for `hydro_fix` is stable across instances due to the fixed volume policy.

Problem	# constraints	# variables	# binary	average tree size
`hydro_fix_1`	282	207	96	29.6
`hydro_var_1`	378	303	144	442.9
`hydro_fix_2`	596	366	216	446.0
`hydro_var_2`	512	342	168	1628.6

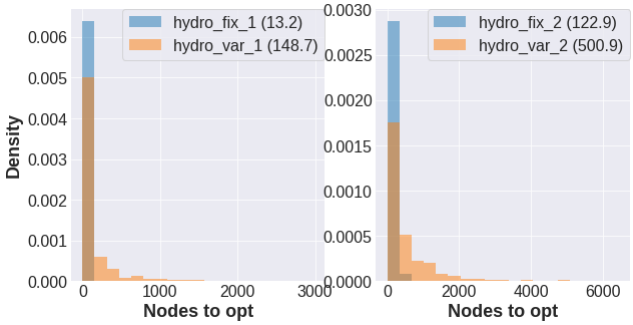
Table 3.3: Dimensions of the different `hydro` problems.



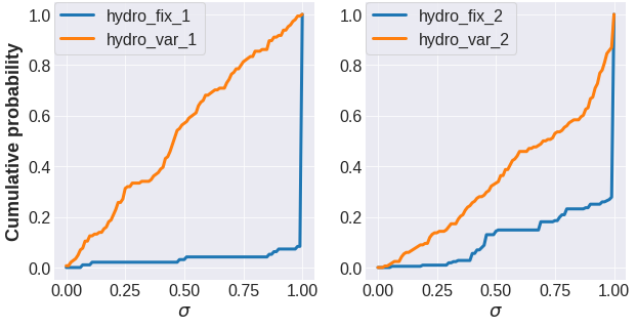
(a) Histogram of tree sizes.



(b) Histogram of primal integral scores.



(c) Histogram of nodes processed before finding an optimal solution.



(d) Empirical cumulative probability of the variability score (3.1).

Figure 3.4: Statistics on the solving of instances from the different configurations of `hydro`. The numbers in parentheses are mean values for the corresponding statistics.

Problem	Percentage
`hydro_fix_1`	6.6
`hydro_var_1`	175.8
`hydro_fix_2`	66.6
`hydro_var_2`	282.4

Table 3.4: Percentages of distinct near-optimal integer solutions found by `CPLEX` on `hydro` problems.

Part II

Learning Oracle Strategies in a Branch and Bound Algorithm

This part holds the main axes developed during this thesis. The objective is to learn strategies in a Branch and Bound algorithm. Throughout the different chapters, we consider Branch and Bound as defined in Algorithm 4, where the only degrees of freedom lie in the branching and node selection strategies. The dual bounds are provided by linear relaxations. In practice, we turn off presolve and cut generation. Such measures are common practice [65, 73, 91, 92], allowing for a strict comparison between policies, independently from interactions with other strategies inherent to the selected solver. As for the notations, we write ζ a Branch and Bound node, defined by the linear relaxation of the original MILP problem and an additional set of constraints induced by the ascendant nodes and the branching decision at its parent node. In the following, the LP of a node will refer to this linear relaxation. Note that we arbitrarily do not include the primal bound if any in the definition of a node. We write $\mathcal{D}(\zeta) \equiv \{D_0(\zeta, j), D_1(\zeta, j)\}$ the direct children of ζ after branching on variable j. Here, $D_0(\zeta, j)$ (resp. $D_1(\zeta, j)$) refers to the node defined by the LP of ζ augmented with the constraint $\{x_j = 0\}$ (resp. $\{x_j = 1\}$).

Consistently with the notations of Algorithm 4, we refer to the sets of non-leaf and leaf nodes once the tree is fully expanded as $\mathcal{N}^\pi$ and $\mathcal{L}^\pi$ when following the strategy $\pi \in \Pi$, where Π is some set of strategies of interest. Note here that these two sets form a partition of $\mathcal{T}^\pi$. Likewise, we write $\mathcal{N}_t^\pi, \mathcal{L}_t^\pi, \mathcal{O}_t^\pi, \mathcal{T}_t^\pi$ the corresponding sets at iteration t of Algorithm 4, where $\mathcal{O}_t^\pi$ stands for the set of open nodes at iteration t.
In this context, with Π a set of policies of interest (branching strategies, node selection strategies or both), we define our global objective as

$$\min_{\pi \in \Pi} \mathbb{E}_{p \sim \mathcal{L}} \left[|\mathcal{T}^\pi(p)| \right] \tag{3.2}$$

where $\mathcal{T}^\pi(p)$ refers to the tree produced by strategy π. Again, the tree size is used as a proxy for computational efficiency and independent of hardware considerations, and optimal policies for (3.2) are called *oracle strategies*. Depending on the selected strategy set Π, we propose different approaches to learn a good strategy with respect to objective (3.2). The three chapters of this part are respectively dedicated to learning the branching strategy, the node selection strategy and finally the two of them jointly.
All the experimental results presented in the following chapters are averaged over 25 random splits of 200 training instances and 300 testing instances. We will refer to the term *training processes* to

indicate the average number of nodes and primal integral scores (see Section 2.3.1, page 58) produced on training instances along any iterative learning procedure. The results presented on test instances are the average increase in terms of number of nodes compared to `CPLEX` (so that negative numbers correspond to an average decrease of the tree size).

Algorithm 4 Branch and Bound algorithm (two policies, minimization case)

Input:
A MILP instance p and ζ_0 its linear relaxation
A global strategy $\pi \equiv (\pi_N, \pi_V)$ with π_N a node selection strategy and π_V a branching strategy

Initialization:
Iteration : $t \leftarrow 0$
Solution and primal bound: $x \leftarrow$ None ; $\beta_t^\pi \leftarrow \infty$
Sets of open, non-leaf and leaf nodes : $\mathcal{O}_t^\pi \leftarrow \{\zeta_0\}$; $\mathcal{N}_t^\pi \leftarrow \varnothing$; $\mathcal{L}_t^\pi \leftarrow \varnothing$

Procedure:
while $\mathcal{O}_t^\pi \neq \varnothing$:
$t \leftarrow t+1$; $\beta_t^\pi \leftarrow \beta_{t-1}^\pi$
Node selection:
$\zeta_t \leftarrow$ a node in $\mathcal{O}_{t-1}^\pi$ using the node selection strategy π_N
$\mathcal{O}_t^\pi \leftarrow \mathcal{O}_{t-1}^\pi \setminus \{\zeta_t\}$
Leaf assessment:
If ζ_t is either MILP-feasible or LP-infeasible or LP-feasible with objective $\tilde{\beta}_t \geqslant \beta_t^\pi$:
$\mathcal{L}_t^\pi \leftarrow \mathcal{L}_{t-1}^\pi \cup \{\zeta_t\}$; $\mathcal{N}_t^\pi \leftarrow \mathcal{N}_{t-1}^\pi$
If ζ_t is MILP-feasible with objective $\tilde{\beta}_t < \beta_t^\pi$ and value x_t:
$\beta_t^\pi \leftarrow \tilde{\beta}_t$; $x \leftarrow x_t$
Branching:
Else (if ζ_t is not a leaf):
$\mathcal{N}_t^\pi \leftarrow \mathcal{N}_{t-1}^\pi \cup \{\zeta_t\}$; $\mathcal{L}_t^\pi \leftarrow \mathcal{L}_{t-1}^\pi$
Select a binary ariable j_t using the branching strategy π_V and update
$\mathcal{O}_t^\pi \leftarrow \mathcal{O}_{t-1}^\pi \cup \{D_0^\pi(\zeta_t, j_t), D_1^\pi(\zeta_t, j_t)\}$
end while

Output:
x ; $\beta^* \equiv \beta_t^\pi$; $\mathcal{N}^\pi \equiv \mathcal{N}_t^\pi$; $\mathcal{L}^\pi \equiv \mathcal{L}_t^\pi$; $\mathcal{T}^\pi \equiv \mathcal{N}^\pi \cup \mathcal{L}^\pi$

Chapter 4

Learning a Dynamic Branching Policy

Content

This chapter is dedicated to the learning of the branching strategy in Algorithm 4. For the sake of simplicity, we denote Π the set of all possible branching strategies and the letter π will refer to a branching policy in the following.

Branching governs the creation of recursive disjunctive sets, at the core of the divide and conquer B&B methodology. Various studies suggest that it plays a crucial, if not the most important, role in the building of B&B trees [49, 63], which explains our interest in learning such strategies. In that matter, it is worth noting that the majority of the approaches involving the learning of B&B strategies is dedicated to branching (see Section 2.3.1). To illustrate the importance of branching, Figure 4.1 shows an histogram of tree sizes generated by random branching decisions on a given instance and compares it to the size of a tree produced by `CPLEX`.

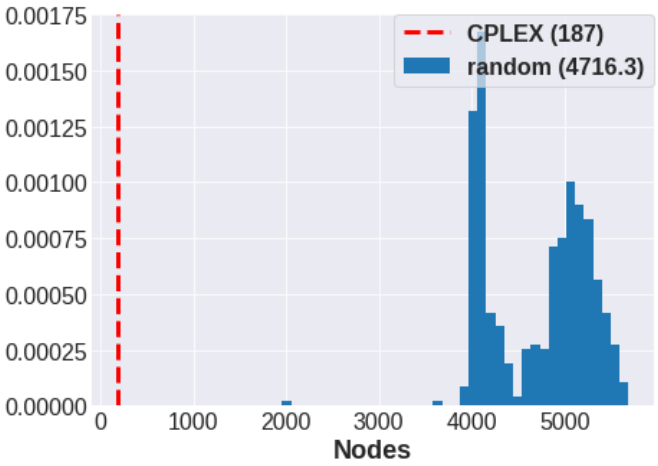


Figure 4.1: Histogram of tree sizes generated by 500 random branching policies on a `micro_bal_T12` instance. Node selection is performed by an identical heuristic in both cases to allow a fair comparison, and values in parentheses are the mean tree size for the random policies and the tree size for `CPLEX`.

Throughout this chapter, the focus will be put on discovering oracle branching strategies, *i.e.* strategies which produce minimal trees for a given node selection strategy. To do so, we have no choice but to take into account their dynamic nature. A simple way of understanding the importance of dynamics is to consider the depth of a single dive towards a given solution x for different branching strategies in Algorithm 4. In this setting, the node selection strategy is to select the deepest node which corresponds to solution x, *i.e.* at iteration t selecting the node $D_k(\zeta_{t-1}, j_{t-1})$ where j_{t-1} is the branching decision at the previous iteration and $k = x_{j_{t-1}}$. Let us investigate on the example (4.1)

the problem of finding the sequence of branching decisions which minimizes the length of such dive towards the solution $(y = 1_{\mathbb{R}^n}, z = 0_{\mathbb{R}^n})$. Figure 4.2 illustrates all the possible paths through branching decisions towards $(y = 1_{\mathbb{R}^n}, z = 0_{\mathbb{R}^n})$ for the case $n = 2$.

$$\begin{cases} \max\limits_{y,z} & \sum\limits_{i=1}^{n} (y_i + z_i) \\ s.t. & y_i + z_i \leqslant 1.2 \ \forall \ i \in \{1, \dots, n\} \\ & y \in \{0,1\}^n, \ z \in \{0,1\}^n \end{cases} \tag{4.1}$$

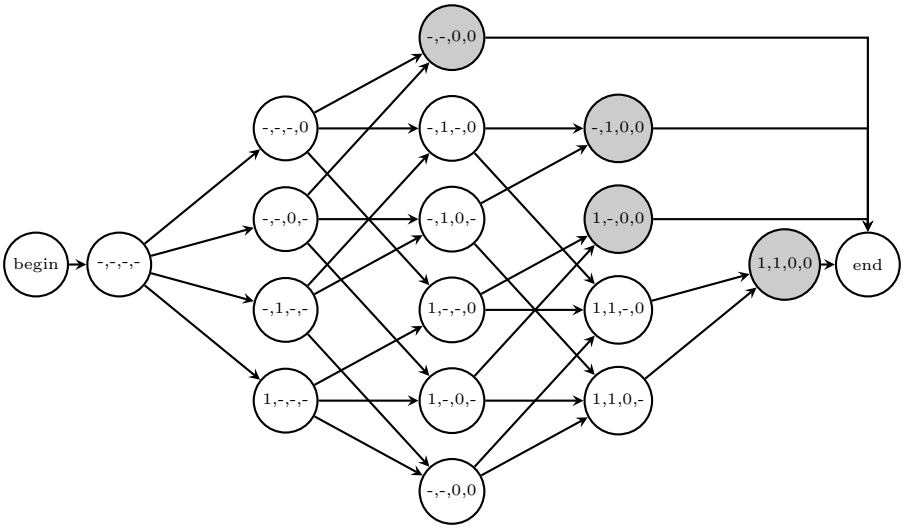


Figure 4.2: This graph represents all the possible branching decisions for Problem (4.1) with $n = 2$ under the specified node selection. The variable values are fixed by branching and node selection at any iteration of Algorithm 4, and each path is associated to a B&B dive towards $(1, 1, 0, 0)$.

Writing $\mathbb{1}_{t,y}$ the indicator of branching on a y variable in iteration t of Algorithm 4, the dive length T towards $(y = 1_{\mathbb{R}^n}, z = 0_{\mathbb{R}^n})$ satisfies the inequality $T \geqslant n + \sum_{t=1}^{T} \mathbb{1}_{t,y}$, equality holding when the branching strategy never selects a variable already fixed by previous branching decisions. Here, an optimal branching strategy is characterized by always branching on a not-fixed z variable. Indeed, fixing y variables do not set z variables to their optimal value (neither to a feasible value, the constraint $y_i = 1$ pushing the LP value of z_i to 0.2), whereas fixing $z_i = 0$ pushes y_i to 1. It illustrates the fact that, even for the simple case of a diving strategy, the quality of a branching decision depends on every other branching decisions in the trajectory. When considering complete B&B trees, things become more complex and may depend on any other decisions in any of the other branches of the tree.

In order to capture this dynamic nature, we first generalize in Section 4.1 classic existing branching heuristics, which intent to produce long-term strategies by short-term decisions. To overcome the computational complexity of the proposed alternative, we propose a novel reinforcement learning framework adapted to the B&B environment to learn these strategies. Theoretical aspects and limitations of RL in this setting are studied and illustrated. Section 4.2 considers and compares two special cases, where a simple cost model is used and is proven to be optimal in some sense. Special care is given to the use of a discount factor, and a biased but more robust cost model is proposed. In Section 4.3, some variations around the presented methodology are given.

4.1 Learning dynamic heuristic strategies

4.1.1 Positioning

As mentioned in Chapter 2, branching strategies are heuristically designed in off-the-shelf solvers. Despite their huge impact on the B&B procedure and the considerable amount of research dedicated to design such policies, we still struggle to characterize good branching strategies. One of the main difficulties when building strategies in B&B is their dynamic nature, as illustrated above. Especially, the impact of a branching decision often depends on remote choices, either made earlier or later on. In this section, we hence aim at incorporating dynamic considerations into heuristic branching.

One of the most famous branching strategies is SB (see Section 2.3.1). Even if the SB rule is based on a heuristic score, it has shown a great ability to produce small B&B trees. The main source of its efficiency is the fact that this heuristic score is derived on potential child nodes. In other words, SB uses some kind of knowledge about the immediate future outcomes ahead of the current branching decision. This knowledge comes with a high computational cost, since it requires to compute all (or some part of) these potential outcomes. To limit this cost, strategies with no look-ahead are often preferred in practice, such as Pseudo-Cost or Most Fractional Branching. In the following, we address the challenge of learning branching policies with look-ahead, like SB. Such an approach has been studied in different papers (see for instance [63, 68]), where SB is learnt by imitation (see Section 2.1.2 for an introduction to IL). However, rather than focusing only on SB or heuristics with no look-ahead, we learn more long-term policies by introducing the notion of h-ahead branching heuristics. We adopt

an RL paradigm in order to learn the dynamics of the branching policy, using some information about future consequences of already made choices.

4.1.2 Markov Decision Process formulation

Before diving into the question of designing and learning strategies, let us formalize a branching policy using an MDP formulation. We define such episodic MDP when solving an instance p as a tuple $< \mathcal{S}, \mathcal{A}, T, c, \gamma >$, where the environment is set by Algorithm 4 and which consists of:

a set of states $\mathcal{S}$: a state $s_t \in \mathcal{S}$ is a tuple $(p, t, \mathcal{H}_t)$ where t is an iteration of Algorithm 4 and $\mathcal{H}_t$ is the history of any decisions or observations made so far, including the node selection performed at iteration t. A B&B node $\zeta(s_t)$ is associated to a state s_t and corresponds to the node visited at iteration t. Note here that the mapping from states to B&B nodes is not an injection: many states can be associated to a single B&B node. Conversely, a unique B&B node is associated to a state.

Note that $\mathcal{S}$ is finite provided that one cannot branch on an already fixed variable: indeed in that case, the number of disjunctive sets, primal bounds and visiting orders is finite.

a set of actions $\mathcal{A} \equiv \mathcal{J}$. At each state, except for states corresponding to leaf nodes, a branching decision has to be made amongst the set of binary variables at the current node.

a transition function T assumed to be deterministic here, either trajectory-based or tree-based according to Definition 4.1.1 and 4.1.2 provided in what follows. Tree-based transitions are introduced to take into account the structure of the B&B environment and its dynamics. Figure 4.3 illustrates the differences between these two transition functions.

Note that transitions are considered deterministic but unknown. A sufficient condition for this hypothesis to be verified is Hypothesis 4.1.1, see below. If it was not for the deterministic hypothesis, these transition functions would nevertheless still be markovian by construction as $\{j_t\} \cup \mathcal{H}_t \subseteq \mathcal{H}_{t'}$ for any $t < t'$.

Note that one should add a fictitious action for states associated to leaf nodes to properly define transitions (and cost functions). We chose not to do so as it would only make the notations more cumbersome.

a bounded cost function $c : \mathcal{S} \times \mathcal{J} \to \mathbb{R}$ which depends on the objective set for the strategy to be learnt.

a discount factor $\gamma \in [0, 1]$.

This finite MDP is episodic as one can bound the number of iterations in Algorithm 4 by $2^{n+1} - 1$ provided that the branching policy does not select the same variable twice in a single branch of the tree.

Hypothesis 4.1.1. *Deterministic node selection hypothesis*

The node selection policy in Algorithm 4 is deterministic: no matter the history, the node selection policy always selects the same node in a given set of open nodes.

Definition 4.1.1. *Trajectory-based transition*

We call trajectory-based transition a deterministic function of the form $T : \mathcal{S} \times \mathcal{J} \to \mathcal{S}$; $T(s_t, a_t) = s_{t+1}$ *which gives the next visited state from a state-action pair, with* $\zeta(s_t) = \zeta_t$ *and* $\zeta(T(s_t, a_t)) = \zeta_{t+1}$ *where* t *corresponds to an iteration of Algorithm 4. Such transition corresponds to the usual type of time-indexed transition functions considered in classic RL tasks.*

Definition 4.1.2. *Tree-based transition*

We call tree-based transition a deterministic transition of the form $T^\pi : \mathcal{S} \times \mathcal{J} \to \mathcal{S} \times \mathcal{S}$; $T^\pi(s_t, a_t) = (D_0^\pi(s_t, a_t), D_1^\pi(s_t, a_t))$ *where* t *corresponds to an iteration of Algorithm 4. The notation* $D_k^\pi(s, j)$ *refers to the state associated to the node* $D_k(\zeta(s), j)$ *when following policy* π*. Here, the transition function provides two different states, their visiting time depending on both the branching and the node selection strategy.*

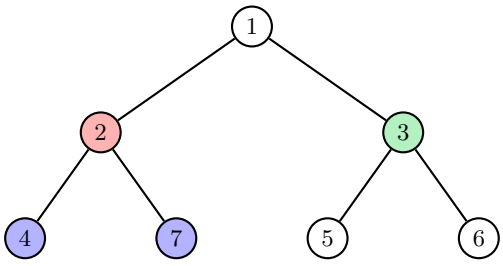


Figure 4.3: Illustration of the difference between trajectory-based and tree-based transitions. The represented tree is a B&B tree, where the number associated to each node is its visiting time. A tree-based transition function maps (s_2, a_2) to (s_4, s_7), whereas a classic trajectory-based transition maps (s_2, a_2) to s_3.

Remark 1. Similar notations, different meanings

At this point, it is important to make the distinction between the notations $D_k(\zeta, j)$ and $D_k^\pi(s, j)$ with ζ the B&B node associated to state s. The former is a B&B node, whereas the latter is a state. As the state comprises any information obtained before processing its associated node, we may have $D_k^{\pi_1}(s, j) \neq D_k^{\pi_2}(s, j)$ for two different strategies π_1 and π_2, even if their associated B&B node is $D_k(\zeta, j)$.

4.1.3 h-ahead branching heuristic

Using the introduced formalism, we define *h-ahead branching heuristics*, a class of decision functions which makes it possible to generalize any classic branching heuristics by incorporating future knowledge. These heuristics can also be perceived as the generalization of the look-ahead strategies presented in [93] and [94], where the authors propose either to apply the SB rule on both child and grandchild nodes or to hybridize it with a look-ahead score based on the entropy in potential child nodes.

Definition 4.1.3. *h-ahead branching heuristic*
Let us denote $\mathcal{D}^{\pi,h}(s, j)$ the set of states encountered in the subtree of depth $h > 0$ rooted in s when branching on variable j and following branching strategy $\pi \in \Pi$. This set corresponds to states reached from s in less than $h + 1$ tree-based transitions when following policy π. We define an h-ahead cost $\nu_h(s, j, \pi)$ for variable j at state s as

$$\nu_h(s, j, \pi) = \nu^0(s, j) + \nu^1\left(\mathcal{D}^{\pi,h}(s, j)\right) \tag{4.2}$$

with $\nu^0 : \mathcal{S} \times \mathcal{J} \to \mathbb{R}$ and $\nu^1 : \mathcal{P}(\mathcal{S}) \to \mathbb{R}$ two score functions.
An h-ahead branching heuristic $\pi_h : \mathcal{S} \to \mathcal{J}$ is then defined as the branching decision minimizing the potential h-ahead cost:

$$\pi_h(s) = \arg\min_{j \in \mathcal{J}} \left(\min_{\pi \in \Pi} \nu_h(s, j, \pi)\right) \tag{4.3}$$

Note that, in Definition 4.1.3, the future information comes from the descendant states $\mathcal{D}^{\pi,h}(s, j)$. Even though information from next iterations could have been used instead (thus using trajectory-

based transitions), this definition with tree-based transitions still remains quite general and embraces well-known heuristics. For example, Most Fractional is a 0-ahead branching heuristic with $\nu^0(s,j) = \left|x_j^s - 0.5\right|$ and $\nu^1 = 0$, where x^s is the LP solution associated to s. Likewise, Strong Branching is a 1-ahead branching heuristic with $\nu^0 = 0$ and $\nu^1(D_0(s,j), D_1(s,j)) = \text{score}(z_{D_0(s,j)} - z_s, z_{D_1(s,j)} - z_s)$, where z_s refers to the LP value at $\zeta(s)$. Note here that, in these two cases, these static heuristics do not need to perform the optimization over the set of branching policies Π in Equation (4.3).

The interest of an h-ahead heuristic is straightforward. Rather than simply branching on a variable with a high heuristic score at current node, it seems appealing to select variables that also lead to high potential child nodes.

Unfortunately, following an h-ahead branching heuristic comes at a prohibiting computational cost. As an illustration, computing such heuristic at the root node by exploring all the available branching policies in Equation (4.3) requires in the worst case to build $\prod_{k=0}^{h-1}(n-k)^{2^k}$ subtrees of depth h.

As introduced in Section 2.3.1, ML can be leveraged to learn off-line an expensive branching strategy. However, such brute force approach can hardly be transposed when considering h-ahead heuristics, as it turns out to be way too computationally expensive even for low values of h. Indeed, as displayed in Algorithm 5, it would require to run an h-ahead branching policy on training instances, collect state-action pairs and the associated h-ahead heuristic outcomes, then learn a mapping $f_\theta : \mathcal{S} \to \mathcal{J}$ between them (it is a classification task as defined in Section 2.1.1). Running an exact h-ahead branching policy then requires to suffer the full cost of exploring over Π in Equation (4.3).

Besides, a subtle point should be highlighted here. When considering h-ahead heuristics as defined in Definition 4.1.3, dynamics are only partially taken into account, as following an h-ahead heuristic does not ensure that consecutive choices are coherent with each other. More specifically, an action is selected at some state to obtain the minimal cost provided that the policy yielding $\min_{\pi \in \Pi} \nu_h(s, j, \pi)$ is followed in descendant states in Equation (4.3). However, this is not guaranteed. In other words, one needs to ensure consistency of h-ahead heuristics through transitions before minimizing the cost over the set of branching strategies Π. A similar remark can be done regarding the fact that vanilla SL methods as presented in Algorithm 5 learn on the state distribution produced by the true heuristic and thus not on the distribution produced by the learnt one. Although using IL methods such as dataset aggregation (see Section 2.1.2) instead of vanilla SL would solve this issue, it would also heavily suffer

from the computational cost previously mentioned. These considerations motivate our choice to use reinforcement learning, where the two minimization steps in Equation (4.3) are somehow merged into one.

Algorithm 5 Brute force training algorithm for h-ahead heuristic branching

Input:
an h-ahead cost function ν_h and its associated heuristic π_h
Initialization:
An empty training dataset $D \leftarrow \varnothing$
Procedure:
for every p in the training instances dataset **do**:
Solve p using π_h and let $\mathcal{T}^{\pi_h}$ be the visited non-leaf states
Collect dataset of visited states and maximal scores for each variable:
$D \leftarrow D \cup \{s, \pi_h(s)\}_{s \in \mathcal{T}^{\pi_h}}$
end for
Learn a mapping function f_θ using data in D using SL (see Section 2.1.1)
Output:
f_θ

4.1.4 Reinforcement learning with tree-based transition

As introduced in Section 2.2, reinforcement learning is a framework designed to solve dynamic control problems. Rather than fully exploring the set Π at each state and then learning from the entire dataset as required by SL, we learn in an iterative manner the sum of future costs associated to an evolving policy. It allows to guide the aforementioned exploration towards "interesting" parts of Π by learning from experiments. In the following, we consider tree-based transitions, which follow the B&B tree structure, to keep consistency with the nature of branching heuristics.

Value functions under tree-based transitions

Formally, let us write $c(s, j)$ the cost associated to a state-action pair and consider branching policies of the form

$$\pi : \begin{cases} \mathcal{S} \to \mathcal{J} \\ s \mapsto \pi(s) = \underset{j \in \mathcal{J}}{\arg\min}\, Q^\pi(s, j) \end{cases} \tag{4.4}$$

Q^π is the Q-value function associated to the branching policy π, defined as the discounted sum of costs occurring in state-action pairs visited by π through tree-based transitions from the state-action

(s, j):

$$Q^\pi : \begin{cases} \mathcal{S} \times \mathcal{J} \to \mathbb{R} \\ s, j \mapsto Q^\pi(s, j) = c(s, j) + \sum_{s' \in \mathcal{D}^{\pi,n}(s,j)} \gamma^{d(s')-d(s)} c(s', \pi(s')) \end{cases} \tag{4.5}$$

where $d(s)$ is the depth of the node $\zeta(s)$ and $\gamma \in [0, 1]$ is a discount factor. In this setting, the discount factor gives more weight to the closest descendant states, *i.e.* the ones with low depths. Note here the fundamental difference with Q-functions under standard trajectory-based transitions, which are defined as the discounted sum of *future* costs.

We see that Q^π can be perceived as an n-ahead score, and the only difference between an h-ahead branching heuristic and the branching policy π defined by (4.4) is the optimization over Π – see (4.3). Instead, the branching policy is now parametrized by itself.

The value function V^π associated to Q^π is defined as

$$V^\pi : \begin{cases} \mathcal{S} \to \mathbb{R} \\ s \mapsto V^\pi(s) = Q^\pi(s, \pi(s)) = c(s, \pi(s)) + \sum_{s' \in \mathcal{D}^{\pi,n}(s,\pi(s))} \gamma^{d(s')-d(s)} c(s', \pi(s')) \end{cases} \tag{4.6}$$

and, due to the tree structure of our environment, these two functions satisfy the Bellman equations

$$V^\pi(s) = c\left(s, \pi(s)\right) + \gamma\Big[V^\pi\left(D_0^\pi(s, \pi(s))\right) + V^\pi\left(D_1^\pi(s, \pi(s))\right)\Big] \tag{4.7}$$

$$Q^\pi(s, j) = c\left(s, j\right) + \gamma\Big[Q^\pi\left(D_0^\pi(s, j), \pi(s)\right) + Q^\pi\left(D_1^\pi(s, j), \pi(s)\right)\Big] \tag{4.8}$$

where $D_0^\pi(s, \pi(s))$ and $D_1^\pi(s, \pi(s))$ are the child states of s when following strategy π. They are both indexed by π as they are visited at iterations which depend on the branching policy, and consequently the information they contain may differ even for two strategies π_1, π_2 selecting the same action at s, *i.e.* such that $\pi_1(s) = \pi_2(s)$ (see Remark 1).

Finding an optimal policy

As introduced in Section 2.2, the objective of reinforcement learning is then to find an *optimal policy* with respect to these value functions, *i.e.* π^* such that, for any $s \in \mathcal{S}$, we have

$$V^{\pi^*}(s) = V^*(s) \equiv \min_{\pi \in \Pi} V^\pi(s) \tag{4.9}$$

This objective is the reason why we adopt the RL paradigm to learn look-ahead branching strategies rather than applying SL to h-ahead branching heuristics. To derive the score associated to a single

branching decision in a h-ahead heuristic, one needs to search over the entire set Π, and this exhaustive search is to be repeated for each individual state. In RL, an optimal policy is defined on the entire state space, allowing the search to be done once at a higher level. This allows the learnt policy to be dynamically coherent, contrary to h-ahead heuristics where the optimizations are performed independently at each state.

Remark 2. An optimal policy for a heuristic cost model does not yield an oracle strategy

At this point, is is worth noting that finding such optimal policy *a priori* does not bring any guarantee on finding an oracle strategy. Although this approach allows to take into account branching dynamics due to the discounted sum in Equation (4.5), considering a cost c corresponding to a "classic" heuristic such as SB do not ensure to minimize the tree size. We introduce in Section 4.2 a particular setting which offers such a guarantee, thus reconciling the notions of optimal policy and oracle strategy.

In classic RL, the optimal policy π^* is derived from the optimal value function V^*, which is the unique fixed point of the dynamic programming operator (Theorem 2.2.1). However, as stated in Proposition 4.1.1, the choice of tree-based transitions prevents us from deriving an optimal policy in the same way.

Proposition 4.1.1. *Without additional assumptions, an optimal policy π^* (in the sense of Equation (4.9)) cannot be defined by setting $\pi^*(s) = \arg\min_{j\in\mathcal{J}} c\left(s,j\right) + \gamma\Big[V^*\left(D_0^*(s,j)\right) + V^*\left(D_1^*(s,j)\right)\Big]$.*

Proof. *We have*

$$
\begin{aligned}
&V^{\pi^*}(s) = V^*(s)\\
&\iff \pi^*(s) = \arg\min_{j\in\mathcal{J}} \left\{\min_{\pi\in\Pi} \left\{c(s,j) + \gamma\Big[V^{\pi}\left(D_0^{\pi}(s,j)\right) + V^{\pi}\left(D_1^{\pi}(s,j)\right)\Big]\right\}\right\}\\
&\iff \pi^*(s) = \arg\min_{j\in\mathcal{J}} \left\{c(s,j) + \gamma \min_{\pi\in\Pi} \left\{V^{\pi}\left(D_0^{\pi}(s,j)\right) + V^{\pi}\left(D_1^{\pi}(s,j)\right)\right\}\right\}
\end{aligned}
$$

However, we give in the proof of Proposition 4.2.2, page 106, a counter-example where

$$
\begin{aligned}
&\arg\min_{j\in\mathcal{J}} \left\{c(s,j) + \gamma \min_{\pi\in\Pi} \left\{V^{\pi}\left(D_0^{\pi}(s,j)\right) + V^{\pi}\left(D_1^{\pi}(s,j)\right)\right\}\right\}\\
&\neq \arg\min_{j\in\mathcal{J}} \left\{c(s,j) + \gamma \min_{\pi\in\Pi} \left\{V^{\pi}\left(D_0^{\pi}(s,j)\right)\right\} + \gamma \min_{\pi\in\Pi} \left\{V^{\pi}\left(D_1^{\pi}(s,j)\right)\right\}\right\}
\end{aligned}
$$

Thus the result. □

Proposition 4.1.1 states that knowing the optimal value function V^* is not sufficient to derive an optimal policy π^* as the equality

$$V^*(s) = \min_{j \in \mathcal{J}} c(s,j) + \gamma\big[V^*(D_0^*(s,j)) + V^*(D_1^*(s,j))\big]$$

is not satisfied in this setting. Note that considering trajectory-based transitions would not induce such inconsistency as it allows to recover the classic RL framework.

Finding a surrogate policy

Since the cost model c is heuristic, an optimal policy π^* would probably not be an oracle strategy anyway. Then, looking for π^* is not a requirement and we will instead settle for seeking the surrogate policy $\tilde{\pi}(s) = \arg\min_{j \in \mathcal{J}} c(s,j) + \gamma\Big[V^{\sim}(D_0^{\sim}(s,j)) + V^{\sim}(D_1^{\sim}(s,j))\Big]$ with $V^{\sim}$ the solution of the dynamic programming equation

$$V^{\sim}(s) = \min_{j \in \mathcal{J}} c(s,j) + \gamma\Big[V^{\sim}(D_0^{\sim}(s,j)) + V^{\sim}(D_1^{\sim}(s,j))\Big] \tag{4.10}$$

To do so, we first show that $V^{\sim}$ can be derived exactly by slightly modifying the classic theoretical dynamic programming procedure introduced by Theorem 2.2.1, page 47. This is stated in Theorem 4.1.1 using *tree dynamic programming operators* introduced by Definition 4.1.4. The Lemma 4.1.1 is necessary for deriving the proof.

Definition 4.1.4. *Tree dynamic programming operator*
We call $\tilde{\mathcal{B}}$ a tree dynamic programming operator any operator over value functions defined as $\tilde{\mathcal{B}}V(s) = \min_{j \in \mathcal{J}} c(s,j) + \gamma\Big[V(D_0(s,j)) + V(D_1(s,j))\Big]$ where $D_0(s,j)$ and $D_1(s,j)$ are two arbitrary states among the states associated to child nodes of $\zeta(s)$ when branching on variable j. These two states may even depend on the value function V.

Lemma 4.1.1. *Any tree dynamic programming operator $\tilde{\mathcal{B}}$ is a contraction for the L_∞ norm as soon as $\gamma \in [0, 0.5)$.*

Proof of Lemma 4.1.1. *Let us show that $||\tilde{\mathcal{B}}V_1 - \tilde{\mathcal{B}}V_2||_\infty \leqslant 2\gamma||V_1 - V_2||_\infty$ for two arbitrary functions V_1 and V_2.*
Let us denote $\mathcal{S}(s,j) = \big\{s^j \in \mathcal{S},\ \zeta(s^j) \in \mathcal{D}(\zeta(s), j)\big\}$ the set of states associated to a child node of $\zeta(s)$ for a given branching variable $j \in \mathcal{J}$. For any $s \in \mathcal{S}$, we write $s^j_{1,0}, s^j_{1,1}$ (resp. $s^j_{2,0}, s^j_{2,1}$) the arbitrary

states in $\mathcal{S}(s,j)$ involved in the derivation of $\tilde{\mathcal{B}}V_1(s)$ (resp. $\tilde{\mathcal{B}}V_2(s)$). We have

$$\begin{aligned}
\left|\tilde{\mathcal{B}}V_1(s) - \tilde{\mathcal{B}}V_2(s)\right| &= \left| \min_{j\in\mathcal{J}} \left\{ c\left(s,j\right) + \gamma\Big[V_1\left(s_{1,0}^j\right) + V_1\left(s_{1,1}^j\right)\Big]\right\} \right. \\
&\qquad \left. - \min_{j\in\mathcal{J}} \left\{ c\left(s,j\right) + \gamma\Big[V_2\left(s_{2,0}^j\right) + V_2\left(s_{2,1}^j\right)\Big]\right\} \right| \\
&\leqslant \max_{j\in\mathcal{J}} \left| c\left(s,j\right) + \gamma\Big[V_1\left(s_{1,0}^j\right) + V_1\left(s_{1,1}^j\right)\Big] - c\left(s,j\right) - \gamma\Big[V_2\left(s_{2,0}^j\right) + V_2\left(s_{2,1}^j\right)\Big]\right| \\
&\leqslant \gamma \max_{j\in\mathcal{J}} \left| V_1\left(s_{1,0}^j\right) + V_1\left(s_{1,1}^j\right) - V_2\left(s_{2,0}^j\right) - V_2\left(s_{2,1}^j\right)\right| \\
&\leqslant 2\gamma \max_{\substack{j\in\mathcal{J} \\ s_1^j\in\left\{s_{1,0}^j, s_{1,1}^j\right\} \\ s_2^j\in\left\{s_{2,0}^j, s_{2,1}^j\right\}}} \left| V_1\left(s_1^j\right) - V_2\left(s_2^j\right)\right| \\
&\leqslant 2\gamma ||V_1 - V_2||_\infty
\end{aligned}$$

□

Theorem 4.1.1. *$V^\sim$ is the unique solution of Equation (4.10) and can be found as $V^\sim = \lim_{k\to+\infty} \tilde{\mathcal{B}}^k V_0$ with V_0 any value function, $\tilde{\mathcal{B}}$ any tree dynamic programming operator and $\gamma \in [0, 0.5)$.*

Proof of Theorem 4.1.1. *The proof follows that of Theorem 2.2.1, with some minor changes.*

By definition, $V^\sim$ is a solution of Equation (4.10). The uniqueness is induced by the contraction property of operator $\tilde{\mathcal{B}}$.

Let us now consider the sequence $(V_k)_{k\in\mathbb{N}}$ defined by $V_{k+1} = \tilde{\mathcal{B}}V_k$ with V_0 any value function, perceived here as a vector in $\mathbb{R}^{|\mathcal{S}|}$. Similarly as in the proof of Lemma 4.1.1, $s_{k,0}^j$ and $s_{k,1}^j$ refer to the states involved in the derivation of $\tilde{\mathcal{B}}V_k(s)$.

The sequence (V_k) is bounded since c is bounded and $\gamma \in [0, 0.5)$ gives:

$$\begin{aligned}
||V_k||_\infty &= \max_{s\in\mathcal{S}} \left| \min_{j\in\mathcal{J}} c(s,j) + \gamma\Big[V_{k-1}(s_{k,0}^j) + V_{k-1}(s_{k,1}^j)\Big]\right| \\
&\leqslant ||c||_\infty + 2\gamma||V_{k-1}||_\infty \\
&\leqslant (2\gamma)^k||V_0||_\infty + ||c||_\infty \sum_{i=0}^{k-1} (2\gamma)^i \\
&\leqslant ||V_0||_\infty + \frac{||c||_\infty}{1-2\gamma}
\end{aligned}$$

Besides, the sequence (V_k) is Cauchy for the L_∞ norm as, for any $k \geqslant p$:

$$||V_k - V_p||_\infty \leqslant 2\gamma||V_{k-1} - V_{p-1}||_\infty \leqslant \cdots \leqslant (2\gamma)^p||V_{k-p} - V_0||_\infty \underset{k,p\to+\infty}{\longrightarrow} 0$$

since (V_k) is bounded and $\tilde{\mathcal{B}}$ is a contraction for $\gamma \in [0, 0.5)$.

As the space $\mathbb{R}^{|\mathcal{S}|}$ equipped with the L_∞ norm is a Banach space, the sequence (V_k) converges. Let us write V_∞ its limit. By passage to the limit in $V_{k+1} = \tilde{\mathcal{B}}V_k$ we have $V_\infty = \tilde{\mathcal{B}}V_\infty$. Thus, by uniqueness

of the fixed point for $\tilde{\mathcal{B}}$*, we have* $\lim_{k\to+\infty} V_k = \lim_{k\to+\infty} (\tilde{\mathcal{B}})^k V_0 = V^{\sim}$. □

Note here the condition $\gamma \in [0, 0.5)$, necessary for obtaining the contraction property of the tree dynamic programming operator and for the dynamic programming sequence to be Cauchy. This theoretical remark will be echoed later on regarding practical considerations.

Learning a surrogate policy

Unfortunately, the complexity of one single step of the algorithm described in Theorem 4.1.1 is $\mathcal{O}(2\,|\mathcal{S}|\,|\mathcal{J}|)$, which prevents us from using it in practice. Thus, we turn to approximation procedures and more specifically Approximate Q-learning (see Section 2.2.3).
The Q-function defined in Equation (4.5) is approximated by a neural network $\hat{Q}(.,.;\theta)$ parametrized by a weight vector θ, and following the DQN method [28], the iterative fitting procedure would be governed by steps towards reducing the empirical equivalent of the theoretical DQN loss function

$$L_i^{DQN}(\theta_i) = \mathbb{E}_{s,j\sim\Delta^i}\Bigg[\Bigg(c(s,j) + \gamma\Big[\min_{j'} \hat{Q}(D_0^{\pi_{\theta_i^-}}(s,j), j'; \theta_i^-) + \min_{j'} \hat{Q}(D_1^{\pi_{\theta_i^-}}(s,j), j'; \theta_i^-)\Big] - \hat{Q}(s,j;\theta_i)\Bigg)^2\Bigg] \tag{4.11}$$

at step i, where θ_i^- is some fixed parameter from previous iterations – see Section 2.2.3, page 49, for a direct transposition. Δ^i refers here to the probability distribution of state-action pairs when following previous policies, randomness being generated by the random variable selecting the instance. The target $c(s,j) + \gamma\Big[\min_{j'} \hat{Q}(D_0^{\pi_{\theta_i^-}}(s,j), j'; \theta_i^-) + \min_{j'} \hat{Q}(D_1^{\pi_{\theta_i^-}}(s,j), j'; \theta_i^-)\Big]$ results from the Q-value counterpart of Bellman Equation (4.10), and is required in a stochastic setting as an approximation for $Q^{\pi_{\theta_i^-}}(s,j)$, which is usually an expectation over a possibly infinite process – note that a similar parallel with Sarsa can be made here, see Section 2.2.3. However, in our deterministic and episodic setting, this term is observable and we can actually consider the loss function

$$L_i(\theta_i) = \mathbb{E}_{s,j\sim\Delta^i}\left[\left(Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j;\theta_i)\right)^2\right] \tag{4.12}$$

In this setting, the agent's policy is to select the action with the minimal predicted cost:

$$\pi_\theta(s) = \arg\min_{j\in\mathcal{J}} \hat{Q}(s,j;\theta) \tag{4.13}$$

The base training procedure is described in Algorithm 6, with N the number of training episodes (or iterations). In practice, the Q-network $\hat{Q}(.,.;\theta)$ is built with $|\mathcal{J}|$ outputs, each output being associated

to a Q-value function estimation $\hat{Q}(., j; \theta)$. During training, each output receives gradient from errors on visited samples where the corresponding action has been selected.

Algorithm 6 Training Algorithm: RL for Variable Selection

Initialization:
Randomly initialize θ_0
Procedure:
for $i = 1$ to N **do**:
Draw randomly an instance p from the training dataset
Solve p using $\pi_{\theta_{i-1}}$ with ε-greedy exploration
Collect dataset $\{(s, j)\}$ of visited states and observed Q-values from $\mathcal{T}^{\pi_{\theta_{i-1}}}(p)$ in a buffer B
Update θ_{i-1} to θ_i following the gradient derived from Equation (4.12) using data from B
end for
Output:
Final parameter θ_N

4.1.5 Experiments

To evaluate the proposed methodology, we test it using three different heuristic cost functions. First, we consider a SB-like heuristic cost by taking

$$c(s, j) = \begin{cases} 0 & \text{if } \zeta(s) \text{ is a leaf node} \\ \max\{0.1\ ;\ (1 - \Delta_0)(1 - \Delta_1)\} & \text{otherwise} \end{cases} \qquad \texttt{(SBc)}$$

with $\Delta_i = \min\left\{1\ ; \frac{z_{D_i^\pi(s,j)} - z_s}{|z_s|}\right\}$ and z_s the LP objective associated with state s. Such definition corresponds to a normalized version of SB, turned into a cost to match our minimization objective. Note that we give a strictly positive cost to non-leaf nodes so as to encourage the agent to encounter leaves. Therefore, this cost pushes the agent to branch on variables which produce either a large change of the dual bound or leaf nodes.

Second, we define a heuristic cost based on the Most-Fractional Branching strategy:

$$c(s, j) = \begin{cases} 0 & \text{if } \zeta(s) \text{ is a leaf node} \\ \dfrac{1}{|\mathcal{J}|} \displaystyle\sum_{j' \in \mathcal{J}} \min\left\{x^*(s)_{j'}\ ;\ 1 - x^*(s)_{j'}\right\} & \text{otherwise} \end{cases} \qquad \texttt{(DBc)}$$

where $x^*(s)$ is the LP solution associated with state s. This heuristic encourages the agent to minimize the mean distance of the binary variables to their bounds.

Last, we consider a simpler version of the previous heuristic by taking

$$c(s,j) = \begin{cases} 0 & \text{if } \zeta(s) \text{ is a leaf node} \\ 1 - \dfrac{1}{|\mathcal{J}|} \displaystyle\sum_{j' \in \mathcal{J}} \mathbb{1}\left(\min\left\{x^*(s)_{j'} \ ; \ 1 - x^*(s)_{j'}\right\} = 0\right) & \text{otherwise} \end{cases} \tag{NBc}$$

which tends to minimize the number of fractional variables.

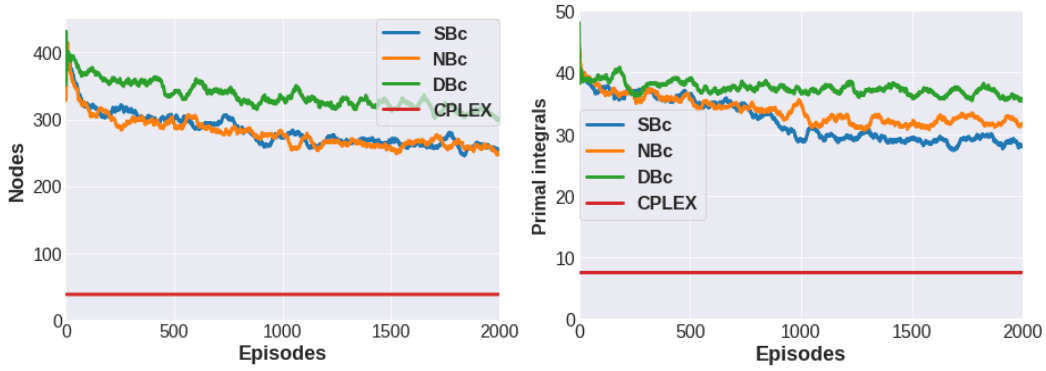
Remark 3. Cost associated to leaf nodes

In the three heuristic costs defined above, a null value is associated to leaf nodes. Not only this is natural as one often seeks to find early leaf nodes, but it also allows these strategies to be more *versatile* than their "blind" counterparts. Let us consider the strong branching heuristic. In SB, which can be considered as a 1-ahead branching heuristic (see Section 4.1.3), the only objective is to make the LP objective vary, with the aim of pruning by bound. Hence, SB is not designed to favour infeasible nodes. When considering `SBc`, nodes with high dual bounds (in the minimization case) are encouraged, as much as infeasible nodes due to their null cost. A similar practice can be found in the look-ahead SB proposed in [93], where the authors take into account the pruning objective in their score.

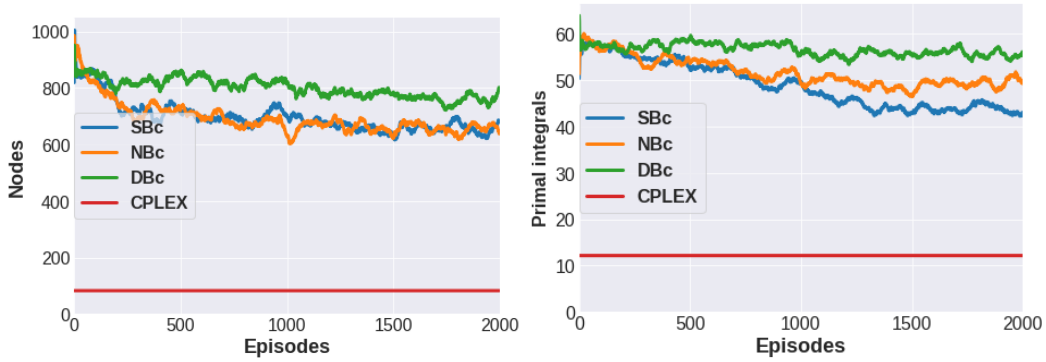
As mentioned in Section 1.2.3, the question of discovering the optimal set of parameters for the chosen approach and problem is not considered in these experiments. We rather compare the different approaches using a unique configuration. The architecture and features use in any of the following experiments are briefly discussed in Section 4.3.2. Unless mentioned otherwise, we set in the remaining experiments the node selection strategy to a DFS strategy where the priority is given to the child node which fixes the branching variable to the bound closest from its LP value. Concretely, if variable j is chosen for branching at some node ζ with LP solution x^*, node $D_0(\zeta, j)$ is visited before $D_1(\zeta, j)$ if $x^*_j < 0.5$ and conversely. This choice is made to fairly compare the branching strategies.

Figure 4.4 displays the training processes for the three heuristic cost models mentioned above. Due to the bad performances observed, we do not show test results for the sake of conciseness. Although the results appear disappointing, we observe that `DBc` is dominated by the two other cost models, both in terms of tree sizes and primal integrals. This observation is not totally surprising. On the one hand, the according Q-function contains little information as the bound distances are aggregated

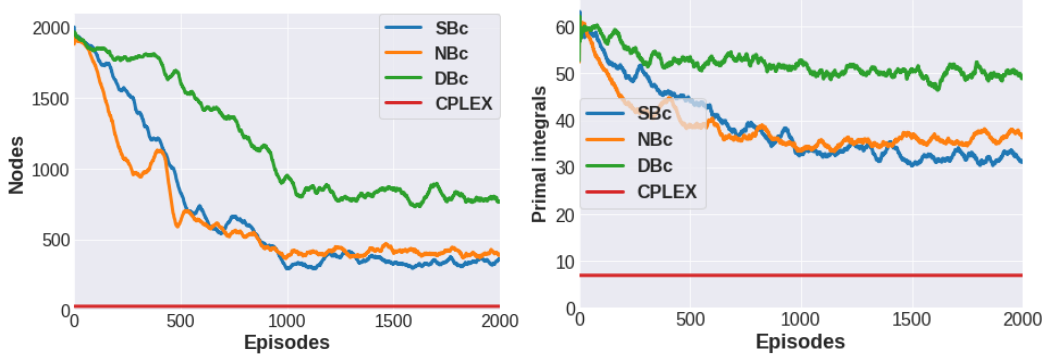
through the mean operator. On the other hand, one can spend a lot of time in near-feasible nodes without being in a position to prune any of them.



(a) Tree sizes – `micro_asym_T6` (b) Primal integral scores – `micro_asym_T6`



(c) Tree sizes – `micro_asym_T8` (d) Primal integral scores – `micro_asym_T8`



(e) Tree sizes – `hydro_fix_1` (f) Primal integral scores – `hydro_fix_1`

Figure 4.4: Training processes: performance of the heuristic cost models under tree-based transitions.

The choice of tree-based transitions appears to be fully justified in the context of these heuristic costs, as illustrated in Figure 4.5. This result is really natural as, except for leaf nodes which may be encouraged in non-descendant nodes by the discovery of a good primal bound, branching decisions only

have an impact on the descendant nodes. Hence, considering the cost associated to non-descendant states when computing the value of some state would have no justification whatsoever.

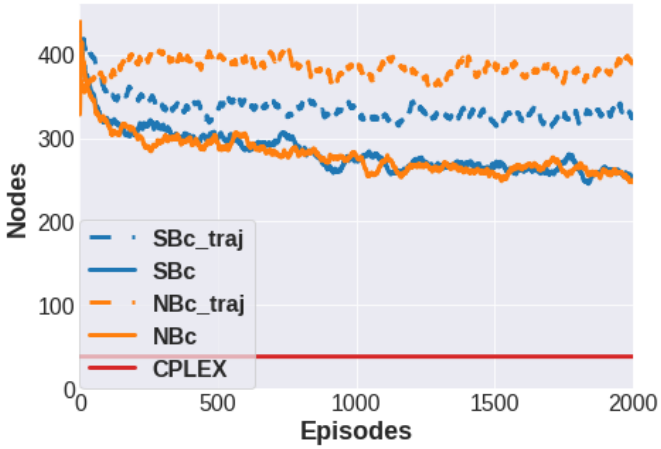
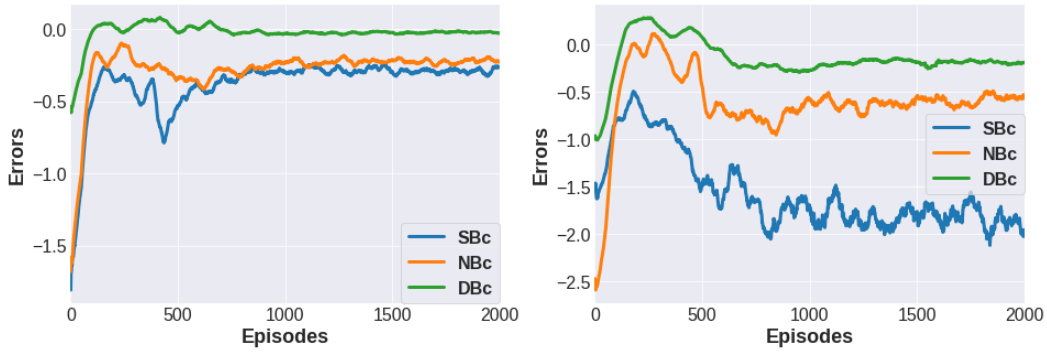


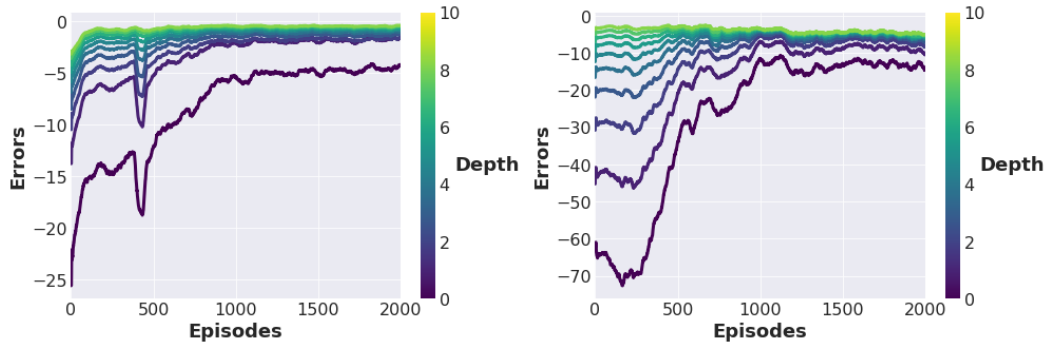
Figure 4.5: Training processes: comparison of trajectory-based transitions (`*_traj`) with tree-based transitions on `micro_asym_T6`.

When observing the errors made by the Q-network (see Figure 4.6), we notice that the agent systematically underestimates the Q-function. However, this bias is not necessarily harmful, as the most important in RL is to properly rank the different actions and not to precisely estimate their value. In this regard, we see when comparing Figure 4.6a and Figure 4.4e that the bias grows in absolute value when the performance improves. We also note that the bias is naturally more important near the root node than in the rest of the tree (see Figure 4.7), as the number of visited samples is lower for these states. To mitigate this effect, more weight has been put in the training loop on the samples near root nodes. However, it does not fully compensates this difference (see Section 4.3, page 132 for some additional comments on this question).



(a) `micro_asym_T6` (b) `hydro_fix_1`

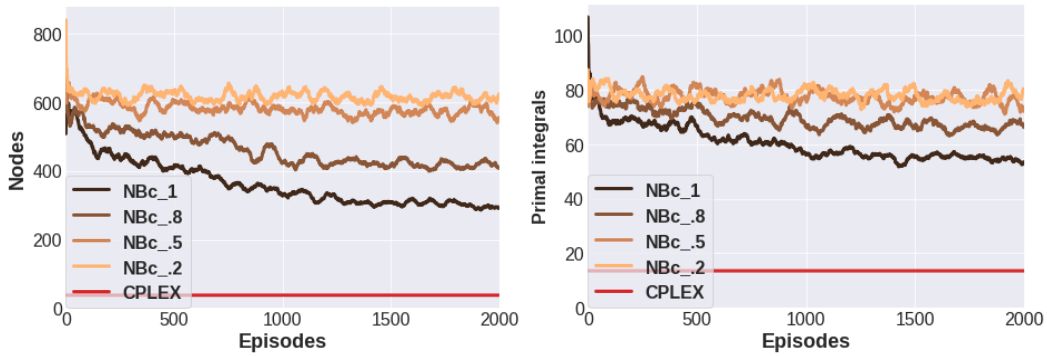
Figure 4.6: Evolution of the errors during the training process for heuristic cost models.



(a) `micro_asym_T6` (b) `hydro_fix_1`

Figure 4.7: Evolution of the errors by depth during the training process for `SBc`.

The previous results have been obtained by considering a unitary discount factor $\gamma = 1$. Figure 4.8 displays the results obtained for different values of γ, and advocates for a high discount factor. As the impact of the discount factor is discussed in Section 4.2.3, we do not elaborate more on this question for the moment.



(a) Tree sizes (b) Primal integral scores

Figure 4.8: Training processes: impact of the discount factor for `NBc` on `micro_asym_T6`.

The poor performances of the agents under heuristic cost models on relatively easy problems highlight the lack of mathematical understanding of the branching dynamics, and suggest that classic heuristics are poor approximations of optimal choices. In the following, we provide theoretical and empirical justifications for using the unitary cost model.

4.2 Reinforcement learning with oracle cost

In the previous section, we used RL to learn a branching strategy by defining a heuristic cost model in the considered MDP. As mentioned earlier, it does not guarantee to find an oracle strategy, *i.e.* optimal with respect to our global objective of minimizing the tree size (3.2). Incidentally, this approach did not produce satisfactory results as shown in the previous section. In the following, we show that solving the MDP with a unitary cost model yields an oracle strategy and conversely. We shall say that such unitary cost is an *oracle cost*. This property is always valid under classical trajectory-based transitions but requires a depth-first search node selection strategy to hold under tree-based transitions. Next, we elaborate on the effect of the γ parameter and propose to use a non-oracle cost model to make the learning task easier. Finally, we offer some insights on different variations around the proposed methodology.

4.2.1 Oracle cost with trajectory-based transitions

Identifying an oracle strategy

As said above, we consider a unitary cost model, setting $c(s, j) = 1$ for any state-action pair. Coupling it with a trajectory-based transition, we obtain the following Bellman equations for the value functions

$$V_\gamma^\pi(s_t) = 1 + \gamma V_\gamma^\pi\left(T(s_t, \pi(s_t))\right) \tag{4.14}$$

$$Q_\gamma^\pi(s_t, j) = 1 + \gamma Q_\gamma^\pi\left(T(s_t, j), \pi\left(T(s_t, j)\right)\right) \tag{4.15}$$

Note that, when setting $\gamma = 1$, we have

$$V_1^\pi(s_t) = \left| \left\{ s_{t'} \in \mathcal{T}^\pi; t' \geqslant t \right\} \right| \tag{4.16}$$

which is the number of iterations of Algorithm 4 left from current state.

We show in Proposition 4.2.1, this value function V_γ^π is consistent with the global objective (3.2), in the sense that an optimal policy for the MDP yields a minimum for the tree size and conversely.

Proposition 4.2.1. *A policy is optimal for the value function V_γ^π defined by Equation* (4.14) *if and only if it is an oracle strategy.*

Proof. *Let us first show it when $\gamma = 1$. At any iteration of Algorithm 4, the size of the tree following policy π from state s_t can be written $T_t^\pi = t - 1 + V_1^\pi(s_t)$. Hence, minimizing the tree size from any state is equivalent to minimizing the value function, which gives*

$$\arg\min_{\pi \in \Pi} T_t^\pi = \arg\min_{\pi \in \Pi} V_1^\pi(s_t) \ni \pi^*$$

The result also stands for $\gamma \in [0, 1)$ as $V_1^{\pi_2}(s_t) \leqslant V_1^{\pi_2}(s_t) \iff V_\gamma^{\pi_2}(s_t) \leqslant V_\gamma^{\pi_2}(s_t)$ where the index indicates the discount factor value. Indeed, writing T_1, T_2 the number of iterations necessary to close all open nodes from s_t following π_1, π_2, we have

$$
\begin{aligned}
& V_1^{\pi_1}(s_t) \leqslant V_1^{\pi_2}(s_t) \\
\iff & T_1 \leqslant T_2 \\
\iff & \sum_{t=0}^{T_1} \gamma^t \leqslant \sum_{t=0}^{T_2} \gamma^t \\
\iff & V_\gamma^{\pi_1}(s_t) \leqslant V_\gamma^{\pi_2}(s_t)
\end{aligned}
$$

Thus, the discount factor does not affect the argmin computation. □

Compared with the heuristic strategies learnt in the previous section, we now have the guarantee of finding an oracle strategy when finding an optimal policy π^*. Besides, using trajectory-based transitions allows us to design an exact algorithm to find an optimal policy for value functions defined by (4.14). The following theorem formally states this point, and is a direct application of Theorem 2.2.1 in our setting.

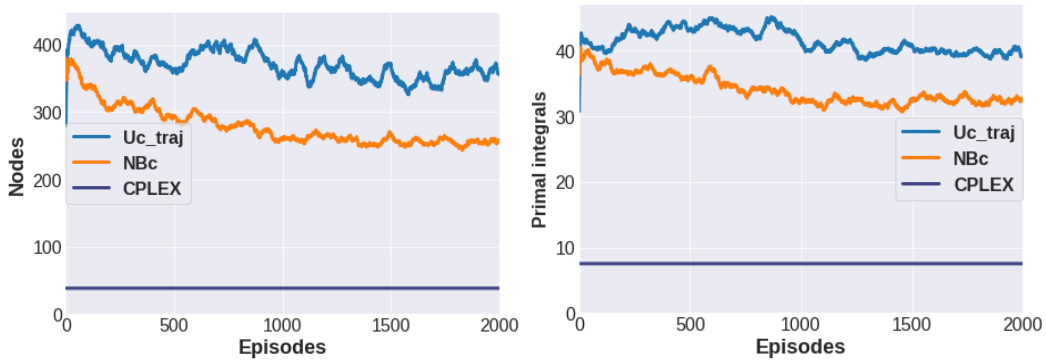
Theorem 4.2.1. *Omitting the reference to the discount factor γ, an optimal policy π^* can be defined by $\pi^*(s) = \arg\min_{j \in \mathcal{J}} V^*(T(s,j))$ and the optimal value function is $V^* = \lim_{k \to +\infty} \mathcal{B}^k V_0$ with V_0 any value function and $\gamma \in [0,1)$.*

Proof of Theorem 4.2.1. *The proof is identical to that of Theorem 2.2.1, considering dirac distributions for transition probabilities to get back to our setting of deterministic transitions.* □

Learning the oracle strategy

So far, we showed that considering trajectory-based transitions allows us to use classical dynamic programming to derive an oracle strategy. Still, the value iteration algorithm of Theorem 4.2.1 is inapplicable in practice and we must leave it to approximation algorithms, see for instance the previous section and Algorithm 6 with the adequate Q-value function (4.15). As trajectory-based transitions put us in the classical RL setting, the bound on the performance of a greedy policy with respect to a sub-optimal policy holds (see Equation (2.24)).

Figure 4.9 compares the training process observed when considering the unitary cost model under trajectory-based transitions with the heuristic cost model `NBc` on `micro_asym_T6`. Due to the bad performances observed, we do not display other experiments at this point.



(a) Tree sizes (b) Primal integral scores

Figure 4.9: Training processes: comparison between the unitary cost model under trajectory-based transitions (`Uc_traj`) and the heuristic cost `NBc` under tree-based transitions on `micro_asym_T6`.

Credit assignment problem

Despite the coherence of the designed value function with respect to the global objective (3.2), using it without taking into account the tree structure of the environment appears to be quite inefficient. Our explanation for this failure is that this value function is not sufficiently informative and localized, and therefore suffers from credit assignment. As presented in Section 2.2, a problem of credit assignment traditionally arises in RL when the rewards are rare, causing difficulties in affecting a pertinent value to individual state-action pairs. More generally, we face such issue when the value of a state-action pair depends on numerous future choices, which implies troubles in assessing its actual value.

An example of such credit assignment issue is given in Figure 4.10 when considering the value function (4.16) induced by trajectory-based transitions with a unitary cost model. Let us assimilate B&B nodes to states and focus on the actions taken at the two children of the root node, s_l (left node) and s_r (right node). Noting j_k^i the action at node s_k ($k \in \{l, r\}$) in Figure 4.10.i), we have $(s_r, j_r^a) = (s_r, j_r^b)$, $(s_l, j_l^b) = (s_l, j_l^c)$. Omitting the reference to the policy, we have when $\gamma = c(s, j) = 1$ for any state-action pair (s, j): $Q(s_l, j_l^a) = 8 < 10 = Q(s_l, j_l^b)$. So state-action (s_l, j_l^a) is better evaluated than (s_l, j_l^b) which seems natural as it allows to produce a smaller tree under s_l. However, we have $Q(s_l, j_l^a) = 8 = Q(s_l, j_l^c)$, although $(s_l, j_l^c) = (s_l, j_l^b)$. This instability is due to a change in policy at s_r.

This example illustrates the fact that a Q-function is less stable, more sensitive to changes in policy

when it depends on long trajectories, hence making it difficult for an agent to precisely assess the value of a specific state-action pair.

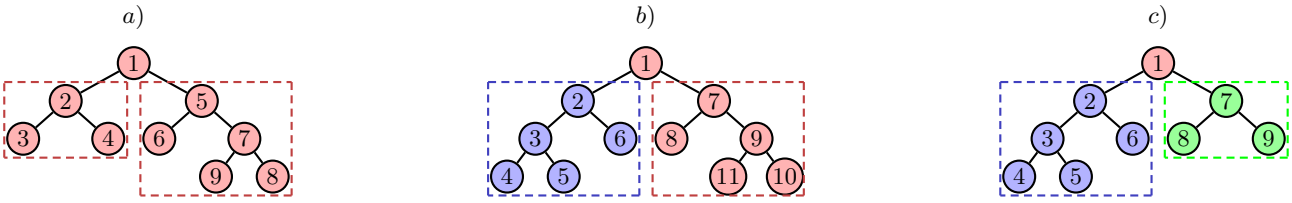


Figure 4.10: Illustration of the credit assignment problem induced by trajectory-based transitions. The three figures represent B&B trees, where the numbers indicate the visiting order of the nodes (the node selection follows here a DFS strategy in each tree). Red nodes correspond to B&B nodes where branching decisions are those of figure a), blue ones are those where actions and/or B&B nodes are different in figure b) compared with a), and green ones are those different in figure c) compared with b).

This example has two interests. First, it allows to highlight the fact that designing a value function in line with a global objective does not naturally comes with an easy learning task, as illustrated in the experiments displayed in Figure 4.9, and that some improvements may be brought upon it. Second, it gives the intuition that focusing on the subtree under a state-action pair may be more stable, as the incoherence presented in Figure 4.10 disappears if one considers the subtree size instead of the global size. Next section is about considering such value function using tree-based transitions and ensuring that, in a specific setting, it still comes with the oracle property of minimizing the tree size.

4.2.2 Oracle cost with tree-based transitions

So far, we defined a value function in line with the objective of minimizing the tree size while considering trajectory-based transitions. Acknowledging a credit assignment issue, we use in the following tree-based transitions to define a new value function and a setting in which an optimal policy is also an oracle strategy and conversely.

Impact of tree-based transitions on value functions

Using a unitary cost model as in the previous section, the Bellman equations (4.7) and (4.8) give the following value functions

$$V^\pi(s) = 1 + \gamma\Big[V^\pi\left(D_0^\pi(s,\pi(s))\right) + V^\pi\left(D_1^\pi(s,\pi(s))\right)\Big] \tag{4.17}$$

$$Q^\pi(s,j) = 1 + \gamma\Big[Q^\pi\left(D_0^\pi(s,j),\pi(s)\right) + Q^\pi\left(D_1^\pi(s,j),\pi(s)\right)\Big] \tag{4.18}$$

When setting γ to 1, $V^\pi(s)$ is the size of the subtree rooted in the B&B node $\zeta(s)$ associated to s when following policy π. Likewise, $Q^\pi(s,j)$ is the size of the subtree rooted in $\zeta(s)$ when branching on variable j at this node.

Let us compare value functions (4.14) and (4.17) to see why using tree-based transitions partially solves the credit assignment problem inherent to trajectory-based transitions pointed out in Figure 4.10. We write here $\mathcal{S}_1(s,\pi)$ (resp. $\mathcal{S}_2(s,\pi)$) the set of states where costs are used in the derivation of the value function (4.14) (resp. (4.17)) for policy π from state s. These sets represent all the descendants of s under the according transitions. We have then $\mathcal{S}_2(s,\pi) \subseteq \mathcal{S}_1(s,\pi)$ as illustrated in Figure 4.11, resulting in the tree-based value function being less dependent to other choices made in the tree, hence more stable and informative.
In this matter, preferring tree-based transitions over trajectory-based transitions matches the remark of Newel, cited by Minsky in its seminal paper [33]: "it is extremely doubtful whether there is enough information in win, lose, or draw when referred to the whole play of the game to permit any learning at all over available time scales ... For learning to take place, each play of the game must yield much more information. This is [...] achieved by breaking the problem into components." When considering a unitary cost model under tree-based transitions, we break the problem of minimizing the tree size into the many sub-problems of minimizing the subtree sizes.

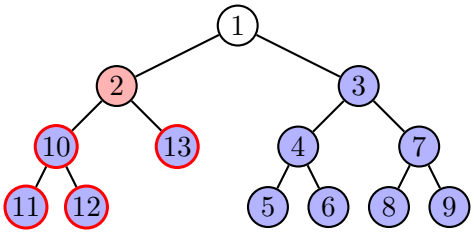


Figure 4.11: This figure represents a B&B tree, where the numbers indicate the visiting order of the nodes. Blue nodes are the descendants of the red node under trajectory-based transitions, whereas blue nodes with red borders are the descendants of the same red node under tree-based transitions.

Reconciling optimal and oracle strategies

Even if tree-based transitions allow to reduce the credit assignment issue, we would enjoy to have both stability and the oracle property at the same time. In fact, we show that an optimal policy for the value function (4.17) is not anymore an oracle strategy in general (Proposition 4.2.2). Nonetheless, Proposition 4.2.4 asserts that it is the case when considering DFS node selection strategies, allowing to reconcile optimal and oracle strategies.

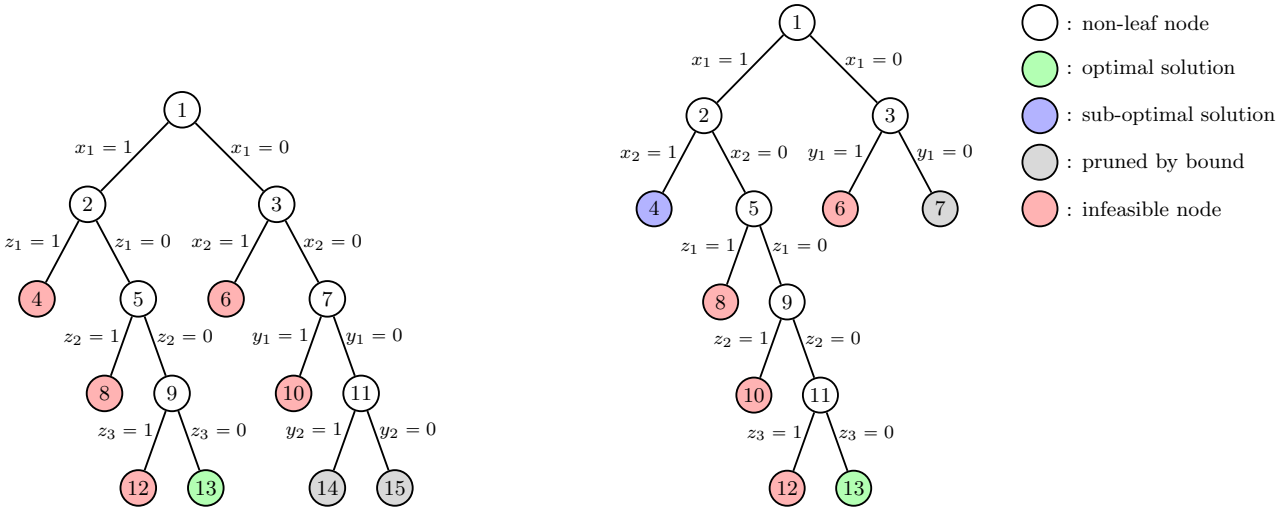
Proposition 4.2.2. *An optimal policy π^* for the value function* (4.17) *is not necessarily an oracle strategy.*

Proof. *Setting $\gamma = 1$, let us build an example where minimizing the subtree does not produces a minimal tree when following a naive Breadth-First Search (BrFS) node selection strategy.*
The idea to produce this example is the following. We build a problem where an optimal solution can only be found in one side of the tree. Then, we design a case where taking a detour (branching on an unnecessary variable to find the optimum) allows to obtain quickly a bound which enables pruning on the non-optimal side of the tree. If one wants to minimize the subtree on the optimal side, the early

bound is not found and the global tree is bigger.

$$
\begin{aligned}
\max_{x,y,z} \quad & 3x_1 - 0.2x_2 + \sum_{i=1}^{3} y_i + 0.005 \sum_{k=1}^{3} z_k \\
s.t. \quad & x_1 + z_k \leqslant 1.5 \ , \ k \in \{1,2,3\} && (c1) \\
& x_1 + y_i \leqslant 1 \ , \ i \in \{1,2,3\} && (c2) \\
& x_1 + \sum_{i=1}^{3} y_i \leqslant 2.4 && (c3) \\
& x_2 - x_1 \leqslant 0 && (c4) \\
& x_2 + z_k \leqslant 1 \ , \ k \in \{1,2,3\} && (c5) \\
& x_1 + x_2 + y_i \geqslant 0.1 \ , \ i \in \{1,2,3\} && (c6) \\
& y_i + \sum_{k=1}^{3} z_k \leqslant 1.2 \ , \ i \in \{1,2,3\} && (c7) \\
& x_i \in \{0,1\} \ , \ y_k \in \{0,1\} \ , \ z_k \in \{0,1\} \ , \ i \in \{1,2\} \ , \ k \in \{1,2,3\} && (c8)
\end{aligned}
\tag{4.19}
$$

Using Algorithm 4 to solve problem (4.19)*, we consider the case where the first branching decision is on* x_1*. From this initial move, we build in Figure 4.12 two B&B trees, with or without the imperative to minimize subtrees using a naive BrFS node selection strategy from left to right. We see that the tree obtained when minimizing each subtree (left tree) is bigger than a minimal tree (right tree).*



(a) Optimal strategy (b) Oracle strategy

Figure 4.12: B&B trees for optimal and oracle strategies using a BrFS node selection strategy for solving problem (4.19). The numbering corresponds to the node visiting order.

Let us explicit why these trees are minimal. First, notice that the optimal solution $\{x_1 = 1, x_2 = 0, y = z = 0\}$ *is unique by construction and belongs to the subtree of node* 2.
If one wants to minimize the subtrees, this solution must, if possible, be found in a single dive under node 2. *Besides, the slackness in constraints* $(c1)$ *and* $(c7)$ *requires to branch on each* z *variable to find this solution. Thus the branching strategy in* a*) produces a minimal subtree under node* 2. *As no primal bound is found before finding the solution and a solution exists in the subtree of node* 3*, minimizing the subtree under this node is achieved by a single dive. The same reasoning at each node guarantees that we minimized each subtree in tree* a*).*
The minimal tree b*) is obtained by taking a detour in the path to the optimal solution so as to prune entirely the subtree of node* 3. *As the detour is of minimal length (it only adds two nodes to the subtree of node* 2*), we know that tree* b*) is minimal.* □

Proposition 4.2.3. *Let* $\mathcal{O}_t$ *be the set of open nodes at a given iteration* t *of Algorithm 4 using a DFS node selection strategy and* s *the state corresponding to the immediately selected node* ζ *in this set. Let us write* s'_1, s'_2 *the states corresponding to the first node* ζ' *in* $\mathcal{O}_t$ *to be visited after* s*, while running respectively the branching policy* π_1 *and* π_2 *between the visits of* ζ *and* ζ'*. Then, under Hypothesis 4.1.1 we have*

(i) $\min_\pi V^\pi(s'_1) = \min_\pi V^\pi(s'_2)$

(ii) For any state $s \in \mathcal{S}$*, the dynamic programming equation*
$V^*(s) = \min_{j \in \mathcal{J}} 1 + \gamma\Big[V^*\left(D_0^*(s,j)\right) + V^*\left(D_1^*(s,j)\right)\Big]$ *holds.*

Proof. *Due to the DFS node selection strategy, the primal bounds at* s'_1 *and* s'_2 *are equal since the full subtree rooted in* s *has been expanded between the visits of* ζ *and* ζ'*, no matter the branching policy. As* $\zeta' = \zeta(s'_1) = \zeta(s'_2)$*, taking the same sequence of branching decisions under* s'_1 *and* s'_2 *will lead to the same subtree in Algorithm 4 due to Hypothesis 4.1.1, which gives* (i)*.*
Consider with no loss of generality that the first visited child is the one corresponding to the constraint $\{x_j = 0\}$*, we have by definition, for any state* $s \in \mathcal{S}$*:*

$$V^*(s) = \min_{\substack{\pi_1, \pi_2 \in \Pi \\ j \in \mathcal{J}}} \left\{1 + \gamma\Big[V^{\pi_1}\left(D_0^{\pi_1}(s,j)\right) + V^{\pi_2}\left(D_1^{\pi_1}(s,j)\right)\Big]\right\}$$

when π_1 *(resp.* π_2*) is the branching policy under the first (resp. second) visited child. Due to* (i)*,* $\min_{\pi_2 \in \Pi} V^{\pi_2}\left(D_1^{\pi_1}(s,j)\right)$ *is independent of* π_1*, which gives* (ii)*.* □

Proposition 4.2.4. *When following a DFS node selection strategy, a policy is optimal for the value function* (4.17) *with* $\gamma = 1$ *if and only if it is an oracle strategy.*

Proof. *Let* s_t *be a state associated to a non-leaf node* $\zeta(s_t)$ *in a B&B tree. Due to DFS, the subree rooted in* $\zeta(s_t)$ *will be fully expanded at iteration* $t_1^{\pi_1}$ *when following* π_1 *before visiting any node of* $\mathcal{O} \equiv \mathcal{O}_{t_1^{\pi_1}} \equiv \mathcal{O}_t \backslash \{\zeta(s_t)\}$*. Thus,* $V^{\pi_1}(s_t)$ *is the subtree size rooted in* $\zeta(s_t)$ *when following* π_1 *in this subtree and we can write* $V^{\pi_2}_{t_1^{\pi_1}}(\mathcal{O}|\pi_1)$ *the number of nodes in the subtrees rooted in* $\mathcal{O}$ *when following* π_2*. By Proposition 4.2.3, we have for any valid branching policy* π_1'*:* $\min_{\pi_2} V^{\pi_2}_{t_1^{\pi_1}}(\mathcal{O}|\pi_1) = \min_{\pi_2} V^{\pi_2}_{t_1^{\pi_1'}}(\mathcal{O}|\pi_1') \equiv V_t^{\pi_2^*}(\mathcal{O})$*. As a consequence, writing* $V_t(\pi_1, \pi_2)$ *the size of the tree obtained by following* π_1 *and* π_2 *after* s_t*, we have*

$$\begin{aligned}
&\arg\min_{\pi_1}\left\{\min_{\pi_2}\left\{V_t(\pi_1,\pi_2)\right\}\right\} = \arg\min_{\pi_1}\left\{\min_{\pi_2}\left\{t-1+V^{\pi_1}(s_t)+V^{\pi_2}_{t_1^{\pi_1}}(\mathcal{O}|\pi_1)\right\}\right\}\\
=&\arg\min_{\pi_1}\left\{V^{\pi_1}(s_t)+\min_{\pi_2}\left\{V^{\pi_2}_{t_1^{\pi_1'}}(\mathcal{O}|\pi_1')\right\}\right\} = \arg\min_{\pi_1}\left\{V^{\pi_1}(s_t)+V_t^{\pi_2^*}(\mathcal{O})\right\}\\
=&\arg\min_{\pi_1}\left\{V^{\pi_1}(s_t)\right\}
\end{aligned}$$

Minimizing the tree size is then equivalent to minimizing the subtree size at each state, which implies that an optimal policy is an oracle strategy and reciprocally. □

At this point, we showed that considering DFS node selection strategies allows to learn an oracle strategy by considering a unitary cost model and tree-based transitions. Using DFS allows in a way to make the tree-based MDP consistent with our objective of tree size minimization. The difference with trajectory-based transitions is a more localized, hence stable, value function, which should alleviate the credit assignment issue. Note that the learning task has not fundamentally changed though: instead of learning to build small trees, we will rather learn to build... smaller trees.

Remark 4. The oracle property is only valid for $\gamma = 1$

Contrary to the previous result using trajectory-based transitions, Proposition 4.2.4 is only valid when setting the discount factor to 1. This is due to the fact that, when considering tree-based transitions, the equivalence $V_1^{\pi_1}(s_t) \leqslant V_1^{\pi_2}(s_t) \iff V_\gamma^{\pi_1}(s_t) \leqslant V_\gamma^{\pi_2}(s_t)$ in the proof of Proposition 4.2.1 does not hold anymore. This is illustrated in the next section (see Figure 4.17b) by comparing wide and deep trees. We exhibit a case where $V_1^{\pi_1}(s) < V_1^{\pi_2}(s)$ and $V_\gamma^{\pi_1}(s) > V_\gamma^{\pi_2}(s)$ with $\gamma \in (0, 1)$.

We investigate in Section 4.2.3 the impact of the discount factor on the learning procedure in our specific environment. From the theoretical standpoint, the following theorem states that one can

design a dynamic programming algorithm to obtain the optimal value function only for cases where $\gamma \in [0, 0.5)$. Note here the difference with the generic case, where an optimal policy could not be obtained (see Proposition 4.1.1). This result is due to the use of a DFS node selection strategy which gives us the dynamic programming equation in Proposition 4.2.3 and can be generalized to any cost model with tree-based transitions. As in classical RL (see Chapter 2), one may still obtain the optimal value function and policy without any condition on γ by solving a linear system as the MDP is finite.

Theorem 4.2.2. *Omitting the reference to the discount factor γ, an optimal policy π^* can be defined when following a DFS node selection strategy by $\pi^*(s) = \arg\min_{j \in \mathcal{J}} V^*\left(D_0^*(s, \pi^*(s))\right) + V^*\left(D_1^*(s, \pi^*(s))\right)$ and the optimal value function is $V^* = \lim_{k \to +\infty} \tilde{\mathcal{B}}^k V_0$ with V_0 any value function and $\gamma \in [0, 0.5)$.*

Proof of Theorem 4.2.2. *This is the same proof as in Theorem 4.1.1, substituting V^* for $V^{\sim}$ thanks to Proposition 4.2.3 and replacing $c(s, j)$ by 1.* □

As previously, the theoretical algorithm described in Theorem 4.2.2 is not applicable in practice, and we use Algorithm 6 as an approximation procedure. As for assessing the performance of a greedy policy with respect to a value function, we obtain in Proposition 4.2.5 a similar bound (up to a factor 2 on γ) as in classical RL – see Chapter 2. Thus, using tree-based transitions does not alter the usual guarantees on the performance of a greedy policy.

Proposition 4.2.5. *Let π be a greedy policy with respect to a Q-value function Q and L_Q the loss function such that $L_Q(s) = \tilde{Q}^{\pi}(s, \pi(s)) - Q^*(s, \pi^*(s))$ for all state $s \in \mathcal{S}$, with $\tilde{Q}^{\pi}(s, \pi(s))$ the evaluation of following π from s. L_Q is then the loss in value of state s resulting from following π instead of an optimal policy π^*. Then, if $|Q^*(s, a) - \tilde{Q}^{\pi}(s, a)| \leqslant \varepsilon$ for all $s \in \mathcal{S}$, we have*

$$L_Q(s) \leqslant \frac{2\varepsilon}{1 - 2\gamma}$$

Proof. *Let $z \in \arg\max_{s \in \mathcal{S}} L_Q(s)$. Consider the optimal action $a = \pi^*(z)$ and the greedy action $b = \pi(z)$ at state z, and z_1, z_2 (resp. z_3, z_4) the child states of z when following π (resp. π^*). By definition of a greedy policy, we have*

$$\tilde{Q}^{\pi}(s, b) \leqslant \tilde{Q}^{\pi}(s, a)$$

Since $Q^*(s,a) - \varepsilon \leqslant \tilde{Q}^\pi(s,a) \leqslant Q^*(s,a) + \varepsilon$ *for all* $s \in \mathcal{S}$*, we also have*

$$\begin{aligned} & Q^*(z,b) - \varepsilon \leqslant Q^*(z,a) + \varepsilon \\ \iff & c(z,b) + \gamma\big[V^*(z_3) + V^*(z_4)\big] - \varepsilon \leqslant c(z,a) + \gamma\big[V^*(z_1) + V^*(z_2)\big] + \varepsilon \\ \iff & c(z,b) - c(z,a) \leqslant 2\varepsilon + \gamma\big[V^*(z_1) + V^*(z_2) - V^*(z_3) - V^*(z_4)\big] \end{aligned}$$

As $V^*(s) = Q^*(s, \pi^*(s))$ *for any* $s \in \mathcal{S}$*, the maximal loss is then*

$$\begin{aligned} L_Q(z) &= \tilde{Q}^\pi(z, \pi(z)) - Q^*(z, \pi^*(z)) \\ &= c(z,b) - c(z,a) + \gamma\big[\tilde{Q}^\pi(z_3, \pi(z_3)) + \tilde{Q}^\pi(z_4, \pi(z_4)) - Q^*(z_1, \pi^*(z_1)) - Q^*(z_2, \pi^*(z_2))\big] \\ &\leqslant 2\varepsilon + \gamma\big[\tilde{Q}^\pi(z_3, \pi(z_3)) + \tilde{Q}^\pi(z_4, \pi(z_4)) - Q^*(z_3, \pi^*(z_3)) - Q^*(z_4, \pi^*(z_4))\big] \\ &\leqslant 2\varepsilon + 2\gamma L_Q(z) \end{aligned}$$

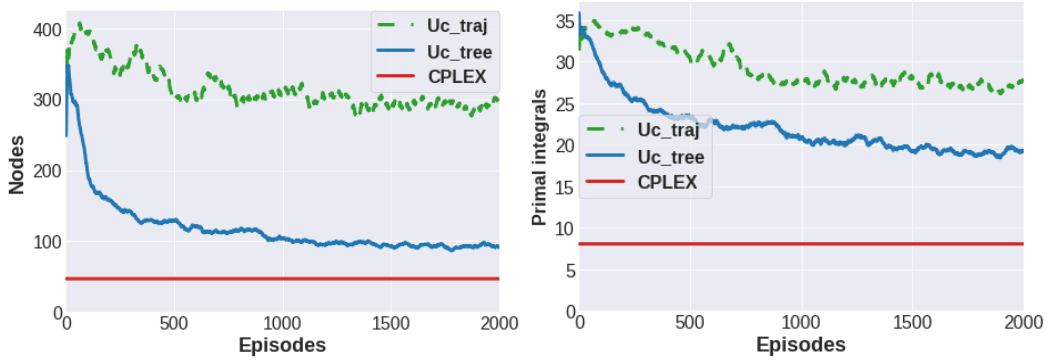
Then,

$$L_Q(z) \leqslant \frac{2\varepsilon}{1 - 2\gamma}$$

□

Experiments

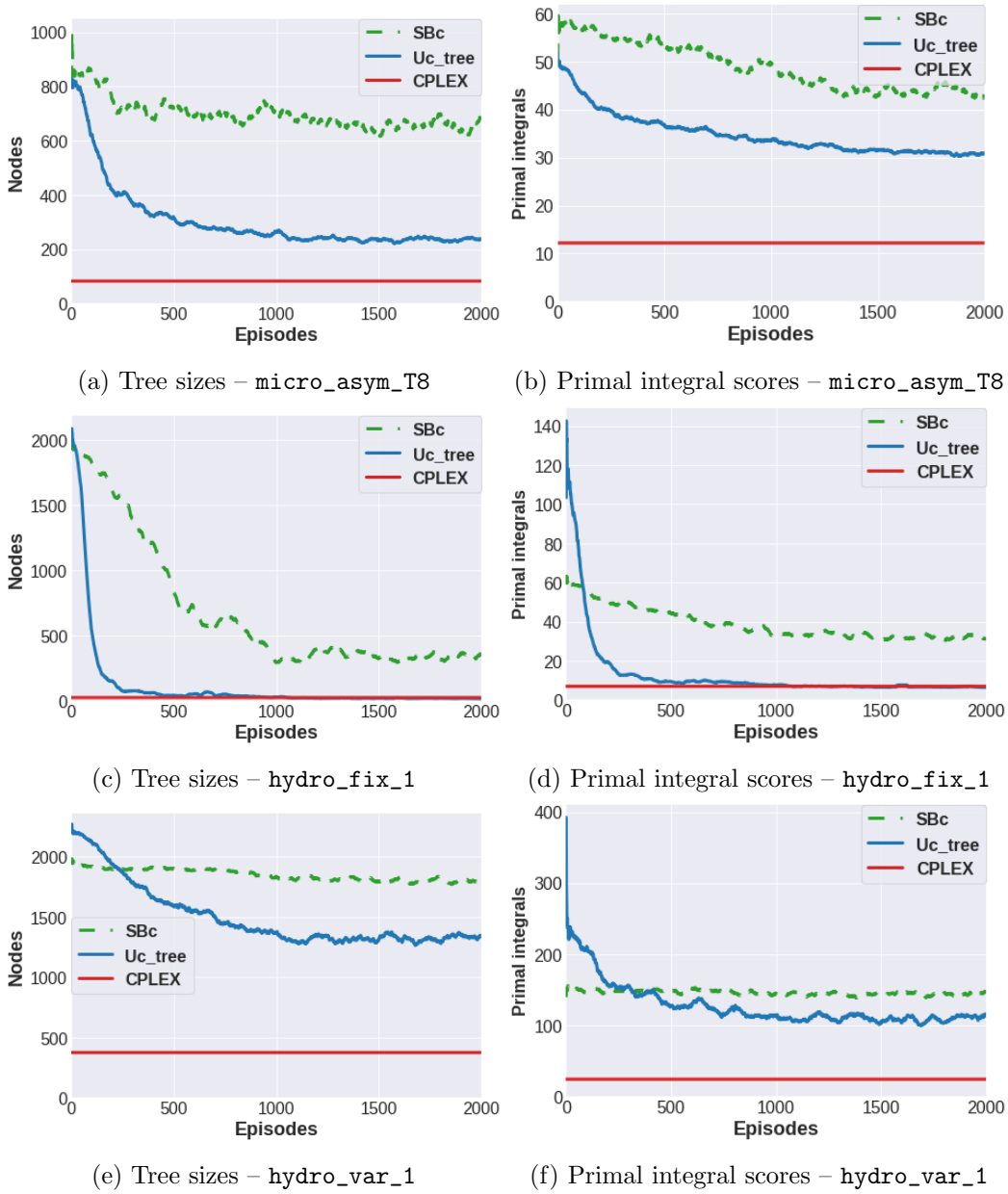
The first point to assess is the relevance of tree-based transitions. Figure 4.13 displays the performances for tree-based and trajectory-based transitions for the unitary cost model. We only display these results on an easy problem as trajectory-based transitions do not show good performances. Further experiments will be presented later on. This result comes as a validation of our decision of considering tree-based transitions and confirms that they allow to ease the learning task, especially by reducing the credit assignment problem.



(a) Tree sizes (b) Primal integral scores

Figure 4.13: Training processes: comparison between tree-based transitions (`Uc_tree`) and trajectory-based transitions (`Uc_traj`) for the unitary cost model on `micro_asym_T6`.

It is also interesting to compare the unitary cost model with heuristic costs. This is the purpose of Figure 4.14, which shows the comparison with the `SBc` heuristic cost – note that DFS is not enforced for the heuristic cost. Here again, we observe better results. However, the comparison with `CPLEX` raises doubts regarding the efficiency of the approach on the more difficult problems. The contrast between the results showed by Figure 4.14c and Figure 4.14e is eloquent. The trained agent competes with `CPLEX` on the problem `hydro_fix_1`, on which any feasible solutions are shared across instances – see Section 3.2. On this problem, the agent manages to find sequences of branching decisions which perform extremely well on every instances. On the problem `hydro_var_1`, even if a random strategy produces more or less the same number of nodes in average (see the intercepts), the agent struggles to find an efficient strategy which can adapt to every instances.



(a) Tree sizes – `micro_asym_T8`

(b) Primal integral scores – `micro_asym_T8`

(c) Tree sizes – `hydro_fix_1`

(d) Primal integral scores – `hydro_fix_1`

(e) Tree sizes – `hydro_var_1`

(f) Primal integral scores – `hydro_var_1`

Figure 4.14: Training processes: comparison of the unitary cost model (`Uc_tree`) with the SB-like heuristic cost (`SBc`) under tree-based transitions on train instances.

4.2.3 On the virtue of short-sightedness: a new cost model

In the previous section, we proposed to use a unitary cost model under tree-based transitions, which provides when solving the MDP to optimality an oracle strategy only when $\gamma = 1$ (see Proposition 4.2.4). Hence, one can legitimately wonder why even considering a discount factor lower than 1

in this setting.

We already saw the theoretical interest of a lower discount factor $\gamma \in [0, 0.5)$, allowing to design an exact dynamic programming procedure in Theorem 4.2.2 to obtain optimal value functions. As Algorithm 6 is an approximate version of such procedure, it may suggest that setting $\gamma < 1$ may have some advantages. In the following, we explicit the practical impact of the discount factor γ and propose a more general value function to strengthen such impact.

Discount factor in classical trajectory-based transitions

From a theoretical and exact standpoint, the discount factor is not justified as soon as the MDP is episodic or finite, as it may alter the optimal policy – see Remark 4. From a practical standpoint however, the discount factor is commonly used in trajectory-based transitions for giving more importance to rewards closer to the current state. This may be desirable, as the observed long-term outcomes have a lower probability to happen under the optimal policy than short-term observations. Let us elaborate on this point and consider our setting, where a state can only be reached from a unique predecessor (recall that a state comprises any information collected so far). Assuming that the current policy is of the form $\pi = \beta\pi^* + (1-\beta)\mathcal{U}_{\mathcal{J}\setminus\{\pi^*\}}$ which is interpreted here as a probability $\beta \in [0, 1]$ to select the optimal action and $1-\beta$ to select uniformly a sub-optimal one. Then, if we observe a sequence $\{s_0, s_1, \dots, s_t\}$, we have by hypothesis the probability of reaching s_t starting from s_0 and following the optimal policy π^*

$$p(s_t|s_0, \pi^*) = \beta^t \tag{4.20}$$

which is a decreasing function of t. Thus, long-term observations should be less trusted than closer ones, which justifies the use of the discount factor in the value function. Such reasoning is independent of the stochasticity or episodic nature of the considered MDP, which are the classical justifications for the discount factor, and hence applies to our setting.

Another point stemming from Equation (4.20) is that the ratio $\frac{p(s_{t'}|s_0,\pi^*)}{p(s_t|s_0,\pi^*)}$ with $t \leqslant t'$ is an increasing function of β. It illustrates that long-term observations are less reliable when the policy is far from the optimal one. Thus, the discount factor also smooths the value function to make it less dependent on the quality of the current policy, which provides more stable targets during the agent training from Algorithm 6, especially in the early episodes.

This simple reasoning allows to understand why the discount factor may be beneficial from a practical

standpoint even if not justified by the considered objective. Other justifications have been highlighted, for example the fact that a lower discount factor allows to tighten error bounds [30] or simply to decrease the targets' standard deviation, hence the needed complexity for the Q-network [32].

Discount factor in tree-based transitions with unitary cost model

The above reasoning still holds when considering tree-based transitions, and the discount factor still makes the value function less dependent on both the quality of the agent and remote states. However, additional remarks can be done when considering a unitary cost model in our specific MDP with tree-based transitions.

First, let us recall that, when γ is set to 1, the value function (4.17) is the subtree size. Such value function induces a high variability in the values used for learning, highly mitigated by the use of a low discount factor. To see this, let us consider a B&B tree of size $N = 2^{p+1} - 1$ with $p \leqslant n$ and let s be the state corresponding to its root node. No matter the shape of this tree, we have $V_1(s) = N$ (the index indicates the value of γ). On the contrary, Proposition 4.2.6 gives an upper bound for $V_\gamma(s)$ which scales much more nicely with respect to N.

Proposition 4.2.6. *The discounted value function* (4.17) *for a state s with subtree size $V_1(s) = N = 2^{p+1} - 1$ is upper bounded by $\frac{1-(2\gamma)^{\log_2(N+1)}}{1-2\gamma}$ for $\gamma \neq 0.5$ and by $\log_2(N+1)$ if $\gamma = 0.5$*

Proof. *A (sub)tree is said full-width when 2^d nodes are visited at depth d, except potentially for the maximal depth. When the tree size is $N = 2^{p+1} - 1$ with $p \leqslant n$, the number of nodes at each depth including the maximal one is 2^d and $p = \log_2(N+1) - 1$ is the maximal depth.*

When using a discount factor $\gamma < 1$, the value function is maximized when the subtree is full-width. This is obtained by solving the following MILP with $\gamma < 1$:

$$\begin{cases} \max\limits_{x} & \sum\limits_{d=0}^{\lceil N/2 \rceil} \gamma^d x_d \\ s.t. & x_0 = 1 \ , \ \sum\limits_{d=0}^{\lceil N/2 \rceil} x_d = N \\ & x_{d+1} \leqslant 2x_d \ , \ d \in \{0, 1, \dots, \lceil N/2 \rceil - 1\} \\ & x \in \mathbb{N}^{\lceil N/2 \rceil + 1} \end{cases}$$

where x_d represents the number of nodes at depth d. The value is then a geometric sum and we have $V_\gamma(s) = \sum_{d=0}^{p} (2\gamma)^d$ which gives the result. □

In addition to limiting the scale of the value function, it also naturally decreases both the average and standard deviation of states' values. Especially, Proposition 4.2.7 asserts that the moments of such values are upper bounded by a constant when using a discount factor lower than 1, whereas they grow linearly and exponentially with the maximal depth when setting $\gamma = 1$.

Proposition 4.2.7. *Let s be the root node of a B&B full-width tree $\mathcal{T}$ of size $N = 2^{p+1} - 1$. When setting $\gamma = 1$, the mean value of states encountered in such tree asymptotically grows linearly with its depth p, while its variance grows exponentially. On the opposite, they both are upper bounded by a constant when using $\gamma \in [0, 1)$ and $\gamma \in [0, \frac{1}{\sqrt{2}})$ respectively.*

Proof. *We write in the following μ_γ and σ^2_γ the empirical mean and variance in a full-width tree of size $N = 2^{p+1} - 1$ when using the discount factor γ.*

For $\gamma = 1$, the value of a state at depth d is $2^{p-d+1} - 1$, which yields

$$\begin{aligned}
\mu_1 &= \frac{1}{N}\sum_{d=0}^{p} 2^d(2^{p-d+1} - 1) = \frac{1}{N}\Big[\sum_{d=0}^{p} 2^{p+1} - \sum_{d=0}^{p} 2^d\Big] = (p+1)\frac{2^{p+1}}{N} - \frac{1}{N}(2^{p+1} - 1) \\
&= \frac{p2^{p+1} + 1}{N} = p\frac{2^{p+1} - 1}{N} + \frac{p+1}{N} = p + \frac{p+1}{N} = \mathcal{O}(p) \\
\sigma_1^2 &= \frac{1}{N}\sum_{d=0}^{p} 2^d(2^{p-d+1} - 1 - \mu_1)^2 \\
&= \frac{1}{N}\left(\sum_{d=0}^{p} 2^d(2^{p-d+1} - 1)^2\right) - \mu_1^2 \quad \textit{(classical variance reformulation)} \\
&= -\mu_1^2 + \frac{1}{N}\sum_{d=0}^{p} 2^d\left(2^{2p-2d+2} + 1 - 2^{p-d+2}\right) \\
&= -\mu_1^2 + \frac{1}{N}\Big[2^{p+1}\left(\sum_{d=0}^{p} 2^{p-d+1}\right) + N - 2^{p+2}(p+1)\Big] \\
&= -\mu_1^2 + \frac{1}{N}\Big[2^{p+1}\left(\sum_{d=1}^{p+1} 2^{d}\right) + N - 2^{p+2}(p+1)\Big] \\
&= -\mu_1^2 + \frac{1}{N}\Big[2^{p+2}N + N - 2^{p+2}(p+1)\Big] \\
&= -\mu_1^2 + \frac{1}{N}\Big[2^{p+2}N + N - 2N(p+1) - 2(p+1)\Big] \\
&= 2^{p+2} - 2p - 1 - \mu_1^2 - 2\frac{p+1}{N} = \mathcal{O}(2^p)
\end{aligned}$$

For $\gamma \in [0, 0.5) \cup (0.5, 1)$ (the case $\gamma = 0.5$ is omitted for the sake of simplicity), the value of a state at depth d is $\frac{(2\gamma)^{p-d+1}-1}{2\gamma-1}$, which gives

$$\begin{aligned}
\mu_\gamma &= \frac{1}{N}\sum_{d=0}^{p} 2^d \frac{(2\gamma)^{p-d+1}-1}{2\gamma-1} = \frac{1}{(2\gamma-1)N}\Big[2^{p+1}\left(\sum_{d=0}^{p}\gamma^{p-d+1}\right) - N\Big] \\
&= \frac{1}{(2\gamma-1)N}\Big[2^{p+1}\left(\sum_{d=1}^{p+1}\gamma^{d}\right) - N\Big] = \frac{1}{(2\gamma-1)N}\Big[2^{p+1}\gamma\frac{1-\gamma^{p+1}}{1-\gamma} - N\Big] \\
&= \frac{1}{(2\gamma-1)N}\Big[N\gamma\frac{1-\gamma^{p+1}}{1-\gamma} + \gamma\frac{1-\gamma^{p+1}}{1-\gamma} - N\Big] \\
&= \mathcal{O}\left(\frac{1}{(2\gamma-1)}\left(\frac{\gamma}{1-\gamma}-1\right)\right) = \mathcal{O}\left(1\right)
\end{aligned}$$

When considering $\gamma \in [0, 0.5) \cup (0.5, \frac{1}{\sqrt{2}})$*, we have*

$$\begin{aligned}
\sigma_\gamma^2 &= \frac{1}{N}\sum_{d=0}^{p} 2^d \left(\frac{(2\gamma)^{p-d+1}-1}{2\gamma-1} - \mu_\gamma\right)^2 = \frac{1}{N}\left(\sum_{d=0}^{p} 2^d \left(\frac{(2\gamma)^{p-d+1}-1}{2\gamma-1}\right)^2\right) - \mu_\gamma^2 \\
&= -\mu_\gamma^2 + \frac{1}{N(2\gamma-1)^2}\sum_{d=0}^{p} 2^d(2\gamma)^{2p-2d+2} + 2^d - 2^{p+2}\gamma^{p-d+1} \\
&= -\mu_\gamma^2 + \frac{1}{N(2\gamma-1)^2}\Big[2^{p+1}\left(\sum_{d=0}^{p}(2\gamma^2)^{p-d+1}\right) + N - 2^{p+2}\sum_{d=0}^{p}\gamma^{p-d+1}\Big] \\
&= -\mu_\gamma^2 + +\frac{1}{N(2\gamma-1)^2}\Big[2^{p+1}\left(\sum_{d=1}^{p+1}(2\gamma^2)^{d}\right) + N - 2^{p+2}\sum_{d=1}^{p+1}\gamma^{d}\Big] \\
&= -\mu_\gamma^2 + \frac{1}{N(2\gamma-1)^2}\Big[2^{p+1}2\gamma^2\frac{1-(2\gamma^2)^{p+1}}{1-2\gamma^2} + N - 2^{p+2}\gamma\frac{1-\gamma^{p+1}}{1-\gamma}\Big] \\
&= -\mu_\gamma^2 + \frac{1}{N(2\gamma-1)^2}\Big[N2\gamma^2\frac{1-(2\gamma^2)^{p+1}}{1-2\gamma^2} + 2\gamma^2\frac{1-(2\gamma^2)^{p+1}}{1-2\gamma^2} + N - 2N\gamma\frac{1-\gamma^{p+1}}{1-\gamma} - 2\gamma\frac{1-\gamma^{p+1}}{1-\gamma}\Big] \\
&= \mathcal{O}\left(\frac{1}{(2\gamma-1)^2}\left(\frac{2\gamma^2}{1-2\gamma^2} + 1 - \frac{2\gamma}{1-\gamma}\right)\right) = \mathcal{O}(1)
\end{aligned}$$

□

Despite the normalizing benefits induced by using a discount factor, it also comes with the risk of flattening too much the value function, then preventing from distinguishing between bad and good actions. This is illustrated by Figure 4.15, where we see that using a too low value of γ makes it difficult to make out the values of highly different states.

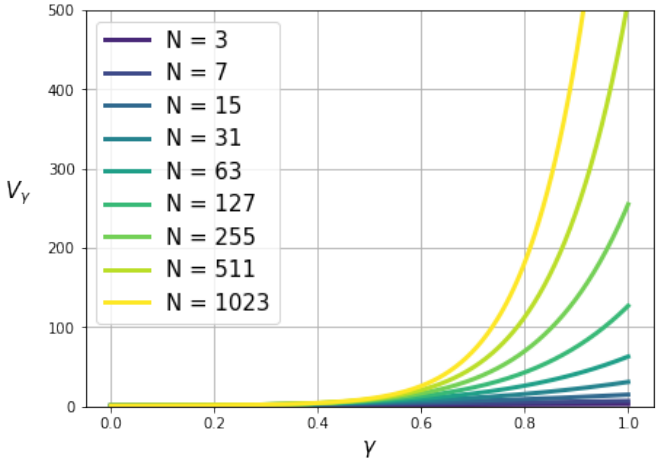


Figure 4.15: Root state value V_γ for full-width trees of size N.

As pointed out in Section 4.2.2, one loses the equivalence between optimal and oracle strategies when setting a discount factor different from 1. In fact, doing so biases the value function in favor of imbalanced trees, as we will illustrate now by elaborating on two extreme cases. Let us consider the root state s_w of a full-width tree of size $N = 2^{p+1} - 1$ and the root state s_d of a full-depth tree with the same size N. Here, we call *full-depth* a tree which, except for depth 0, only contains two nodes at each depth. When such tree is of size N, its depth is $\frac{N-1}{2}$. Figure 4.16 illustrates the two types of trees. We have $V_1(s_w) = V_1(s_d) = N$, $V_\gamma(s_w) = \frac{1-(2\gamma)^{\log_2(N+1)}}{1-2\gamma}$ and $V_\gamma(s_d) = 1 + 2\gamma\frac{1-\gamma^{\frac{N-1}{2}}}{1-\gamma}$ for $\gamma \in [0, 0.5) \cup (0.5, 1)$. Figure 4.17a, which displays the ratio $V_\gamma(s_w)/V_\gamma(s_d)$ as a function of γ, shows that discounted value functions assign a higher value to full-width trees, with a non-monotonic impact. Figure 4.17b represents the difference $V_\gamma(s_w) - V_\gamma(s_d)$ when $V_1(s_d) = V_1(s_w) + 100$ and illustrates the lost of the equivalence between optimal and oracle strategies mentioned earlier. In practice, the discount factor thus encourages the agent to make choices which allow for early pruning, which is in general a safe strategy. This is all the more interesting as biasing the value function towards early pruning incorporates some short-term considerations in the value function, allowing to have meaningful feedbacks even in the early stages of training – see Equation (4.20).

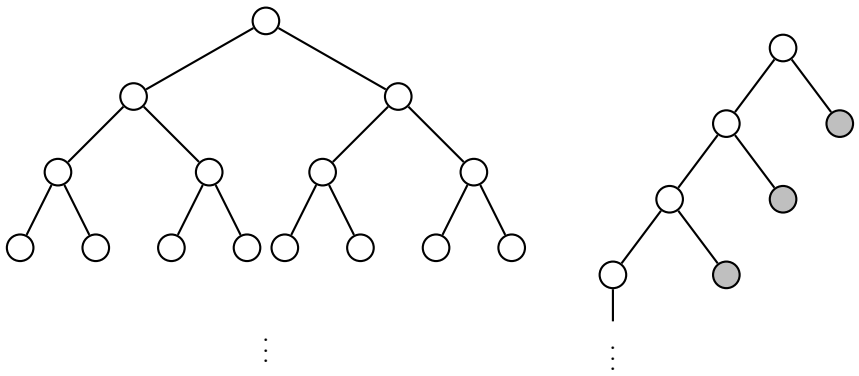
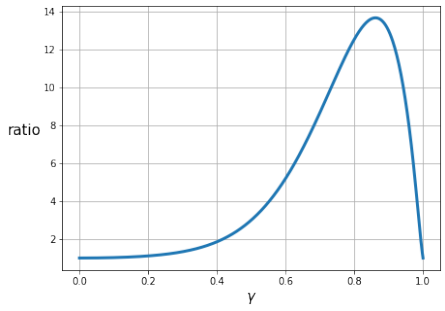
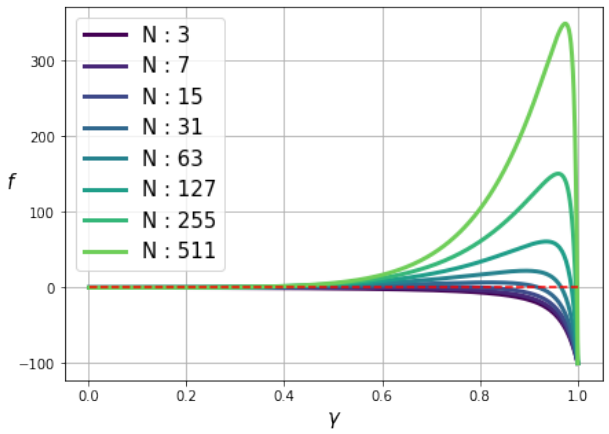


Figure 4.16: Full-width (left) and full-depth (right) trees, grey nodes representing leaves.



(a) $V_\gamma(s_w)/V_\gamma(s_d)$ as a function of γ for trees of size $V_1(s_d) = V_1(s_w) = 511$.

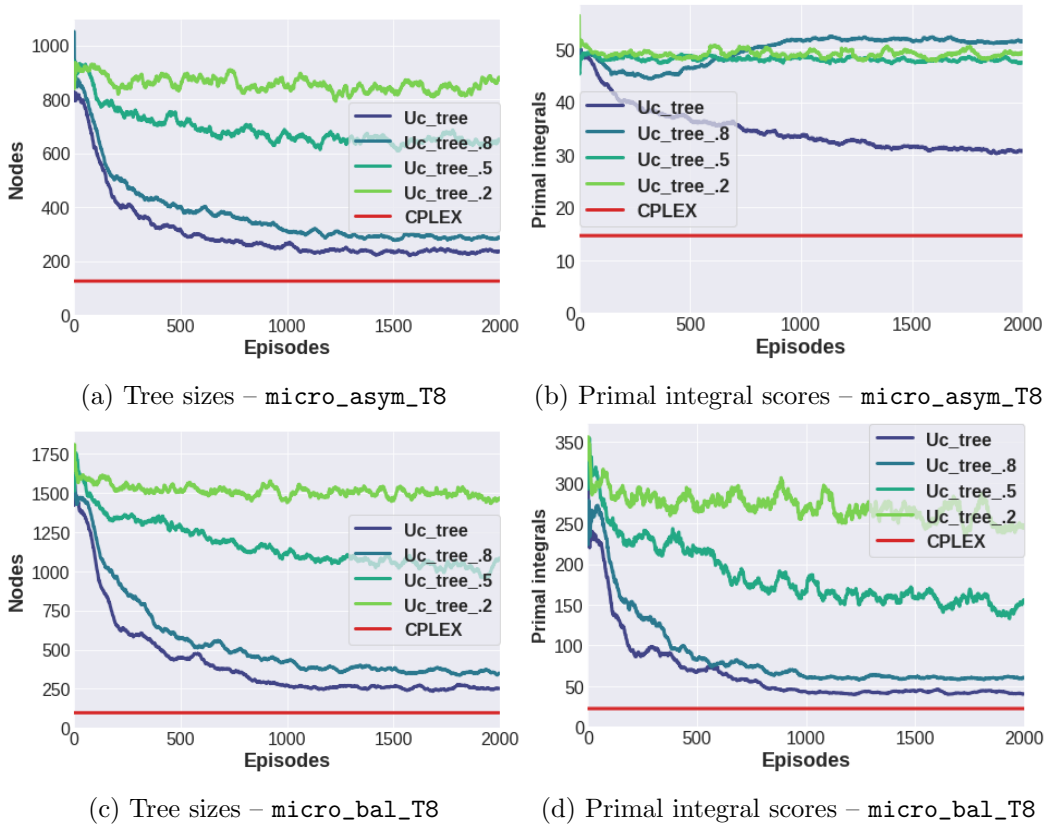


(b) $V_\gamma(s_w) - V_\gamma(s_d)$ when $V_1(s_d) = V_1(s_w) + 100 = N + 100$ for different values of N. This figure illustrates the fact that the equivalence $V_1^{\pi_1}(s_t) \leqslant V_1^{\pi_2}(s_t) \iff V_\gamma^{\pi_1}(s_t) \leqslant V_\gamma^{\pi_2}(s_t)$ in the proof of Proposition 4.2.1 does not hold anymore when considering tree-based transitions.

Figure 4.17: Illustrations of the impact of the discount factor on full-depth and full-width trees.

Experiments on the discount factor for the unitary cost model under tree-based transitions

The influence of the discount factor on a unitary cost model is displayed in Figure 4.18. First, we see that low values of γ prevent the agent from learning efficiently. As explained earlier, this comes from the fact that the resulting squashing of the value function makes it less informative. Second, one can notice when observing Figure 4.18b that the discount factor may have an unexpected effect on the agent's ability to find early the optimal solution. Especially, the curves associated to the value $\gamma = 0.8$ show a decreasing tree size with an increasing primal integral value.



(a) Tree sizes – `micro_asym_T8`

(b) Primal integral scores – `micro_asym_T8`

(c) Tree sizes – `micro_bal_T8`

(d) Primal integral scores – `micro_bal_T8`

Figure 4.18: Training processes: impact of the discount factor ($\gamma \in \{0.2, 0.5, 0.8, 1\}$) in the unitary cost model under tree-based transitions.

Solving the dilemma of the discount factor for the unitary cost model

From a practical standpoint, we saw that the value function obtained when considering the unitary cost model is not particularly well suited for the learning task. Indeed, as explained earlier in the setting of tree-based transitions, choosing $\gamma = 1$ makes the value function equal to the subtree size rooted in the considered state, which implies many issues.

First, it produces a target distribution with a large support, as the value at the root node equals the tree size whereas that at a leaf node equals one. Second, this value highly varies with the quality of the agent and its ability to produce small trees. In other words, the targets pursued by the agent are permanently moving along the training process as it becomes more efficient. Third, their support (and thus the mean value) also depends on the instance sampled at a given episode, some being intrinsically more difficult to solve than others. Last but not least, the value function may vary drastically from a state to its direct child, as illustrated in Figure 4.19, which makes the function highly non-smooth in

the state-space and thus makes the learning task harder.

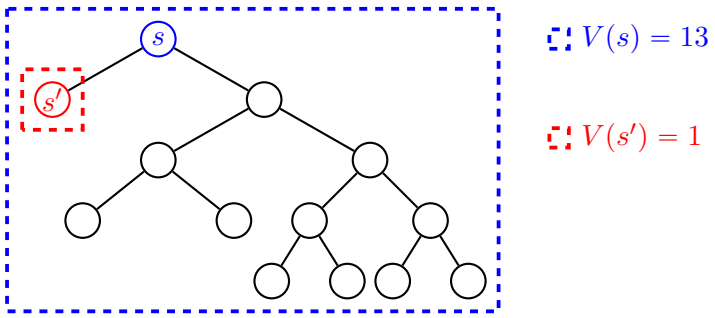


Figure 4.19: Illustration of the fact that the unitary cost model produces highly non-smooth value functions. In this example, s and s' may be similar states with very different values.

At this point, we are facing some kind of a dilemma. On the one hand, we saw that a low value of gamma may be interesting as it smooths the highly volatile value function and makes it less dependent on errors made in future choices. On the other hand, a low discount factor squashes the value function and does not discriminate between full-width and full-depth trees. Besides, one loses the oracle property when setting $\gamma < 1$ under tree-based transitions.
We solve this question by considering an h-ahead score (see Section 4.1.3) instead of a unitary cost model. By setting $\nu^0 = 1$ and ν^1 to the cardinal function in the definition of an h-ahead heuristic, we consider the heuristic cost model $c(s, j) = \nu_h(s, j, \pi) = 1 + \left|\mathcal{D}^{\pi,h}(s, j)\right| \equiv |\mathcal{T}_h^\pi(s, j)|$, which is the size of the subtree of depth h rooted in s when following π after branching on j. We call this cost model the *subtree cost model*, leading to the value function

$$V_{h,\gamma}^\pi(s) = |\mathcal{T}_h^\pi(s)| + \gamma\Big[V_{h,\gamma}^\pi\left(D_0^\pi(s, \pi(s))\right) + V_{h,\gamma}^\pi\left(D_1^\pi(s, \pi(s))\right)\Big] \tag{4.21}$$

Note that by setting $h = 0$ we get back to our previous discounted value function under the unitary cost model. This generalization can be seen as a way of "trusting" more the impact of an action up to a certain horizon h. We see in Figure 4.20 that this new value function even strengthens the discrimination between balanced and imbalanced trees.

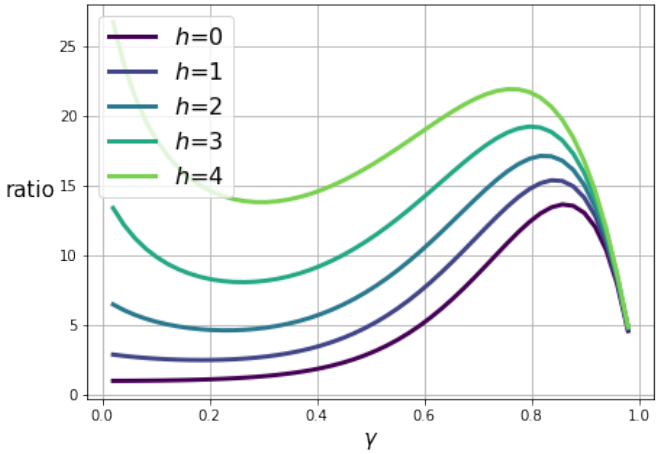
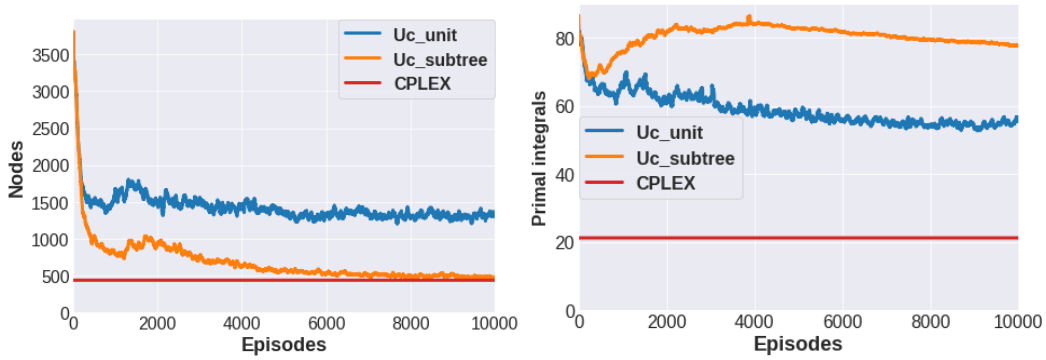


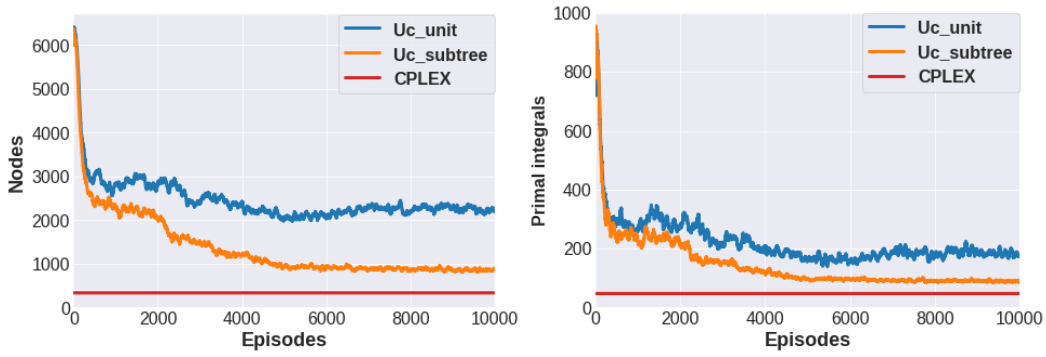
Figure 4.20: $V_{h,\gamma}(s_w)/V_{h,\gamma}(s_d)$ as a function of γ for trees of size $V_1(s_d) = V_1(s_w) = 511$, s_d and s_w being root states of respectively full-depth and full-width trees.

Experiments on the subtree cost model under tree-based transitions

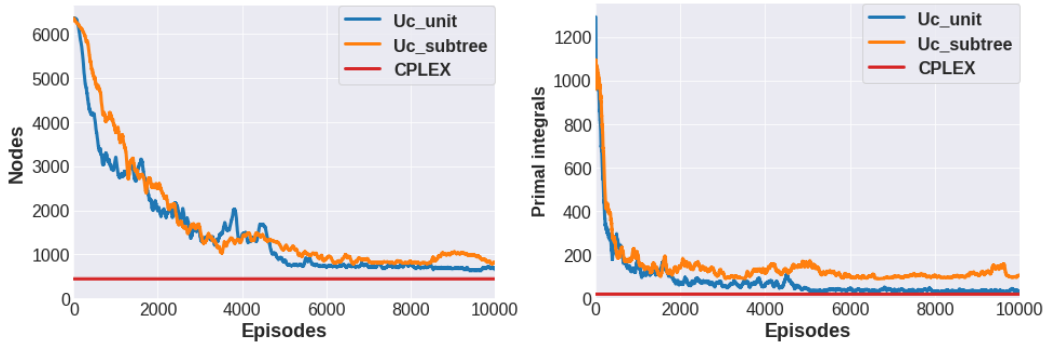
The solution put forward to reduce the volatility of the value function and solve the discount factor dilemma was to use the value function defined in Equation (4.21), using the subtree cost model. Figure 4.21 displays the results of such value function and compares it with the unitary cost model on more difficult problems than those used in previous experiments. The results on train instances appear systematically better when using the subtree cost model and deserve some additional investigations. Before doing so, note that a potential undesirable outcome of using a more short-sighted value function is given by Figure 4.21b, in which we see poorer results regarding primal integral scores.



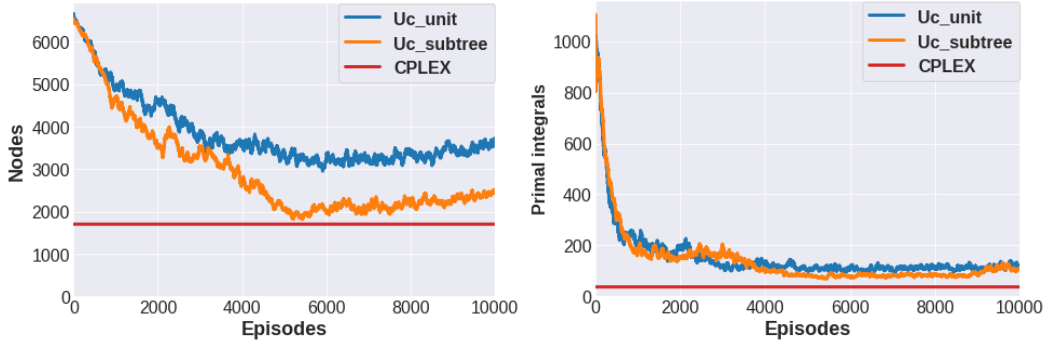
(a) Tree sizes – `micro_asym_T12` (b) Primal integral scores – `micro_asym_T12`



(c) Tree sizes – `micro_bal_T12` (d) Primal integral scores – `micro_bal_T12`



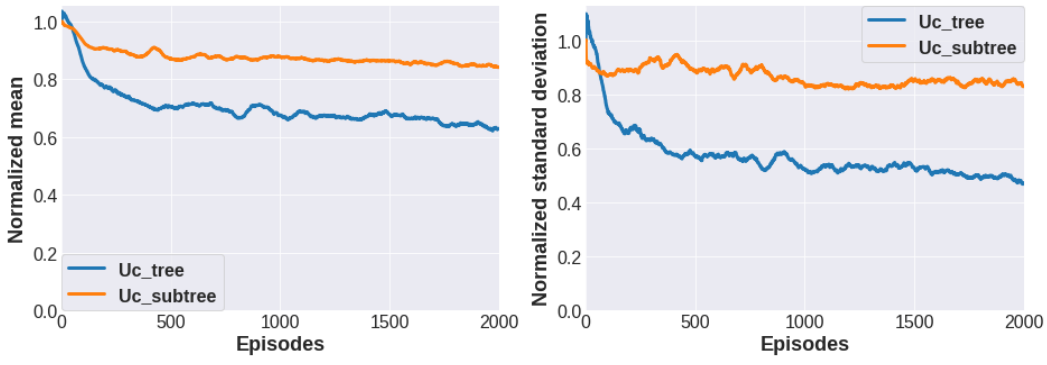
(e) Tree sizes – `hydro_fix_2` (f) Primal integral scores – `hydro_fix_2`



(g) Tree sizes – `hydro_var_2` (h) Primal integral scores – `hydro_var_2`

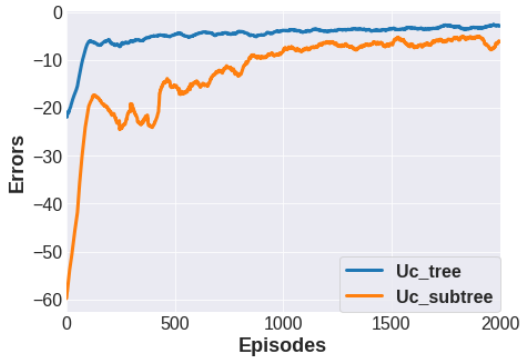
Figure 4.21: Training processes: comparison of the subtree cost model with $\gamma = 0.8$ and $h = 6$ (`Uc_subtree`) against the unitary cost model (`Uc_tree`) under tree-based transitions.

As suggested above, the improvement of the performances may be due to the fact that this new value function exhibits a more stable behaviour during the learning process. Indeed, Figure 4.22 shows that both the mean and standard deviation of the targets are more stable during the learning process, which illustrates that the targets are less dependent on the agent's performances. To elaborate on that matter, Figure 4.23 displays the evolution of these two statistics by depth during the learning process. We see that the subtree cost model allows to standardize the targets along the tree and thus helps to compare actions that may be taken at different depths. Besides, Figure 4.22c highlights that our approximation of the Q-function is biased under both cost models, which does not appear as a subject of concern since the more biased model here gives the best performances.



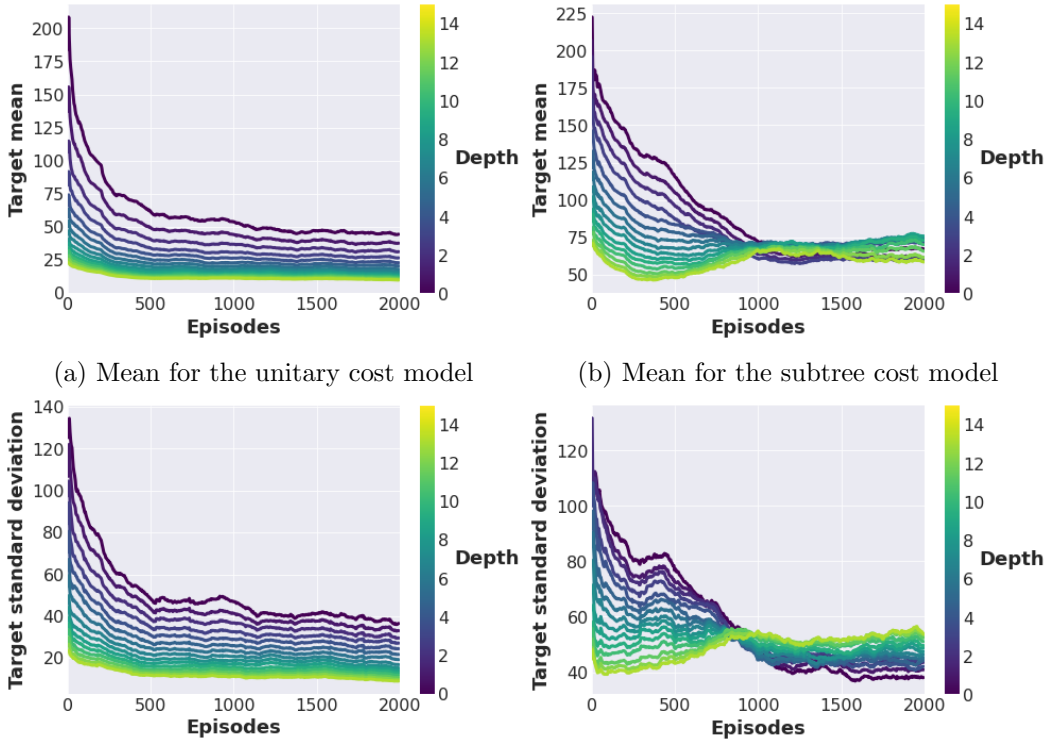
(a) Normalized mean

(b) Normalized standard deviation



(c) Mean errors

Figure 4.22: Evolution of target's statistics and errors during the training process on `micro_asym_T6`.



(a) Mean for the unitary cost model

(b) Mean for the subtree cost model

(c) Standard deviation for the unitary cost model

(d) Standard deviation for the subtree cost model

Figure 4.23: Evolution of target's statistics by depth during the training process on `micro_asym_T6`.

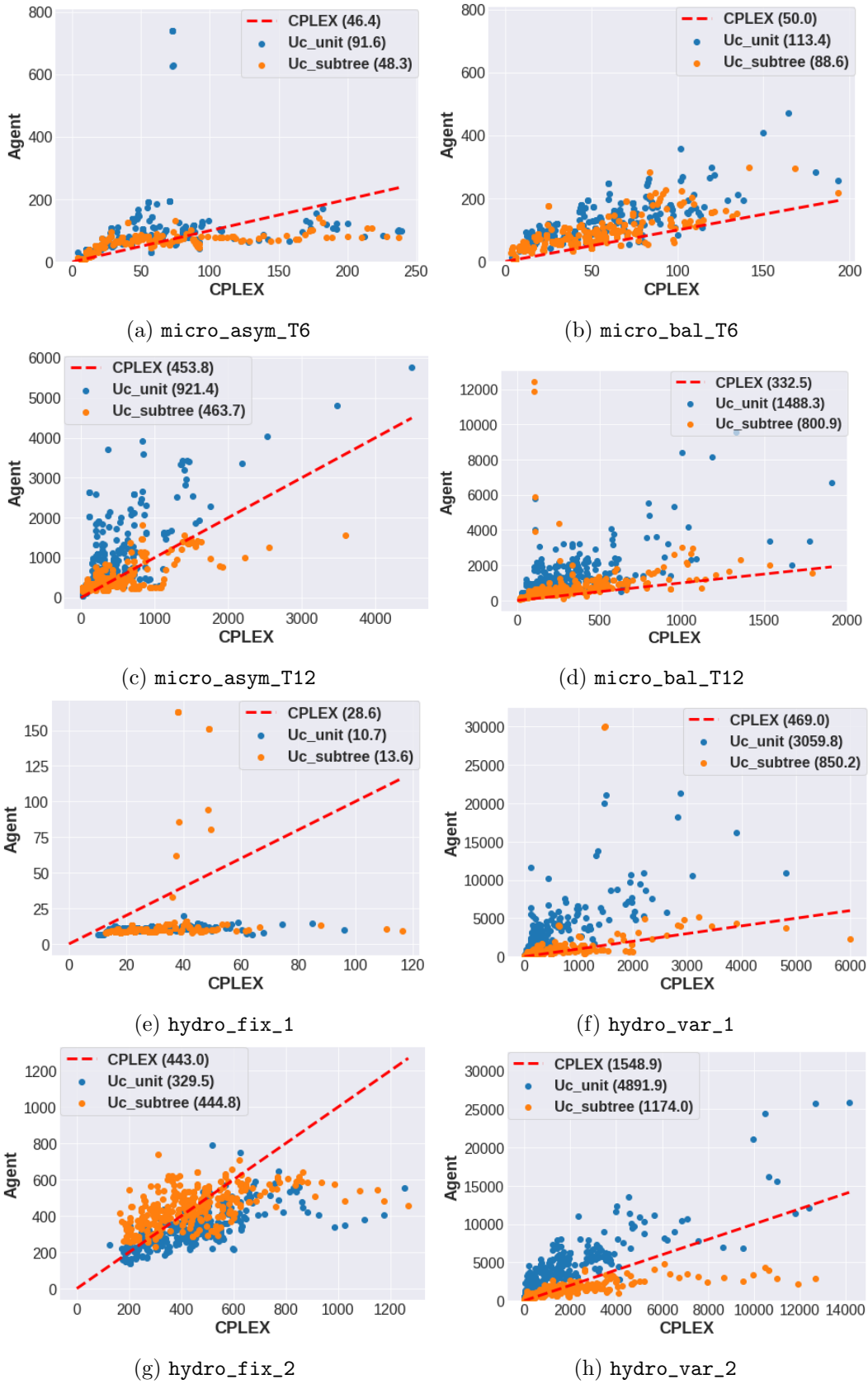
Let us now investigate on the behaviour of the agents on test instances. Figure 4.24 shows the tree sizes produced on test instances by a learnt agent under both the unitary and the subtree cost model. We see that the gains are mostly obtained on the instances on which `CPLEX` has some difficulties. On the contrary, the agents often show relatively low performances on instances easily solved by `CPLEX`. This observation can be explained by the fact that mechanically less samples are available for learning on "easy" instances. An exception of this fact is the case of `hydro_fix_1`, where almost every instances can be solved in less than 10 nodes. Note that trained agents find the according strategies and are able to almost systematically beat `CPLEX` (which is not so trivial as was shown in Figure 4.14c, page 113, where we saw that random strategies for this problems yield large trees).

As suggested by the training processes, we see in Table 4.1 that the subtree cost model generally produces more efficient agents, both in average and when considering only the best agents on train. Agents are generally less efficient on test instances than on train instances, which was expected, but do not show excessive signs of overfitting. We generally compete with the branching strategy

of CPLEX, except on micro_bal_T12. As expected, learning an efficient strategy on this problem is more complicated than on micro_asym_T12, the higher variability in the optimal solutions making the branching dynamics more dependent to instances' data.

	Uc_tree				Uc_subtree			
Problem	Train		Test		Train		Test	
	Mean	Best	Mean	Best	Mean	Best	Mean	Best
micro_asym_T6	+65%	+7%	+93%	+91%	**+21%**	**-4%**	**+38%**	**+4%**
micro_bal_T6	+122%	+64%	+157%	+117%	**+61%**	**+37%**	**+100%**	**+77%**
micro_asym_T12	+222%	+ 92%	+233%	+109%	**+6%**	**-5%**	**+22%**	**+2%**
micro_bal_T12	+602%	+261%	+618%	+358%	**+157%**	**+111%**	**+183%**	**+141%**
hydro_fix_1	**-33%**	-67%	**-35%**	**-63%**	+31%	**-73%**	+36%	-53%
hydro_var_1	+2502%	+480%	+1585%	+570%	**+739%**	**+26%**	**+279%**	**+81%**
hydro_fix_2	**+24%**	**-29%**	**+ 24%**	**-25%**	+335%	-1%	+32%	+0%
hydro_var_2	+433%	+151%	+187%	+187%	**+45%**	**-35%**	**+26%**	**-24%**

Table 4.1: Number of nodes on train and test instances against CPLEX. The performances are displayed in average and for the best agent on train instances over 25 independent training processes.



(a) `micro_asym_T6` (b) `micro_bal_T6`

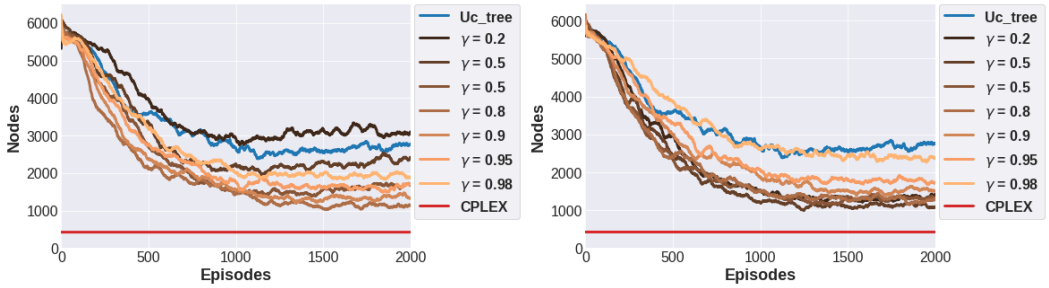
(c) `micro_asym_T12` (d) `micro_bal_T12`

(e) `hydro_fix_1` (f) `hydro_var_1`

(g) `hydro_fix_2` (h) `hydro_var_2`

Figure 4.24: Tree sizes against `CPLEX` on test instances for an individual agent under the subtree cost model with $\gamma = 0.8$ and $h = 6$ (`Uc_subtree`) and the unitary cost model (`Uc_tree`) under tree-based transitions. Number in parentheses are average tree sizes.

Last, Figure 4.25 displays the impact on the parameters γ and h on the performances under the subtree cost model. In this regard, we see that a too high value of gamma makes the agent behave similarly to an agent trained under a unitary cost model, and that an excessively low value makes it too short-sighted. Naturally, low values of gamma penalize more the agent when h is also low.



(a) $h = 3$ (b) $h = 6$

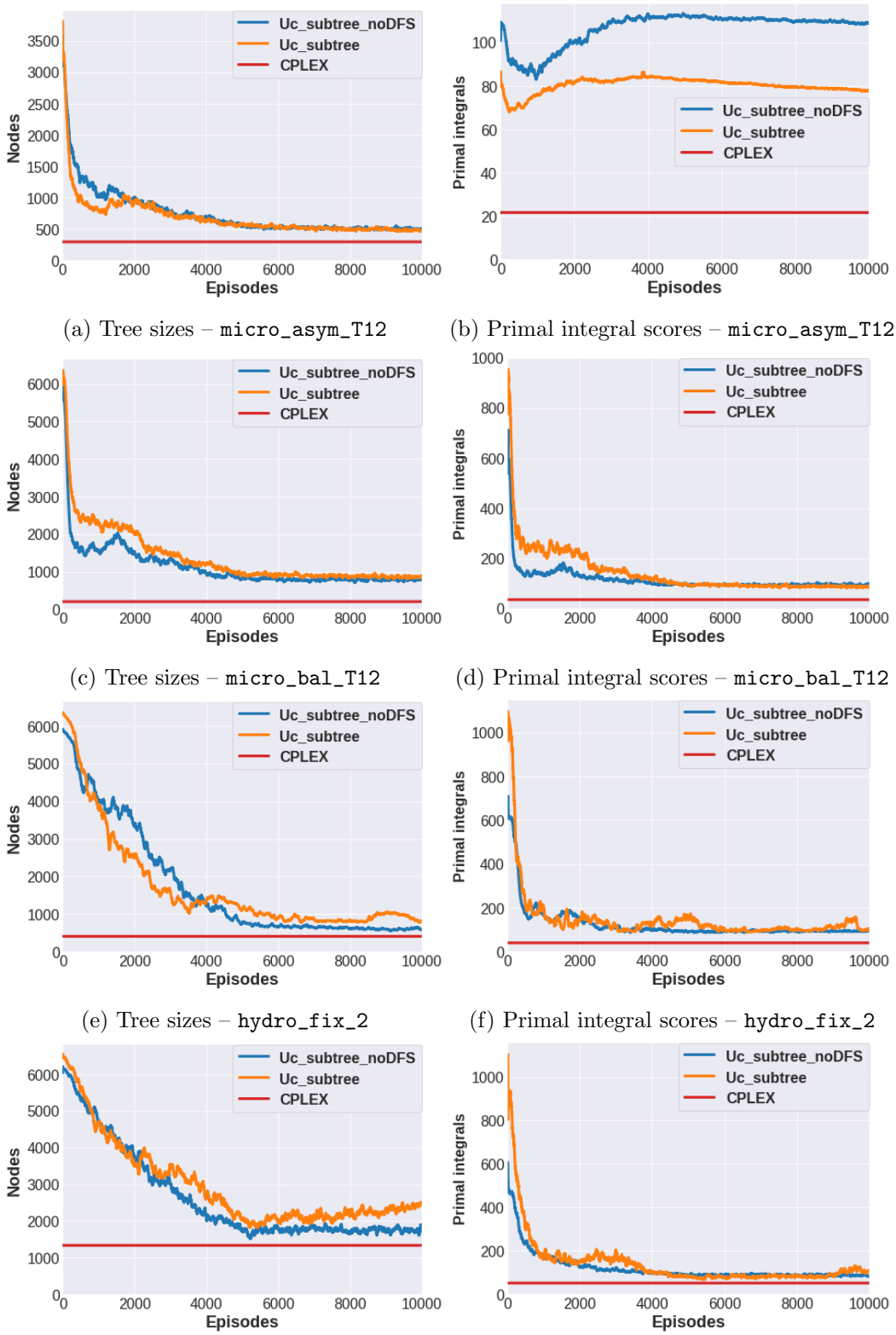
Figure 4.25: Training processes: impact of the discount factor for different values of h in the subtree cost model under tree-based transitions on `hydro_var_1`.

Remark 5. The DFS condition is not restrictive in practice

In this chapter, we used DFS so as to theoretically reconcile oracle strategies and optimal policies for the unitary cost model with tree-based transitions. Nonetheless, minimizing the subtree size is still a relevant objective when the node selection strategy is not DFS. This is illustrated in Figure 4.26, where we observe similar training processes. Performances on test instances, displayed in Table 4.2, confirm these results. Every trained agents are evaluated against `CPLEX` using the same node selection strategy as they do, so as to evaluate the relative performance of the agents compared to `CPLEX` under equal conditions. In other words, an agent under DFS is evaluated against `CPLEX` under DFS and conversely.

	`Uc_subtree_noDFS`				`Uc_subtree`			
Problem	Train		Test		Train		Test	
	Mean	Best	Mean	Best	Mean	Best	Mean	Best
`micro_asym_T12`	+ 66%	+ 35%	+ 88%	+ 95%	**+6%**	**-5%**	**+22%**	**+2%**
`micro_bal_T12`	+ 295%	+ 183%	+ 322%	+ 249%	**+157%**	**+111%**	**+183%**	**+141%**
`hydro_fix_2`	**+ 29%**	**-25%**	**+ 29%**	**-22%**	+ 335%	-1%	+ 32%	+ 0%
`hydro_var_2`	**+ 23%**	**-41%**	**-0%**	**-21%**	+ 45%	-35%	+ 26%	-24%

Table 4.2: Number of nodes on train and test instances against `CPLEX`. The performances are displayed in average and for the best agent on train instances of 25 independent training processes.



(a) Tree sizes – `micro_asym_T12`

(b) Primal integral scores – `micro_asym_T12`

(c) Tree sizes – `micro_bal_T12`

(d) Primal integral scores – `micro_bal_T12`

(e) Tree sizes – `hydro_fix_2`

(f) Primal integral scores – `hydro_fix_2`

(g) Tree sizes – `hydro_var_2`

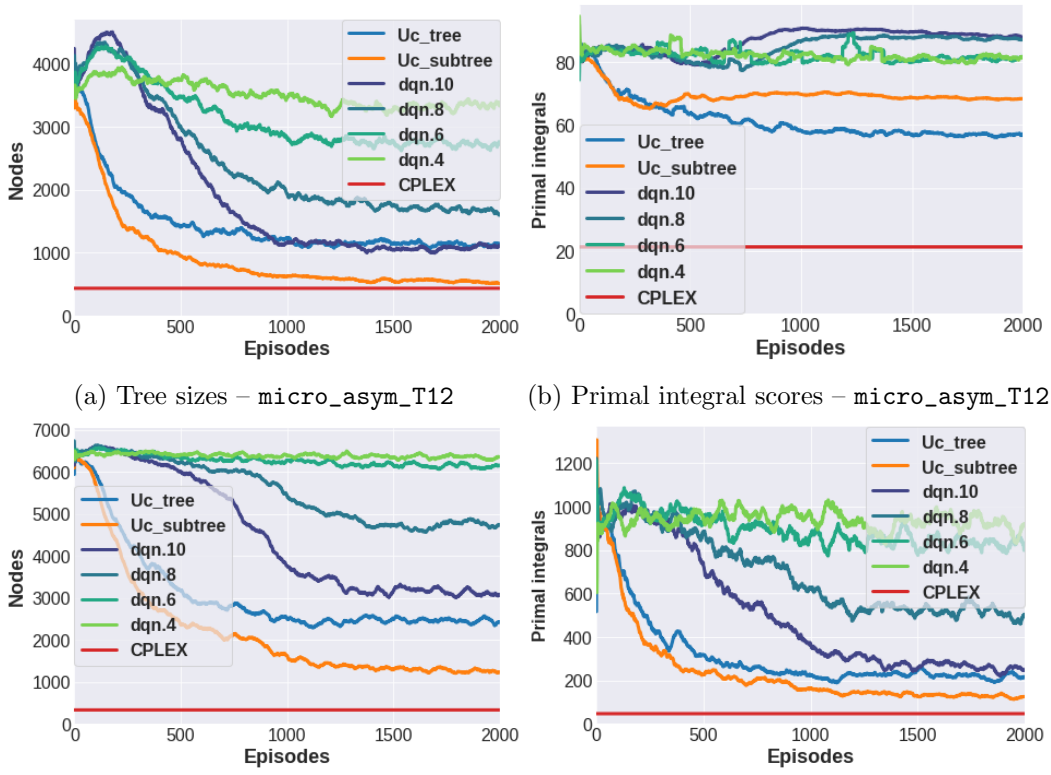
(h) Primal integral scores – `hydro_var_2`

Figure 4.26: Training processes: impact of the DFS restriction in the subtree cost model under tree-based transitions.

4.3 Variations

4.3.1 DQN loss function

When we introduced our RL methodology, we presented our loss function $L_i(\theta_i)$ (4.12) as a counterpart for the classic DQN loss $L_i^{DQN}(\theta_i)$ (4.11) (see page 94). We justified our choice by the deterministic and episodic properties of our environment, making the Q-value observable for the current policy. Figure 4.27 displays training processes when using the loss function L_i^{DQN} for different values of the discount factor γ. We see the interest of our loss function, which learns much more rapidly than its DQN counterpart. This result is due to the fact that DQN's targets are nothing but noise at the beginning of training, which explains the slower slope in the early episodes.



(a) Tree sizes – `micro_asym_T12` (b) Primal integral scores – `micro_asym_T12`

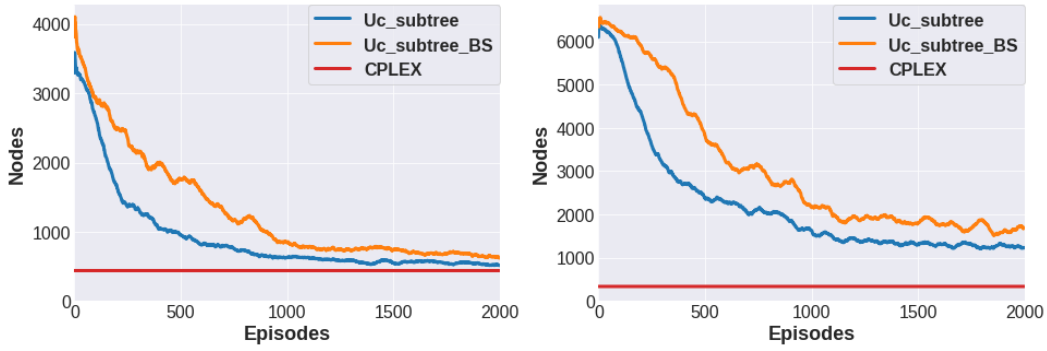
(c) Tree sizes – `micro_bal_T12` (d) Primal integral scores – `micro_bal_T12`

Figure 4.27: Training processes: comparison with the DQN loss function for different values of γ.

4.3.2 State representation, network architecture and training parameters

All the parameters in the results presented in this chapter and in the next two are kept identical to allow a fair comparison between the methods. We briefly describe here the features used as well as some of the hyperparameters, and discuss about different variations explored around them.

As recommended in the literature [65], we use both static (related to instances) and dynamic features (characterizing the progress of the B&B procedure) to represent states. However, we prevent ourselves from using any statistics or inference provided by the solver, as the main idea of this work is to discover strategies independently of the used commercial software.
Regarding static features, we use a 15-dimensional PCA [95] embedding of the instance data, and some standardized statistics on the objective function and constraints. As for dynamic features, they consist in a one-hot encoding of the branching state and additional statistics taken from [70] such as statistics on the number of leaves, open nodes, LP values, *etc.*, without considering solver's dependent scores. To give an idea of the gains obtained from additional dynamic features, we compare in Figure 4.28 agents using only the branching state and static features to agents using the branching state, static and additional dynamic features.



(a) Tree sizes – `micro_asym_T12` (b) Tree sizes – `micro_bal_T12`

Figure 4.28: Training processes: impact of the additional dynamic features. `Uc_subtree_BS` refers to the case where only the branching state and static features are used.

The related question of the neural network architecture has been investigated without much success. For the experiments displayed in this document, our Q-network is a dense architecture composed

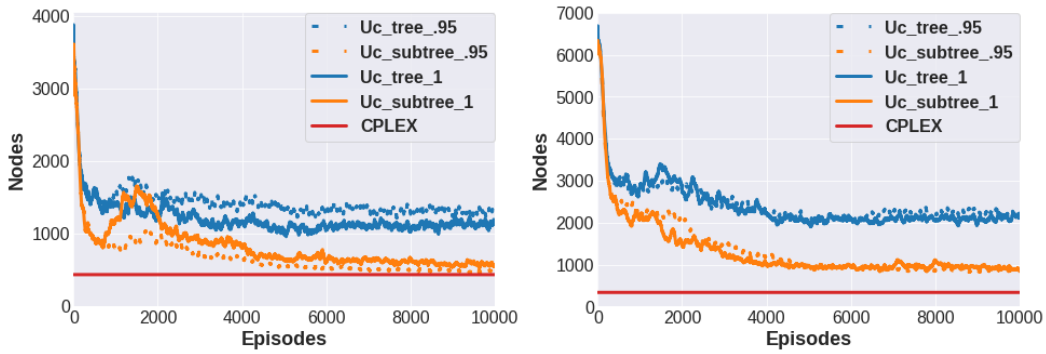
of 4 hidden layers, each one holding $2\,|\mathcal{J}|$ units with selu activation functions. The output layer has $|\mathcal{J}|$ units and linear activation functions. In addition, we attempted to embed the problem's structure in locally dense neural networks using constraint-wise convolutions, without any improvement of the performances. To go further in this direction, it could be interesting to use the bipartite graph convolutional neural network proposed in [68]. Likewise, dueling architectures [96] have been implemented to handle the instance-related target variability without any clear impact on the performances. Regarding the training parameters, we use a decreasing learning rate, a fixed-size buffer and a prioritized experience replay scheme such that samples in the buffer with high prediction errors are more often sampled [41].

Acknowledging that choices are more important near the root node, the results presented in this chapter use a weighting scheme to prioritize the corresponding training samples. Concretely, we associate to each sample the weight

$$\omega(s) \propto \frac{\lambda^{d(s)}}{|\mathcal{T}(s)|}$$

in the loss function (4.12), with $d(s)$ the depth of the B&B node associated to state s and $|\mathcal{T}(s)|$ the size of the tree in which s has been sampled. Experiments are performed using $\lambda = 0.95$. This scheme gives more weight to states near the root node (when $\lambda < 1$) and to states in small trees, as the corresponding instances are mechanically less represented in the training set.

Tuning all the hyperparameters is a hard task, all the more so because their impact varies depending on the configuration. To illustrate this point, we show in Figure 4.29 the impact of setting the λ parameter in the previous weighting scheme. We see that its effect on the training process differs not only depending on the problem but also on the cost model.



(a) Tree sizes – `micro_asym_T12` (b) Tree sizes – `micro_bal_T12`

Figure 4.29: Training processes: impact of the λ parameter in the sample weighting scheme, with unitary (`Uc_tree`) and subtree (`Uc_subtree`) cost models under tree-based transitions. The dashed lines are associated to the value $\lambda = 0.95$ and the solid ones to $\lambda = 1$.

4.3.3 Guiding the exploration with expert's demonstrations

When observing the tree generated during the training process of Algorithm 6, we observe as expected that the agents produce very large trees in the early episodes. As shown in Figure 4.14c, which displays an example of training process on `hydro_fix_1`, this can happen even if the problem is actually easy to solve. This phenomenon has two main drawbacks. Naturally, it slows down the training process. Most importantly, it makes the value function more noisy. For instance, random decisions may have drastic impacts on the subtree size and thus largely alter the value function under the unitary cost model. To alleviate this issue, one idea is to use expert's demonstrations, for instance provided by `CPLEX`, to guide the agent, either at exploration time or at learning time.

Expert's demonstrations at exploration time

An intuitive idea is to use the demonstrations for guiding the exploration. Instead of sampling actions randomly or following the agent's policy, some actions are taken following the expert's decisions with probability ε_i at episode i. Of course, the probability of requesting the expert should be decreasing through episodes, so as to make the agent learn on its own distribution at the end of the training process. This kind of approach has been for instance used in [46].

Expert's demonstrations at learning time

One may also leverage these demonstrations at learning time, considering they allow to get some precious information on non-taken actions. This idea, later used in different contexts, consists in considering that the expert's choices should always be better than those of the agent. When updating the weights of the agent, we use this trick to get a gradient for both the explored actions and the expert's recommendations.

For some state s and agent's weight θ, let us note j the visited action and j^* the expert's demonstration. As only j has been explored, only $Q^{\pi_\theta}(s,j)$ can be observed, which is naturally used to get a gradient from the error $Q^{\pi_\theta}(s,j) - \hat{Q}(s,j;\theta)$. Here, we also use this observation to get a gradient from $Q^{\pi_\theta}(s,j) - \hat{Q}(s,j^*;\theta)$, only in cases where the expert's action is estimated at a higher cost that the visited one, *i.e.* if $\hat{Q}(s,j^*;\theta) > Q^{\pi_\theta}(s,j)$. To this end, we modify the loss function $L_i(\theta_i)$ (see Equation (4.12)) and rather consider

$$\begin{aligned}\tilde{L}_i(\theta_i) =& \mathbb{E}_{s,j,j^*\sim\Delta^i}\left[\left(Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j;\theta_i)\right)^2\right] \\ &+ \mathbb{P}(A)\mathbb{E}_{s,j,j^*\sim\Delta^i}\left[U.\left(Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j^*;\theta_i)\right)^2 | A\right]\end{aligned} \tag{4.22}$$

with $A = (j \neq j^*) \cap \left(\hat{Q}(s,j^*;\theta_i) > Q^{\pi_{\theta_i^-}}(s,j)\right)$ and Δ^i the buffer joint distribution for states, visited actions and expert's recommendations. $U \sim \mathcal{B}(\varepsilon_i)$ is a Bernoulli random variable of parameter ε_i, controlling the probability of using these recommendations.

Differentiating this loss function with respect to the weight vector gives the gradient for iteration i

$$\begin{aligned}\nabla_{\theta_i}\tilde{L}_i(\theta_i) \propto\ & K\nabla_{\theta_i}L_i(\theta_i) \\ &+ \varepsilon_i\mathbb{E}_{s,j\sim\Delta^i}\left[\left(Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j^*;\theta_i)\right)\nabla_{\theta_i}\hat{Q}(s,j^*;\theta_i)\mathbb{1}_A\right]\end{aligned} \tag{4.23}$$

with K some normalizing constant.

Both the approaches have pros and cons. On the one hand, using demonstrations at exploration time should allow to produce smaller trees and thus decrease the noise around the estimation of actions' values, but makes the agent learn on samples outside of its distribution. On the other hand, using them at learning time allows to obtain feedback on both selected and non-selected actions, which should increase the sample efficiency, but also assumes that experts' decisions are better than the visited actions, which may not be true.

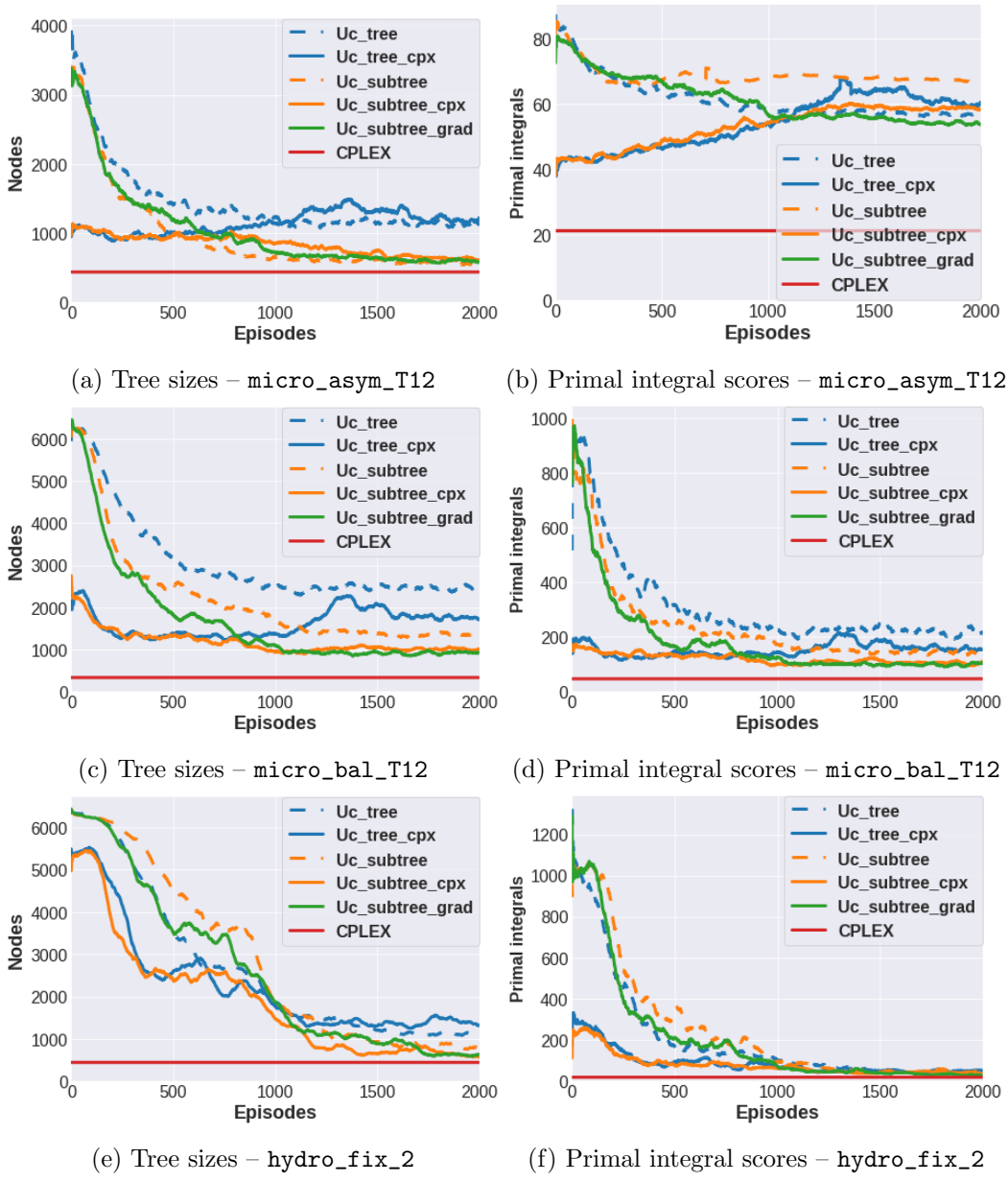
Remark 6. Building its own expert with Monte Carlo Tree Search (MCTS)

An alternative to using `CPLEX` for demonstrations would be to build its own expert, using MCTS [97]. The idea of MCTS is to guide the exploration of the environment through sampling, maintaining a trade-off between exploration and exploitation. We implemented such method without success. The size of the search space and the computational cost prevented us from testing many variations, and more work may be done in this direction in the future.

Experiments

Figure 4.30 shows the results of the two previous approaches. For these experiments, ε_i linearly decreases from 0.35 to 0 and stays null in the last third of the training process, for both methods. As expected, we observe lower tree sizes in the early episodes when the expert is used during exploration as its decisions are better than random moves. However, the long-term gains of using the expert at the beginning of learning tends to be non significant. Especially, we observe that the guided agent under the unitary cost model (`Uc_tree_cpx`) faces a worsening of its performance at some point (this is really salient on Figure 4.30c). We explain this phenomenon by the following argument. We already saw that the agent tends to underestimate the targets, especially in the early phases (see Figure 4.6). When the expert becomes less active in the exploration, the actions are taken following the agent's estimations. At this moment, the less explored actions (probably not so good as less taken by the expert) appear to the agent as producing a lower cost, and thus are selected. One may then wonder why such phenomenon does not appear for the subtree cost model. Our guess is that the more homogeneous targets' distribution (see Figure 4.23) helps the agent to adapt the estimations of new actions.

Regarding the comparison of the two methods for leveraging expert demonstrations, they appear to have similar performances. The advantage of using the expert at exploration time is to produce trees of lower sizes, thus speeding up the training process.



(a) Tree sizes – `micro_asym_T12` (b) Primal integral scores – `micro_asym_T12`

(c) Tree sizes – `micro_bal_T12` (d) Primal integral scores – `micro_bal_T12`

(e) Tree sizes – `hydro_fix_2` (f) Primal integral scores – `hydro_fix_2`

Figure 4.30: Training processes: using expert's demonstrations.
`Uc_tree` (resp. `Uc_subtree`) indicates the use of the unitary (resp. subtree) cost model and `*_cpx` (resp. `*_grad`) refers to the use of `CPLEX` demonstrations at exploration (resp. learning) time.

4.3.4 Bounding the predictions in the unitary cost model

When training an agent using the loss function $L_i(\theta_i)$ (see Equation (4.12)), we already mentioned that information is only propagated in the neural network through the output neurons representing the actions a which has been triggered, thus allowing to evaluate $Q^{\pi_\theta}(s, a)$. Therefore, one does not

have any control on the predictions made by the agent for the non-selected actions. This can lead to the emergence of discarded actions in the case where non-visited actions have durable overestimated value predictions.

Figure 4.31 displays the predictions at the root node for the chosen action on multiple episodes and that of some fixed action b. As we see, this action b would never be picked at the root node if it was not for the epsilon-greedy exploration. In addition, we observe that its predictions may reach high values without any particular good reason since this action has been at best poorly sampled during training.

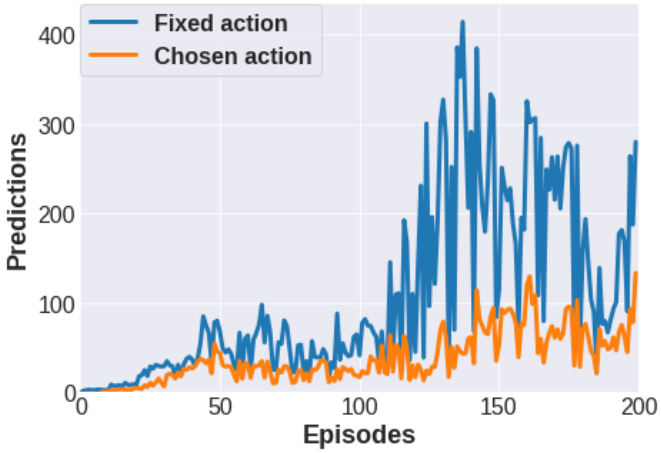


Figure 4.31: Predictions at root node for the chosen minimal Q-valued action and a fixed arbitrary one.

Such observation is valid for any Approximate Q-learning framework, as one needs to select first an action in a given state to collect the associated cost. Thus, it is often recommended to design cost signals as rich as possible. In the following, we explore an idea to control the range of the neural network outputs, using the following proposition.

Proposition 4.3.1. *Let action a be any chosen action at state s and b any other action. Then, under the unitary cost model and tree-based transitions, a node selection strategy exists such that the following inequality holds for the unobserved optimal Q-value of action b and for any given branching policy π:* $Q^*(s,b) \leqslant 1 + 2Q^\pi(s,a)$.

Proof. *Let us call $\mathcal{T}_a$ the generated subtree rooted in node $\zeta \equiv \zeta(s)$ of size $Q^\pi(s,a)$. We must show here that there exists a strategy π^{ub} which, selecting the action b at state s instead of a, fully expands*

the underlying subtree $\mathcal{T}_b$ in less than $1 + 2Q^\pi(s, a)$ nodes. Let us note β the upper bound available at s and distinguish two cases.

- *Case 1: No better bound than β has been found in $\mathcal{T}_a$. In that case, π^{ub} is built by following the policy used in $\mathcal{T}_a$ under the two nodes $D_0(\zeta, b)$ and $D_1(\zeta, b)$, starting by branching on a. Following this strategy under these two nodes, any node $\zeta' \in \mathcal{T}_b$ has a distinct counterpart in $\mathcal{T}_a$, equivalent to the hypothetical node $D_0(\zeta', b)$ (resp. $D_1(\zeta', b)$) when considering ζ' in the descendants of $D_0(\zeta, b)$ (resp. $D_1(\zeta, b)$). Thus, π^{ub} can be defined with a slight abuse of notations by $\pi^{ub}(D_i(s, b)) = \pi(s)$ with $i \in \{0, 1\}$. With this strategy, any node ζ' pruned in $\mathcal{T}_a$ is associated to corresponding nodes in $\mathcal{T}_b$, equivalent to the nodes $D_0(\zeta', b)$ and $D_1(\zeta', b)$ (as no better bound has been found). Thus they are also pruned in $\mathcal{T}_b$. Indeed, if ζ' has been pruned*

  (i) by infeasibility, then $D_0(\zeta', b)$ and $D_1(\zeta', b)$ are also pruned by infeasibility since their feasibility set is included in that of ζ';

  (ii) by bound, then $D_0(\zeta', b)$ and $D_1(\zeta', b)$ are also pruned by bound since, for the same reason, their relaxed optimal values are greater or equal than that of ζ';

  (iii) by integrality, then $D_0(\zeta', b)$ and $D_1(\zeta', b)$ are pruned by bound. Indeed, as no bound better than β has been found in $\mathcal{T}_a$, the node ζ' can then also be pruned by bound. The point is then proven using the same argument as in (ii).

- *Case 2: A new bound β_a has been found in $\mathcal{T}_a$, given by the solution x^* at node ζ_a. In this setting, we build π^{ub} exactly as in Case 1, except that we impose the first node selection decisions to lead to the node ζ_a – in particular, this is achieved by visiting first the node $D_{x_b^*}(\zeta, b)$. Doing so, we get the same new bound β_a as in $\mathcal{T}_a$ and can apply the same reasoning as in Case 1.*

In both cases, we exhibited strategies which lead to build trees smaller than $\mathcal{T}_a$ (or with the same size) under $D_0(\zeta, b)$ and $D_1(\zeta, b)$. Counting the initial node ζ, we obtain the proposed bound of $1+2Q^\pi(s, a)$.

□

Note in the previous proof that, for the inequality to hold, one needs to take control of the node selection strategy if a primal bound has been found under the current node. Omitting this point, we intend to provide some gradient on non-taken actions so as to force the predictions to respect this

bound. To this end, we use the same trick as in the previous Section and modify the loss function $L_i(\theta_i)$ to rather consider

$$\begin{aligned}\tilde{L}_i(\theta_i) =& \mathbb{E}_{s,j\sim\Delta^i}\left[\left(Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j;\theta_i)\right)^2\right] \\ &+ \sum_{j'\in\mathcal{J}\setminus\{j\}} \mathbb{P}(A_{j'})\mathbb{E}_{s,j\sim\Delta^i}\left[\left(1 + 2Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j';\theta_i)\right)^2 \big| A_{j'}\right]\end{aligned} \tag{4.24}$$

with $A_{j'} = \left(1 + 2Q^{\pi_{\theta_i^-}}(s,j) < \hat{Q}(s,j';\theta_i)\right)$.
Differentiating this loss function with respect to the weight vector gives the gradient for iteration i

$$\begin{aligned}\nabla_{\theta_i}\tilde{L}(\theta_i) \propto\ & K\nabla_{\theta_i}L_i(\theta_i) \\ &+ \sum_{j'\in\mathcal{J}\setminus\{j\}} \mathbb{E}_{s,j\sim\Delta^i}\left[\left(1 + 2Q^{\pi_{\theta_i^-}}(s,j) - \hat{Q}(s,j';\theta_i)\right)\nabla_{\theta_i}\hat{Q}(s,j';\theta_i)\mathbb{1}_{A_{j'}}\right]\end{aligned} \tag{4.25}$$

with K some normalizing constant.
Equipped with this loss function, the agent receives feedbacks on non-taken actions as soon as their estimations violate the bound for the considered state. This loss function is supposed to increase the competition between actions and diminish the risk of discarding unfortunately some of them. Figure 4.32 shows the resulting training process, with and without enforcing the previous bound. Results under the subtree cost model are also displayed, even if the bound does not theoretically hold for the corresponding Q-function. The results are disappointing, and the agent seems to barely improve during the process. Figure 4.33 shows the predictions at root nodes for each variable, with or without enforcing the bound. It explains the bad results observed in Figure 4.32: equipped with the loss function (4.24), the agent is not able to discriminate actions anymore. A potential cause for this phenomenon could be a too high magnitude for the gradients given to non-taken actions, which could be mitigated by tuning the normalizing constant K. Note that this behaviour may also be induced by the fact that our estimator for the Q-function is biased (see Figure 4.22c).

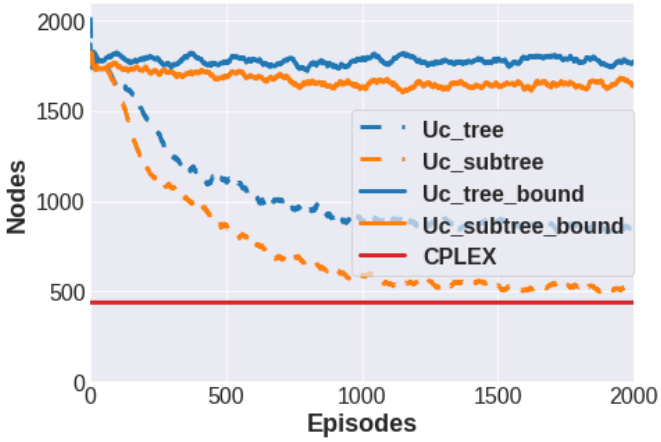
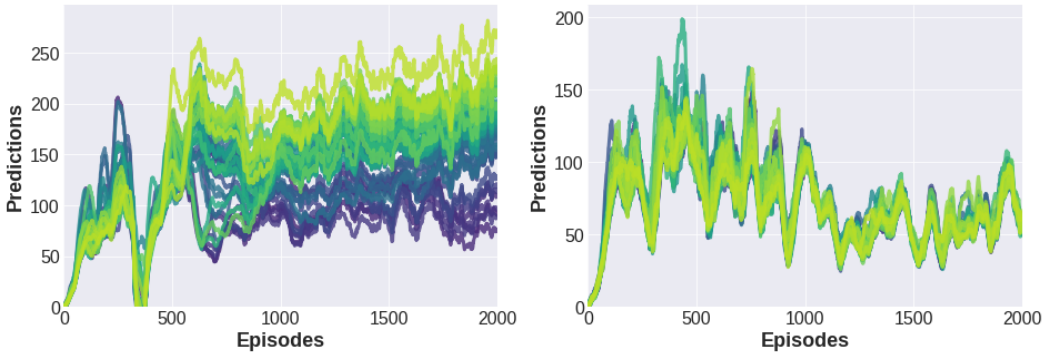


Figure 4.32: Training processes: failure of learning when bounding the predictions (`*_bound`).



(a) Without predictions' bounding

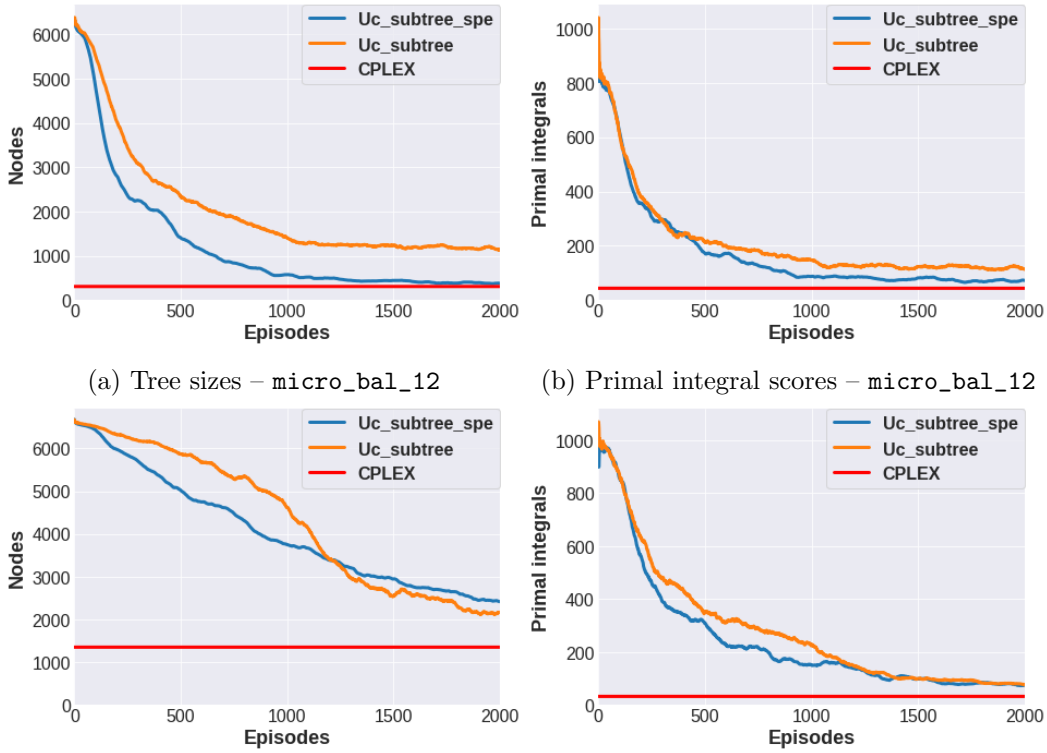
(b) With predictions' bounding

Figure 4.33: Example of predictions for each variable at the root node under the unit tree model, with or without predictions' bounding.

4.3.5 Some room for improvement

To conclude, we measure the room for improvement of the proposed method. To do so, we place ourselves in the idealized case where one agent is specialized on a unique instance. Figure 4.34 compares the average training process of these specialized agents with that of a single agent learning on the whole set of instances. The results are presented on the two most difficult problems considered in this chapter, that is `micro_bal_12` and `hydro_var_2`. We see that the strategies discovered by the specialized agents are clearly outperforming that of the generic agent on `micro_bal_12`. This observation is mitigated by the results obtained on `hydro_var_2`. On this problem, we see that specialization does not allow to discover more effective policies. To ameliorate the efficiency of the search, many levers may be

triggered, such as improving the state representation, the network architecture, the hyperparameters, the exploration method, the learning process... A plethora of ideas still remain to be explored.



(a) Tree sizes – `micro_bal_12`

(b) Primal integral scores – `micro_bal_12`

(c) Tree sizes – `hydro_var_2`

(d) Primal integral scores – `hydro_var_2`

Figure 4.34: Training processes: comparison of a single agent learning on the whole set of instances with specialized agents (`Uc_subtree_spe`).

Chapter 5

Learning the Node Selection Strategy

Content

In this chapter, the objective is similar to that of the previous one: learning a strategy in a B&B algorithm so as to perform well, on average, for a given MILP problem.
Here, we aim at learning the node selection strategy in Algorithm 4. The main difference with the previous chapter will lie in the fact that, under certain conditions, one can derive an expensive but tractable oracle strategy for tree size minimization. Its derivation is the purpose of Section 5.1. As this strategy cannot be used at test time, Section 5.2 proposes different ways of learning its choices, either by supervised learning or reinforcement learning. Last, Section 5.3 presents the performances of the proposed approaches.

Before diving into the specifics, it is worth highlighting that the tree size is generally less sensitive to node selection than to the branching strategy. This is illustrated in Figure 5.1, where we show the distribution of tree sizes generated by random node selection strategies on a given instance, and compare it with the heuristic used in the previous chapter and random branching strategies.

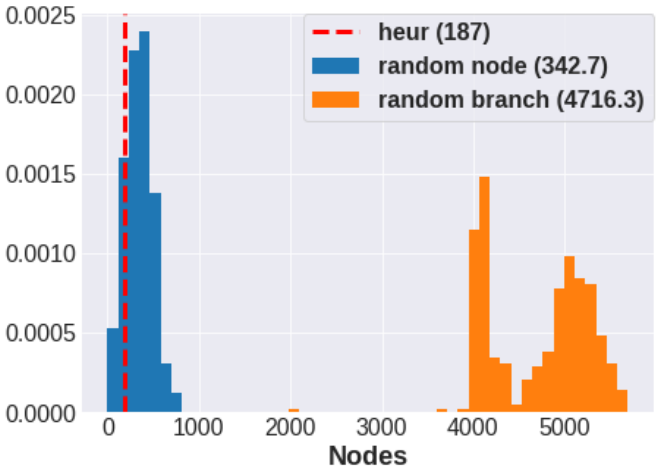


Figure 5.1: Histogram of tree sizes generated by 500 random node selection policies on an instance of `micro_bal_T12`. Branching is performed by `CPLEX` in both cases, and values in parentheses are the mean tree size for the random policies and the tree size for the heuristic. The histogram of random branching performances is also shown for the comparison.

5.1 An oracle strategy for tree size minimization

We still consider a B&B algorithm as described in Algorithm 4. Keeping the previous notations, the search space Π in Equation (3.2) now denotes the set of node selection strategies. Discovering a high-

performance policy in this set will turn out to be easier than learning to branch for a fundamental reason: we know what makes a policy efficient. In the following, we explicit the exact conditions sufficient to derive an oracle node selection strategy.

5.1.1 A simplification hypothesis

In addition to the B&B setting defined by Algorithm 4, we make a specific assumption regarding the branching policy, stated in Hypothesis 5.1.1.

Hypothesis 5.1.1. *Deterministic branching hypothesis*
The branching policy is deterministic and only depends on the set of branching constraints associated to the current node.

Hypothesis 5.1.1 states that for a given B&B node, the branching policy is always the same, independently from the iteration in Algorithm 4, the current primal bound, etc. As a consequence, the node selection does not impact the set of direct children for a given non-leaf node. Concretely, let $\pi_1, \pi_2 \in \Pi$ be two node selection strategies: for any non-leaf node $\zeta \in \mathcal{N}^{\pi_1} \cap \mathcal{N}^{\pi_2}$, we have $\mathcal{D}^{\pi_1}(\zeta) = \mathcal{D}^{\pi_2}(\zeta)$.

Note that Hypothesis 5.1.1 allow for simplifications but may not be verified in practice, for instance when the branching policy depends on the history of decisions and observations made before processing the current node in the tree. Likewise, stochastic policies violate this assumption. Note also that a similar assumption was made in the previous chapter (Hypothesis 4.1.1). Fundamentally, these hypotheses allow to prevent cases where the non-controlled policy is built either stochastically or in an adversarial fashion, for instance selecting the worst decisions only when we take the best ones.

5.1.2 Restriction of the search space

The two main implications of Hypothesis 5.1.1 are given by Propositions 5.1.1 and 5.1.2, which will allow us to reduce drastically the search space.

Proposition 5.1.1. *Under Hypothesis 5.1.1, the node selection strategy does not impact the B&B tree size once the optimal solution is found.*

Proof. *Let π_1, π_2 be two node selection strategies and $\beta_t^{\pi_1}, \beta_t^{\pi_2}$ their primal bounds at iteration t. The two strategies are assumed to be identical until the optimal objective value β^* is found at iteration t^*.*

Considering a B&B node ζ such that $\zeta \in \mathcal{N}^{\pi_1} \cap \mathcal{N}^{\pi_2}$, let us show that as soon as $\zeta' \in \mathcal{D}^{\pi_1}(\zeta)$ is visited at iteration $t_1 > t^*$ following strategy π_1, we have $\zeta' \in (\mathcal{N}^{\pi_1} \cap \mathcal{N}^{\pi_2}) \cup (\mathcal{L}^{\pi_1} \cap \mathcal{L}^{\pi_2})$.
$\zeta' \in \mathcal{D}^{\pi_1}(\zeta) \implies \zeta' \in \mathcal{N}^{\pi_1} \cup \mathcal{L}^{\pi_1}$. But according to Hypothesis 5.1.1, we have $\mathcal{D}^{\pi_1}(\zeta) = \mathcal{D}^{\pi_2}(\zeta) \implies \zeta' \in \mathcal{D}^{\pi_2}(\zeta) \implies \zeta' \in \mathcal{N}^{\pi_2} \cup \mathcal{L}^{\pi_2}$. Let t_2 be the iteration at which ζ' is visited following π_2.

- *In case $\zeta' \in \mathcal{N}^{\pi_1}$, ζ' is neither MILP-feasible nor LP-infeasible, nor pruned by bound in $\mathcal{T}^{\pi_1}$. But $\beta_{t_1}^{\pi_1} = \beta_{t_2}^{\pi_2} = \beta^*$ as $t_1 > t^*$ and $t_2 > t^*$. Then ζ' is not pruned by bound in $\mathcal{T}^{\pi_2}$, so $\zeta' \notin \mathcal{L}^{\pi_2} \implies \zeta' \in \mathcal{N}^{\pi_2}$.*

- *In case $\zeta' \in \mathcal{L}^{\pi_1}$, if ζ' is MILP-feasible or LP-infeasible, then $\zeta' \in \mathcal{L}^{\pi_2}$. If ζ' is pruned by bound in $\mathcal{T}^{\pi_1}$, so it is in $\mathcal{T}^{\pi_2}$ by the same argument as previously and $\zeta' \in \mathcal{L}^{\pi_2}$.*

By recurrence, the two strategies end up with the same B&B tree. □

The immediate interest of such observation is the following. As the node selection strategy has no impact on proving optimality, it implies that one could learn only on the choices made before reaching the optimal solution, which could theoretically allow to reduce drastically the exploration cost by stopping trees' expansion early during training. We elaborate on this point when considering learning methods.

A second consequence of Hypothesis 5.1.1 is that one can safely focus on the set of DFS node selection strategies to find an oracle policy, *i.e.* which produces a tree of minimal size, as stated in Proposition 5.1.2. This result justifies our choice to only consider DFS node selection strategies in the following. Of course, plenty of such strategies should turn out to perform poorly, but our hope is here to discover, among these policies, one which is optimal or, at least, near optimal.

Proposition 5.1.2. *Under Hypothesis 5.1.1, one can always build a B&B tree of minimal size following a DFS node selection strategy.*

Proof. *Let $\mathcal{T}^{\pi} = \{\zeta_0, \zeta_1, \dots, \zeta_T\}$ be a B&B tree built using a node selection policy π, ζ_t representing the t^{th} visited node. Notations are consistent with Algorithm 4 and $\mathcal{N}^{\pi}, \mathcal{L}^{\pi}$ form the corresponding partition of $\mathcal{T}^{\pi}$.*

Exhibition of a better strategy *Let us build a DFS policy π' which will perform at least as good as π. We write $\zeta_{\varphi(t)}$ a node visited at time t following π', and β_t^{π} (resp. $\beta_t^{\pi'}$) the primal bound at iteration t following π (resp. π').*

As $\mathcal{T}^{\pi}$ is fully expanded, there exists a node $\zeta_j \in \mathcal{L}^{\pi}$ such that an optimal MILP solution is found at ζ_j, which depth is noted $d(\zeta_j) \equiv d^*$. Then, one can build a first dive $\left\{\zeta_{\varphi(0)}, \zeta_{\varphi(1)}, \dots, \zeta_{\varphi(d^*)}\right\}$ such that $\varphi(0) = 0$, $\varphi(d^*) = j$, and $\zeta_{\varphi(i+1)} \in \mathcal{D}^{\pi}(\zeta_{\varphi(i)})$ for all $i < d^*$. Note that this is possible due to Hypothesis 5.1.1, which ensures branching consistency between π and π'. This first dive gives $\left\{\zeta_{\varphi(0)}, \dots, \zeta_{\varphi(d^*-1)}\right\} = \mathcal{N}^{\pi'}_{d^*} \subset \mathcal{N}^{\pi}$, $\left\{\zeta_{\varphi(d^*)}\right\} = \mathcal{L}^{\pi'}_{d^*+1} \subset \mathcal{L}^{\pi}$ and $\beta_t^{\pi'} \leqslant \min\left\{\beta_{t'}^{\pi},\ t' \leqslant T\right\}$ for any $t > d^*$. The tree $\mathcal{T}^{\pi'}$ corresponding to policy π' will then be built by making the following node selections tt iteration t:

- *if $t \leqslant d^*$: set $\varphi(t)$ as previously defined;*
- *if $t > d^*$: set $\varphi(t) \in \arg\max_{\zeta} \left\{d(\zeta) | \zeta \in \mathcal{O}^{\pi'}_{t-1}\right\}$;*
- *select node $\zeta_{\varphi(t)}$.*

By construction, this strategy satisfies DFS. Hence, we just need to prove that $\left|\mathcal{T}^{\pi'}\right| \leqslant |\mathcal{T}^{\pi}|$. Let us prove that $\mathcal{T}^{\pi'} \subseteq \mathcal{T}^{\pi}$ by using the same arguments as in the proof of Proposition 5.1.1 from a different starting point.

Set of non-leaf nodes *First, let us show by induction that $\mathcal{N}^{\pi'} \subset \mathcal{N}^{\pi}$.*

Initialization: ζ_0 is the root node for both strategies, so it belongs to $\mathcal{N}^{\pi'} \cap \mathcal{N}^{\pi}$ – the case $\zeta_0 \in \mathcal{L}^{\pi'} \cap \mathcal{L}^{\pi}$ is trivially omitted.

Induction hypothesis: let us assume that a node ζ exists such that $\zeta \in \mathcal{N}^{\pi'} \cap \mathcal{N}^{\pi}$.

Heredity: let us show that if $\zeta_{\varphi(t)} \in \mathcal{D}^{\pi'}(\zeta)$ belongs to $\mathcal{N}^{\pi'}$, then $\zeta_{\varphi(t)} \in \mathcal{N}^{\pi}$. Since the case $\zeta_{\varphi(t)} \in \left\{\zeta_{\varphi(k)}\right\}_{k \leqslant d^*}$ has been trivially built, we omit it in the following and consider $\zeta_{\varphi(t)}$ visited in $\mathcal{T}^{\pi'}$ at a time $t > d^*$.

According to Hypothesis 5.1.1, $\zeta_{\varphi(t)} \in \mathcal{D}^{\pi}(\zeta)$, so $\zeta_{\varphi(t)} \in \mathcal{N}^{\pi} \cup \mathcal{L}^{\pi}$. Let us show that $\zeta_{\varphi(t)} \notin \mathcal{L}^{\pi}$. $\zeta_{\varphi(t)} \in \mathcal{N}^{\pi'}$, so $\zeta_{\varphi(t)}$ is neither MILP-feasible nor LP-infeasible. Besides, it was not pruned by bound in $\mathcal{T}^{\pi'}$, so it cannot be pruned by bound in $\mathcal{T}^{\pi}$ as $\beta_t^{\pi'} \leqslant \beta^{\pi}_{\varphi(t)}$ $(t > d^*)$. Then $\zeta_{\varphi(t)} \notin \mathcal{L}^{\pi}$, which implies that $\zeta_{\varphi(t)} \in \mathcal{N}^{\pi}$.

Set of leaf nodes *We now show that* $\mathcal{L}^{\pi'} \subset \mathcal{N}^{\pi} \cup \mathcal{L}^{\pi}$.
Let $\zeta \in \mathcal{L}^{\pi'}$. *By construction, there exists a node* $\zeta' \in \mathcal{N}^{\pi'}$ *such that* $\zeta \in \mathcal{D}^{\pi'}(\zeta')$. *As we just showed,* $\mathcal{N}^{\pi'} \subset \mathcal{N}^{\pi}$ *so* $\zeta' \in \mathcal{N}^{\pi}$. *Due to the Hypothesis 5.1.1, we have that* $\zeta \in \mathcal{D}^{\pi}(\zeta')$, *hence* $\zeta \in \mathcal{N}^{\pi} \cup \mathcal{L}^{\pi}$.

Conclusion *We proved that* $\mathcal{N}^{\pi'} \cup \mathcal{L}^{\pi'} \subseteq \mathcal{N}^{\pi} \cup \mathcal{L}^{\pi}$ *and thus* $\mathcal{T}^{\pi'} \subseteq \mathcal{T}^{\pi}$. *Since* $\mathcal{N}^{\pi'} \cap \mathcal{L}^{\pi'} = \varnothing$, *we have the desired result* $\left|\mathcal{T}^{\pi'}\right| \leqslant |\mathcal{T}^{\pi}|$.
As a consequence, one can always build a DFS strategy which produces a tree at least as small as that of any given node selection strategy. Applying such construction to an optimal node selection strategy justifies the result. □

Before discussing the advantages of the restriction to strategies obeying to DFS, let us first define the considered MDP $< \mathcal{S} \times \mathcal{J}, \mathcal{A}, T, C, \gamma >$ corresponding to a DFS node selection policy. We use consistent notations with respect to the MDP presented in the previous chapter (see Section 4.1.2) to facilitate comparisons.

- a finite set of states $\mathcal{S} \times \mathcal{J}$: a state $(s_t, j_t) \in \mathcal{S} \times \mathcal{J}$ is a tuple $((p, t, \mathcal{H}_t), j_t)$ where t is an iteration of Algorithm 4 and $\mathcal{H}_t$ is the history of any decisions or observations made so far. A B&B node $\zeta(s_t)$ is associated to any state (s_t, j_t) and corresponds to the node visited at iteration t. $j_t \in \mathcal{J}$ is the output of the branching strategy for the current node, left to the solver.

- a set of actions $\mathcal{A} = \{0, 1\}$. Under DFS in Algorithm 4, the node selection strategy at iteration t comes down to either selecting the unique deepest node in $\mathcal{O}_t^{\pi}$ when $\zeta_{t-1} \in \mathcal{L}_t^{\pi}$, or choosing one of the two children of the last visited node ζ_{t-1}. If $\zeta_{t-1} \in \mathcal{N}_t^{\pi}$ and j_{t-1} is the outcome of an exogenous branching strategy, the action $a_t = 0$ (resp. $a_t = 1$) will correspond to the selection of the child node $D_0(\zeta_{t-1}, j_{t-1})$ (resp. $D_1(\zeta_{t-1}, j_{t-1})$). When the deepest node is unique, that is to say when a leaf node is visited at iteration $t-1$, selection is not considered as an action but rather as part of the transitions. Indeed, the action is totally determined by the DFS requirement. Thus, we will only consider states corresponding to non-leaf nodes in the following.

- a transition function $T(s_t, a_t)$ assumed to be deterministic (and thus markovian): the next node to visit is fully determined by the action as leaf assessment is deterministic, and so is the branching policy by Hypothesis 5.1.1. Again, this transition would also be markovian by

construction without any assumption on branching as $\{j_t\} \cup \mathcal{H}_t \subset \mathcal{H}_{t'}$ for any $t < t'$. Either trajectory-based or tree-based transitions may be considered.

- a discount factor $\gamma \in [0, 1]$.
- a bounded cost function $c : \mathcal{S} \times \mathcal{J} \times \mathcal{A} \to \mathbb{R}$ which depends on the objective set for the strategy to be learnt.

The advantages of enforcing DFS are twofold.
First, as shown when constructing the MDP, it reduces the action set to a set of cardinal 2, which decreases drastically the search space $\mathcal{S} \times \mathcal{J} \times \mathcal{A}$. Note that it also allows the action set to be fixed, which would not trivially be the case otherwise. Indeed, without any assumption, we would have a varying action set $\mathcal{A}_t = \mathcal{O}_t$ as in [76].
Second, using DFS will make it possible to reduce the credit assignment problem, exactly as in the previous chapter (see Section 4.2.2), by confining an action's effect on a subset of the search space – concretely the subtree.

5.1.3 Exhibition of an oracle strategy

In this setting, it is possible to build an oracle strategy as stated in Proposition 5.1.3, whenever the problem has a unique optimal solution x^*. Such a strategy is obtained by first performing a dive towards x^*, and more generally due to Proposition 5.1.1 the following policy π^* is an oracle strategy:

$$\pi^* : \begin{cases} \mathcal{S} \times \mathcal{J} \to \mathcal{A} = \{0, 1\} \\ (s_t, j_t) \mapsto \pi^*(s_t, j_t) = x^*_{s_t}(j_t) \end{cases} \tag{5.1}$$

where $x^*_{s_t}$ is an optimal solution of the sub-problem associated to s_t.

Proposition 5.1.3. *Under Hypothesis 5.1.1, executing first the dive to the optimal solution x^* is an oracle strategy provided the optimal solution is unique.*

Proof. *It directly comes from the proof of Proposition 5.1.2 as the mentioned π' is a direct dive toward the unique optimal solution. Note here that the result does not hold if we do not have the uniqueness of the solution, and diving to an optimal solution of minimal depth is not sufficient to ensure the minimality of the tree (think of a tree where diving first to an optimal solution at depth d allows to entirely prune the branch of an optimal solution at depth $d-1$). However, one can show that a tree built from such strategy has at worst n more nodes than the minimal tree.* □

Even though the statement of Proposition 5.1.3 is simple and intuitive, finding such a strategy is NP-hard as it requires to actually solve the considered MILP instance. As a consequence, this strategy cannot be used directly, since there is no interest in producing a minimal tree for solving a problem if it has already been solved. However, assessing if an action is an oracle choice is expensive but tractable off-line, as it only requires to check if it can lead to an optimal solution for the corresponding sub-problem. At worst, such verification costs one call to the B&B procedure and may thus be leveraged for learning. This is an important difference with Chapter 4 regarding learning the branching policy, where assessing the optimality of a choice (in the sense of the minimization of the tree) would have required an exponential number of B&B calls.

A parallel may be drawn between the oracle defined in Equation (5.1) and the BFS node selection strategy – see Section 2.3.1. BFS selects the open node with the lowest LP value (in the minimization case), the idea being to obtain early a good primal bound. The oracle strategy aforementioned has the same objective, except that it knows exactly where the best primal bound can be found. In the following, we propose different approaches for learning oracle strategies, with or without using demonstrations from the exhibited oracle.
At this point, it may be relevant to point out that the oracle strategy is not necessarily unique, whether DFS is enforced or not. Indeed, first visiting nodes which cannot be pruned by bound does not harm the tree size.

5.2 Learning approaches

As proposed in [75, 76], behavioral cloning and dataset aggregation are two imitation learning paradigms appropriate for learning a node selection strategy from demonstrations. We compare these approaches with a RL alternative and leverage the binary nature of our action space and propose a way of using demonstrations in RL to increase the sample efficiency.

According to Proposition 5.1.1 and acknowledging that choices are more important near the root node than in deeper nodes, we use the following weighting scheme when learning from training samples

$$\omega_\delta(s_t) = \begin{cases} \delta^{d(s_t)} & \text{if } \beta_t > \beta^* \\ 0 & \text{if } \beta_t = \beta^* \end{cases} \tag{5.2}$$

with $\delta \in [0, 1]$ a hyperparameter and $d(s_t)$ the depth of the node associated to state (s_t, j_t). In this section, ϕ refers to a state embedding function.

5.2.1 Behavioral cloning

Behavioral cloning (see Section 2.1.2) simply consists in learning the behavior of the oracle on its own distribution. As the oracle policy π^* follows the optimal solution x_s^* of the sub-problem associated to state s, we directly learn a mapping $f_\theta : \phi(\mathcal{S}) \to [0, 1]^n$ between our embedding space and these solutions, *e.g.* parametrized by a feed forward neural network.

In this setting, behavioral cloning is then reduced to a classic multivariate supervised classification task on oracle first dives, the target classes being the optimal solution in $\{0, 1\}^n$. It can be addressed by ERM (see Section 2.1.1) using the weighting scheme (5.2). The weight vector θ is then calibrated following the program

$$\min_{\theta \in \Theta} \sum_{p \in D_{tr}} \sum_{s_t, j_t \in \Delta^*(p)} \omega_\delta(s_t) l\left(f_\theta(\phi(s_t)), x^*_{s_t}\right) \tag{5.3}$$

with l a convex loss. $\Delta^*(p)$ denotes here the set of visited states by the oracle strategy on instance p before reaching the optimal solution. Note that, as the oracle knows the optimal solution, a unique target (an optimal solution) is associated to each state of a same tree. Hence, one could instead simply learn a mapping from the instances to their optimal solutions.
Seeing Δ^* as a probability distribution on $\mathcal{S}$, the theoretical equivalent of (5.3) is

$$L(\theta) = \mathbb{E}_{s_t, j_t \sim \Delta^*}\left[\omega_\delta(s_t) l\left(f_\theta(\phi(s_t)), x^*_{s_t}\right)\right] \tag{5.4}$$

At testing time, the agent will then act according to the strategy

$$\pi_\theta(s_t, j_t) = \arg\min_{\nu \in \{0;1\}} |f_\theta(\phi(s_t))(j_t) - \nu| \tag{5.5}$$

which simply comes down to selecting first the child node characterized by the constraint corresponding to the bound closer to the predicted value $f_\theta(\phi(s_t))(j_t)$.

Two main drawbacks of behavioral cloning can be highlighted here. First, there is no clear reason why to chose any loss function l over another, and the usual loss functions undoubtedly do not correspond to our objective of producing minimal trees. Second, this approach comes down to learning the oracle distribution, and there is no guarantee on the performance of the learnt classifier when facing nodes which do not belong to these trajectories, *i.e.* that may be drawn from another distribution as the one produced by the oracle. In other words, there is a risk of overfitting in this setting, as the learnt policy may not be robust to variations from the training dataset. On the contrary, one may be tempted to exchange the optimal first dive $\Delta^*(p)$ for the complete tree $\mathcal{T}^{\pi^*}(p)$. However, due to Proposition 5.1.1, it may lead to underfitting as the relevant choices would likely represent only a minority of the collected samples.

5.2.2 Dataset aggregation

The second issue raised by the behavioral cloning approach can be handled using dataset aggregation (see Section 2.1.2). Rather than learning on the samples collected by the oracle, the idea is to learn on the states visited by the agent along the training process. To do so, we use a variant of DAgger [7] (see Section 2.1.2, Algorithm 1) authorized by the iterative training procedure of neural networks and using a replay buffer, as described in Algorithm 7. The point of such methodology is to transform the minimization of (5.4) into an iterative fitting procedure governed by the empirical equivalent of the theoretical loss

$$L_i(\theta) = \sum_{k \leqslant i} \mathbb{E}_{s_t, j_t \sim \Delta^{\theta_k}} \left[\omega_\delta(s_t) l\left(f_\theta(\phi(s_t)), x^*_{s_t} \right) \right] \tag{5.6}$$

where Δ^θ denotes the probability distribution for states under strategy π_θ. The agent's policy remains the one defined by Equation (5.5).

Algorithm 7 Adapted Dagger Algorithm for Node Selection

Initialization:
  Randomly initialize θ_0
Procedure:
  for $i = 1$ to N **do**:
    Draw randomly an instance p from D_{tr}
    Solve p using $\pi_{\theta_{i-1}}$
    Collect dataset $\{((s, j), \pi^*(s, j))\}$ of visited states from $\mathcal{T}^{\pi_{\theta_{i-1}}}(p)$ and oracle actions in a buffer B
    Update θ_{i-1} to θ_i following the gradient derived from Equation (5.6) using data from B
  end for
Output:
  Final parameter θ_N

Even if our objective is to leverage oracle demonstrations to learn a near-optimal strategy, one should remember that the advantage of IL compared to vanilla SL is robustness.
An extreme case of Algorithm 7 is when our model is rich enough and the number of training iterations infinite. In such a configuration, the training loss should tend to zero provided the classifier is complex enough. Thus, the agent and oracle distributions would be aligned, which would make no difference compared with the supervised setting (*i.e.* learning on the oracle distribution). To improve generalization, one may authorize some random exploration during the learning phase, in order to keep learning on small variations from the optimal strategy.

A limitation of the procedure described above is the lack of consideration for the cost model associated to the MDP mentioned earlier. As the agent only learns to mimic the oracle without any information about the cost of a mistake, it will penalize equally harmless errors and those downgrading heavily the global objective (3.2). This is easily exemplified using a worst-case argument, as an error at depth d can potentially increase the global tree size by $2^{n+1-d} - 1$. Such discrepancy is illustrated in Figure 5.2, showing that the impact of a single choice different from the oracle strategy has way more impact on the global tree size near the root node than deeper in the tree.
This issue has been highlighted in [9] in a general context, the authors proposing to take this cost into account using massively the oracle – see Section 2.1.2. As calling the oracle may be quite expensive, especially in a tree-structured environment, we present in the following a reinforcement learning

approach to reduce the calls to the expert.

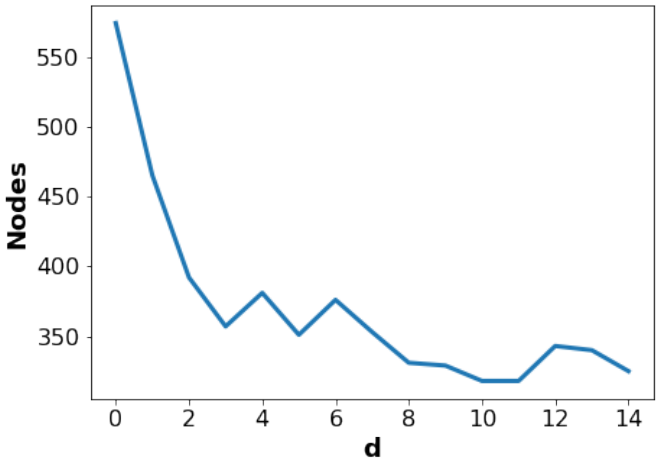


Figure 5.2: Number of nodes on a `micro_bal_12` instance when diverging from oracle choices at depth d.

5.2.3 Reinforcement learning for node selection

In this section, the objective is to overcome the shared limitations of the two previous approaches, by reducing the calls to the oracle, which may be expensive, and taking into account a cost model associated to the MDP, in order to align with our objective of tree size minimization (3.2). To do so, we propose to adapt the RL approach presented in Chapter 4 on learning the branching strategy. We rely on Proposition 5.2.1 to immediately transpose the procedure to the node selection setting: at state (s, j), the value of visiting first a child node is set to the observed size of the subtree rooted in the B&B node corresponding to s. Again, this is done by considering a unitary cost model under tree-based transitions. Proposition 5.2.1 states that minimizing this cost at each state is an oracle strategy. Note here that, compared with section 5.1.3, we do not need the uniqueness of the solution to ensure the oracle property.

Proposition 5.2.1. *Under Hypothesis 5.1.1, minimizing the subtree size is an oracle strategy and conversely.*

Proof. *This is essentially the same proof as that of Proposition 4.2.4, substituting Hypothesis 4.1.1 by Hypothesis 5.1.1.* □

Formally, the value of an action $a \in \{0; 1\}$ at state (s, j) when following a policy π is denoted

$Q^\pi(s, j, a)$, and corresponds to the subsequent size of the subtree rooted in (s, j). As for the branching strategy, a discounted version $Q^\pi_\gamma(s, j, a)$ may be encompassed.

As discussed in Section 4.2.2, the same theory can be derived in this setting and will not be repeated at this stage. Again, this cost function is approximated by a model $\hat{Q}(s, j, a; \theta)$ parametrized by a weight vector θ, and the iterative fitting procedure is governed by steps towards reducing the empirical equivalent of the theoretical loss function

$$L_i(\theta_i) = \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\left(Q^{\pi_{\theta_i^-}}(s, j, a) - \hat{Q}(s, j, a; \theta_i)\right)^2\right] \tag{5.7}$$

where θ_i^- is a fixed value from previous iterations and Δ^i is the probability distribution of state-action pairs when following some previous policies. Differentiating this loss function with respect to the weight vector gives the gradient for iteration i

$$\nabla_{\theta_i} L_i(\theta_i) \propto \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\left(Q^{\pi_{\theta_i^-}}(s, j, a) - \hat{Q}(s, j, a; \theta_i)\right)\nabla_{\theta_i}\hat{Q}(s, j, a; \theta_i)\right] \tag{5.8}$$

In this setting and according to Proposition 5.2.1, the agent's policy is now to select the action with the minimal predicted cost:

$$\pi_\theta(s, j) = \arg\min_{a\in\{0;1\}} \hat{Q}(s, j, a; \theta) \tag{5.9}$$

Note here the fundamental difference with previous updates from (5.4) and (5.6): Δ^θ is a distribution for (s, j, a) triplets, as the cost function can only be evaluated on taken actions in a RL setting. Thus, the exploration space ($\mathcal{S} \times \mathcal{J} \times \mathcal{A}$) is twice as big as the data space in the two previous approaches ($\mathcal{S} \times \mathcal{J}$).

The training procedure is described in Algorithm 8, and is really a direct adaptation of Algorithm 6.

Algorithm 8 Training Algorithm: RL for Node Selection

Initialization:
  Randomly initialize θ_0
Procedure:
  for $i = 1$ to N **do**:
    Draw randomly an instance p from the training dataset
    Solve p using $\pi_{\theta_{i-1}}$
    Collect dataset $\left\{\left((s, j), a, Q^{\theta_{i-1}}(s, j, a)\right)\right\}$ of visited states and observed Q-values from $\mathcal{T}^{\pi_{\theta_{i-1}}}(p)$ in a buffer B
    Update θ_{i-1} to θ_i following the gradient derived from Equation (5.7) using data from B
  end for
Output:
  Final parameter θ_N

Remark 1. Benefits from Proposition 5.1.1

On the one hand, a random strategy (typically at the beginning of a learning process) will be as good as any other strategy as soon as the optimal solution is found, which reduces the occurrences of sub-optimal choices, hence the exploration cost. On the other hand, but it is really the second side of the same coin, the targets (e.g. the subtree size) will appear without any noise due to future actions in the proving phase since all actions are optimal then.

5.2.4 Reinforcement learning with oracle insight

An expected benefit in integrating the cost model as defined above is quantifying the intrinsic risk of different states. However, as the agent is trained on its own behaviour, without using any insight from the oracle, Algorithm 8 yields a low sample efficiency, inherent to pure RL. In the following, we propose a simple modification to the previous approach to take into account oracle insights. The trade-off between increasing the sample efficiency and limiting the number of calls to the oracle is controlled by a hyperparameter.

As the cost model defined earlier is coherent with the objective of minimizing the tree size (see Proposition 5.2.1), any oracle strategy is a minimizer for the Q-function. Thus, for any DFS node selection strategy π, the inequality

$$Q^{\pi^*}(s, j, \pi^*(s)) \leqslant Q^{\pi}(s, j, a) \tag{5.10}$$

holds. This lower bound can be easily incorporated in the precedent learning procedure by using the same trick as in Section 4.3, *i.e.* by modifying the used loss function:

$$\begin{aligned} L_i^*(\theta_i) = \; & L_i(\theta_i) \\ & + \mathbb{P}(A_1)\, \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\, U.\left(Q^{\theta_i^-}(s,j,a) - \hat{Q}(s,j,a^*;\theta_i)\right)^2 \middle| A_1\right] \\ & + \mathbb{P}(A_2)\, \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\, U.\left(Q^{\theta_i^-}(s,j,a^*) - \hat{Q}(s,j,1-a;\theta_i)\right)^2 \middle| A_2\right] \end{aligned} \tag{5.11}$$

with $U \sim \mathcal{B}(\alpha)$ an independent Bernoulli random variable, $A_1 = (a \neq a^*) \cap \left(Q^{\theta_i^-}(s,j,a) < \hat{Q}(s,j,a^*;\theta_i^-)\right)$ and $A_2 = (a = a^*) \cap \left(\hat{Q}(s,j,1-a;\theta_i^-) < Q^{\theta_i^-}(s,j,a^*)\right)$.

Differentiating this loss function with respect to the weight vector now gives the gradient for iteration i $\nabla^*_{\theta_i}$

$$\begin{aligned}\nabla_{\theta_i} L_i^*(\theta_i) \propto\ & K \nabla_{\theta_i} L_i(\theta_i) \\ & + \alpha \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\left(Q^{\theta_i^-}(s,j,a) - \hat{Q}(s,j,a^*;\theta_i)\right)\nabla_{\theta_i}\hat{Q}(s,j,a^*;\theta_i)\mathbb{1}_{A_1}\right] \\ & + \alpha \mathbb{E}_{s,j,a\sim\Delta^i}\left[\omega_\delta(s)\left(Q^{\theta_i^-}(s,j,a^*) - \hat{Q}(s,j,1-a;\theta_i)\right)\nabla_{\theta_i}\hat{Q}(s,j,1-a;\theta_i)\mathbb{1}_{A_2}\right]\end{aligned} \tag{5.12}$$

with $\alpha \in [0;1]$ the proportion of samples where calls to the oracle should be made to augment the gradient and K some normalizing constant.

Such modification encourages the model to satisfy the lower bound provided by any oracle (Equation (5.10)) through two additional feedbacks: (i) oracle choices should have a lower cost than taken actions and (ii) the non-taken actions are at least as bad as oracle choices if the latter are taken. These feedbacks allow the agent to learn even on non-taken actions, which should increase the sample efficiency.

Note here that expert's demonstrations are used only if the agent is making mistakes in ranking the actions. Besides, Equation (5.11) does not require to compute the cost of following the oracle strategy from state s as in [9]. Instead, we only need to determine if the taken action is an oracle choice or not, which does not always require a call to the B&B procedure. Besides, recreating exactly the state (s, j) may not be trivial with certain MILP solvers. It also allows to solve the corresponding sub-problem with any desired configuration of the solver, which may permit some computational gain. On the contrary, computing the cost of the oracle strategy $Q^{\pi^*}(s, j, a)$ would require to solve the sub-problem with the exact same configuration to ensure coherence.

5.3 Experiments and discussions

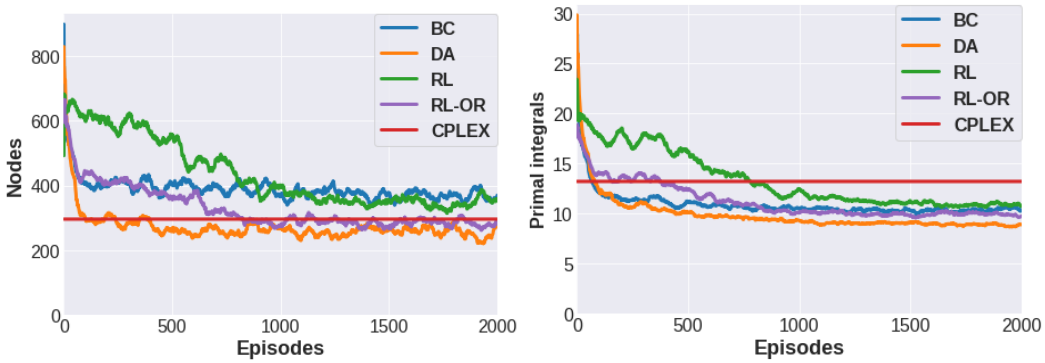
In order to evaluate the sampling efficiency of the different methodologies, the behavioral cloning agent is trained sequentially, using the same number of training samples as the other competitors. This manoeuvre does not change the fact that data are collected once and for all by observing the oracle, but allows to better evaluate the sample efficiency of dataset aggregation and reinforcement learning agents.

Figure 5.3 displays training processes for the presented strategies under the subtree cost model (the

results for the unitary cost model are similar see Figure 5.4), and Table 5.1 presents results on test instances. If we compare these results, not only on train but also on test instances, with those obtained when learning the branching strategy, we see that the performances are better here. This should not come as a surprise. First, the learning task is easier as the action space is reduced. Second, many actions do not impact the performance, according to Proposition 5.1.1. Last, we already mentioned that the branching strategy is often more important than the node selection strategy. Here, we let `CPLEX` make the branching decisions, which allows the errors in the node selection to be relatively benign. The fact that node selection has a lower impact on the tree size than branching is illustrated by the lower gains obtained during the learning processes than those observed in Chapter 4.

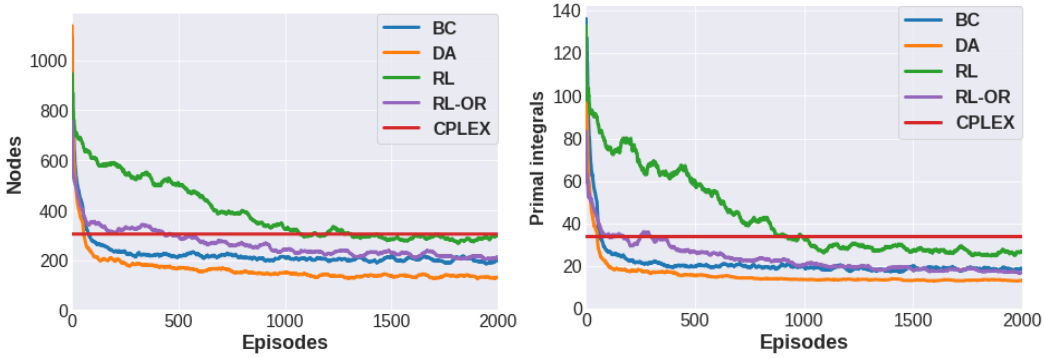
Notice here that reinforcement learning methods compare well with both BC and DA in terms of sample efficiency. This is due to the fact that our action space contains only two elements, which reduces drastically the need for exploration. Likewise, using additional gradients from the oracle strategy (`RL-OR`) allows to improve the sample efficiency in comparison with the RL approach without demonstrations (`RL`).

As expected due to the nature of the strategies, we see that they compare well with `CPLEX` regarding primal integral scores.



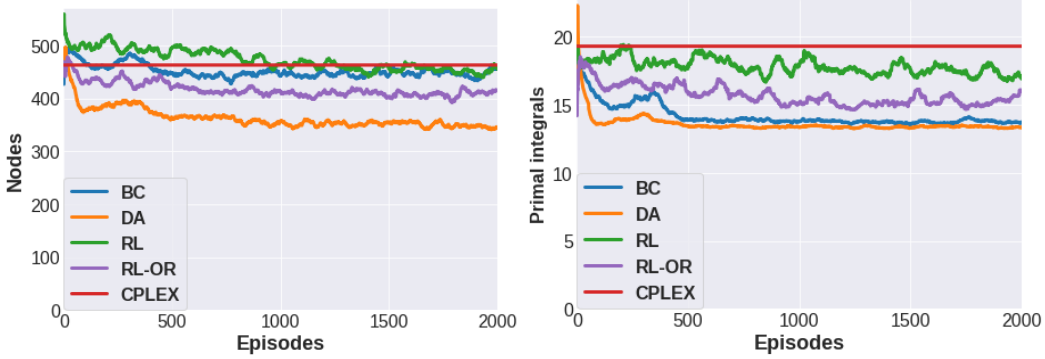
(a) Tree sizes – `micro_asym_T12`

(b) Primal integral scores – `micro_asym_T12`



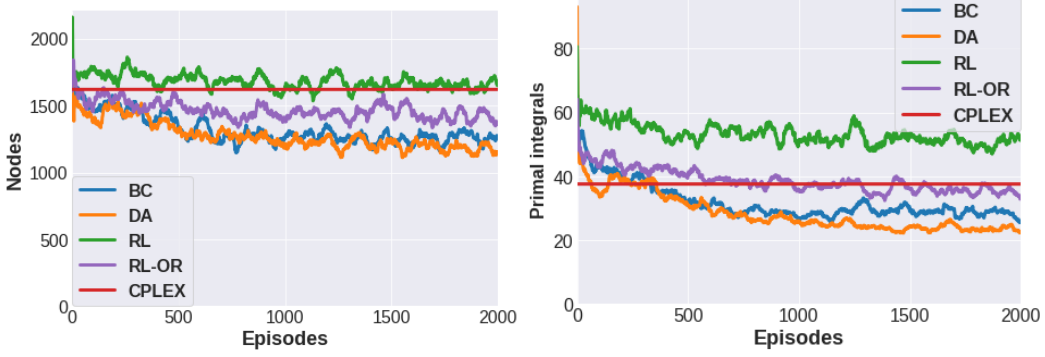
(c) Tree sizes – `micro_bal_T12`

(d) Primal integral scores – `micro_bal_T12`



(e) Tree sizes – `hydro_fix_2`

(f) Primal integral scores – `hydro_fix_2`



(g) Tree sizes – `hydro_var_2`

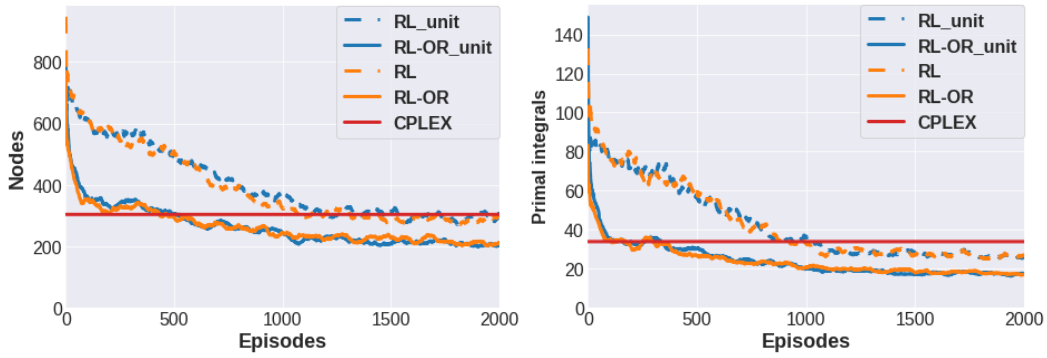
(h) Primal integral scores – `hydro_var_2`

Figure 5.3: Training processes: comparison of node selection strategies.
`BC` refers to the behavioral cloning approach, `DA` to dataset aggregation, `RL` to "pure" reinforcement learning and `RL-OR` to reinforcement learning with expert demonstrations.

	BC		DA		RL		RL-OR	
	Train	Test	Train	Test	Train	Test	Train	Test
micro_asym_T8	+16%	+28%	-10%	+8%	-4%	+6%	-2%	**-1%**
micro_bal_T8	-27%	-25%	-49%	**-36%**	-40%	-28%	-48%	-24%
micro_asym_T12	15%	27%	-16%	**-12%**	-5%	0%	-9%	-2%
micro_bal_T12	-41%	-30%	-60%	**-37%**	-24%	-14%	-43%	-26%
hydro_fix_2	-10%	-4%	-29%	**-18%**	-15%	-15%	-21%	-16%
hydro_var_2	-21%	8%	-25%	**-8%**	0%	6%	-9%	21%

Table 5.1: Tree sizes comparison against CPLEX on train and test instances for the best agent on train over 25 seeds.

We see in Table 5.1 that taking into account the cost associated to actions (as we do in RL) do not provide better generalization performances. Note that, contrary to the learning of the branching strategy, the unitary and subtree cost models have similar performances, as illustrated in Figure 5.4.



(a) Tree sizes – micro_asym_T12 (b) Primal integral scores – micro_asym_T12

Figure 5.4: Training processes: comparison between the unitary (*_unit) and subtree cost models.

Chapter 6

RL for Branching and Node Selection Strategies

Content

In this chapter, we present different ways of combining the approaches of Chapter 4 and Chapter 5 to learn the full strategy, that is to say a policy which selects both variables and nodes. Starting from the full RL framework presented in Section 6.1, we propose in Section 6.2 variations based on the developments of Chapter 5. Section 6.3 presents some experimental results.

6.1 Unifying the branching and node selection strategies

The RL methodologies presented in Chapter 4 and Chapter 5 can be unified in a straightforward way to learn both strategies at the same time. Considering the MDP defined in Chapter 4 (Section 4.1.2), we only modify the action space. Instead of defining an action as the selection of a branching variable at the current node, we set it to the choice of both a branching variable and a child node priority. When enforcing DFS, such choice fully characterizes the two strategies.

Concretely, this is performed by considering the action space $\mathcal{J} \times \{0, 1\}$ and an according Q-function $Q^\pi : \mathcal{S} \times \mathcal{J} \times \{0, 1\} \to \mathbb{R}$. This setting doubles the size of the search space compared with Chapter 4, which may lead to a more tedious exploration. Let us take the example of the unit cost model under tree-based transitions to make the analogy easier with the case of learning only the branching strategy. Consider a state s and an action selected by policy π which is "branching on variable j and visit first the child node associated to the additional constraint $x_j = k$" with $j \in \mathcal{J}$ and $k \in \{0, 1\}$. By definition, the Q-value of this state-action pair is the subtree size rooted in the node $\zeta(s)$ when following the according strategy. Note that this construction allows us to retain the oracle property of an optimal policy under DFS (see Proposition 5.2.1). For that matter, it is important to understand that the Q-value is not the subtree size rooted in the node $\zeta\left(D_k^\pi(s)\right)$ but that rooted in $\zeta(s)$, as the property previously mentioned would not hold in such case.

The same remarks as those made in Chapter 4 regarding the impact of transitions on the credit assignment problem can be done, hence we do not repeat them here. Likewise, the justifications for incorporating a bias by considering the subtree cost model are identical.

6.2 Variations

In light of the developments made in Chapter 5 when considering the learning of the node selection strategy, different variations can be built from the direct methodology mentioned above (later referred to as `RL`). Since the node selection strategy has no impact on the tree size as soon as the optimal solution is found (see Proposition 5.1.1), it seems inefficient to learn node selection actions after this point. In addition, the objective of node selection under DFS has been clearly set: given a branching decision, one must prioritize the child node leading to the best solution. The variations proposed below seek to exploit such facts.

`RL-2` – The first variation consists in training two agents, which act at different locations in the B&B tree. The first agent has to select both the branching variable and node priority for any node prior the discovery of an optimal solution. As for the second, it only focuses on branching decisions past this point, node priority being given by some exogenous heuristic. Of course, this strategy cannot be maintained at test time, as one does not know anymore if a feasible solution is optimal or not. Therefore, at test time, the second agent is to take charge as soon as a feasible solution is found.

`RL-DA` – The oracle node selection strategy exhibited in Chapter 5 is valid for any given branching strategy. As a consequence, we naturally propose to learn separately branching and node selection, the former being learnt by an agent using an MDP as defined in Chapter 4 and the latter by imitation learning, and more precisely dataset aggregation, as proposed in Chapter 5.

`RL-OR` – For the same reason, one can consider to leverage oracle demonstrations for guiding a full RL approach. Considering a unique agent with the action space $\mathcal{J} \times \{0, 1\}$, one can obtain cost signals for non-taken actions as proposed in Chapter 5 by accounting for the fact that oracle actions should be better than selected actions (see Equation (5.11)). As an oracle decision consists in selecting the child node which contains the optimal solution under the current node, we already mentioned that using this strategy is expensive. An alternative may consist in using the optimal solution of the instance rather than that of the subproblem associated to the current node to derive the additional gradient. This strategy is referred as `RL-OR-partial` in the experiments.

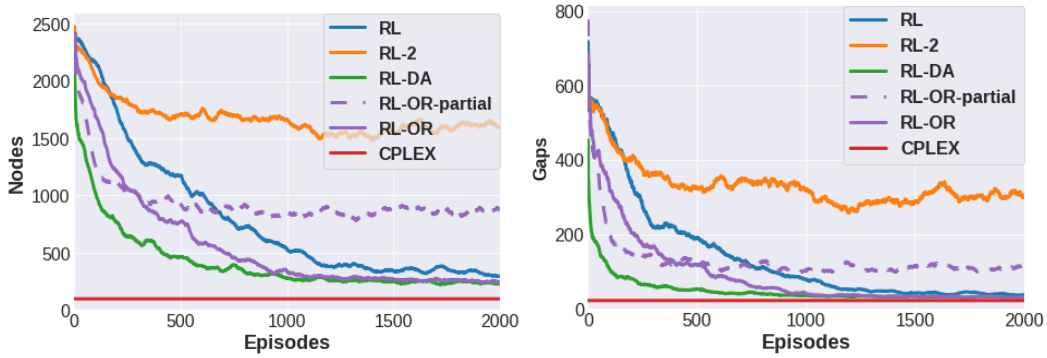
6.3 Experiments and discussions

Figure 6.1 displays an example of training processes for the strategies mentioned above, using the subtree cost model.

First, we observe that `RL-OR-partial` discovers less efficient strategies than `RL-OR`. This result was expected and highlights the fact that the agent has to adapt to the case where it strays from the first optimal dive – recall that, according to Proposition 5.1.3, the oracle strategy first performs a dive toward the optimal solution.

Next, we note that `RL-2` is the worst approach. This failure may be caused by the fact that each agent has fewer samples at its disposal for learning. Another potential explanation is that the sample distribution of the second agent depends on the behavior of the first, which causes coordination issues.

The two agents which discover the best strategies are `RL-DA` and `RL-OR`, before `RL`. This result is in line with those observed in Chapter 5, where a similar ranking was observed.

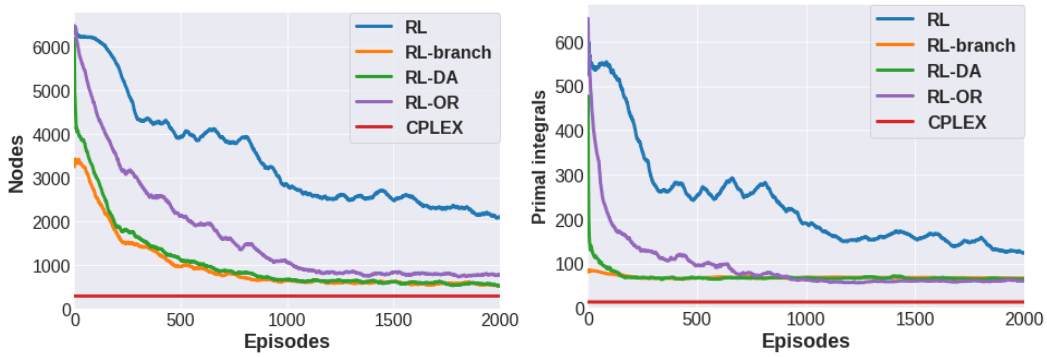


(a) Tree sizes – `micro_bal_T8` (b) Primal integral scores – `micro_bal_T8`

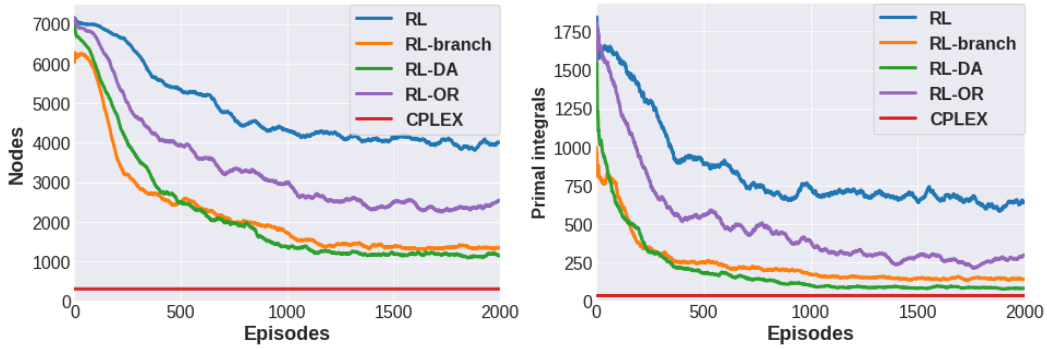
Figure 6.1: Training processes: comparison of full strategies on `micro_bal_T8`.

We see when looking at the primal integral scores in Figure 6.1 that performances with respect to that score are weaker than those obtained when learning the node selection only – in the experiments of Chapter 5, all the agents obtained comparable or better scores than `CPLEX` (see Figure 5.3). An explanation of this phenomenon may lie in the fact that the distribution of the learning samples is evolving with the branching strategy, which makes the learning task more difficult. Of course, the quality of the branching strategy may also play some role. In that matter, Figure 6.2 shows experiments on more difficult problems, and also displays as a reference the training process observed when learning

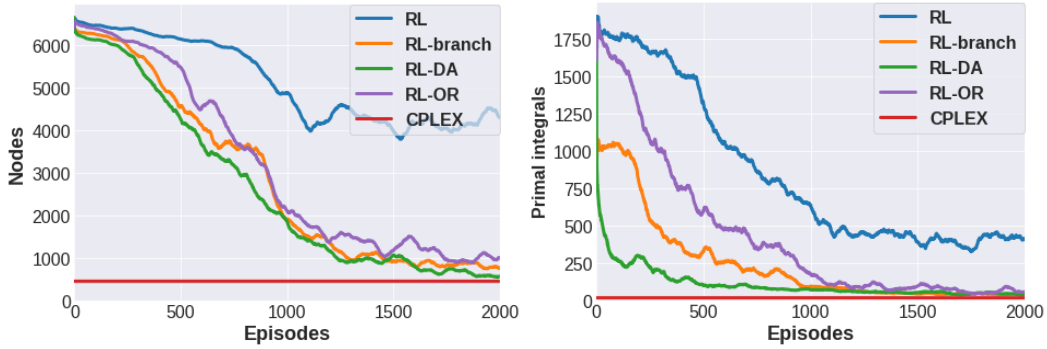
only the branching strategy. We see that we do not manage to obtain better performances when learning the full strategy compared with learning only the branching strategy. Actually, we even observe poorer performances on `hydro_var_2`. Except for `hydro_fix_2`, the primal integral scores do not decrease faster compared with branching only, which uses a stationary node selection strategy.
These difficulties are also explained by the enlargement of the search space, even in `RL-DA` where the node selection is learnt by dataset aggregation. Indeed, even if the state space is actually identical between `RL-DA` and `RL-branch`, the latter samples states in a more restrained area of the search space due to the stationarity of the node selection strategy. Thus, even `RL-DA` faces in practice a greater variance in the training samples, which can be assimilated as an increase of the search space and a greater need for generalization.
When looking into the results on test instances (Table 6.1), we see that the performances of the best agent are generally better when focusing on learning the branching strategy. Note however that it is not the case on `hydro_fix_2`, where the invariance of the feasible set allows an agent learning both strategies to obtain better performances.



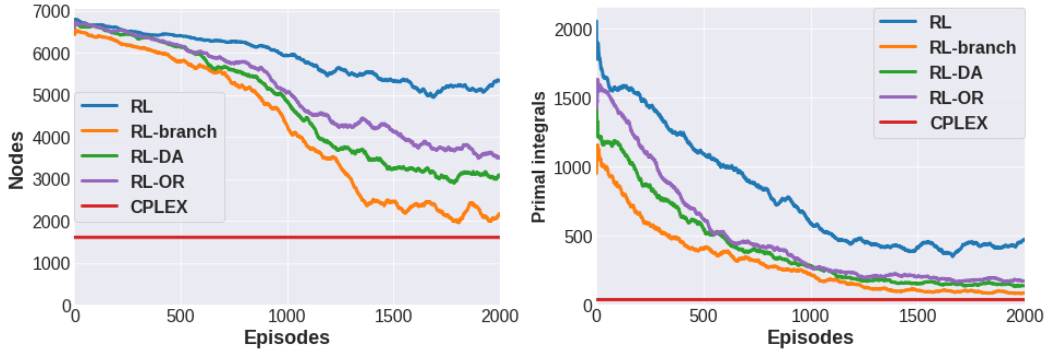
(a) Tree sizes – `micro_asym_T12` (b) Primal integral scores – `micro_asym_T12`



(c) Tree sizes – `micro_bal_T12` (d) Primal integral scores – `micro_bal_T12`



(e) Tree sizes – `hydro_fix_2` (f) Primal integral scores – `hydro_fix_2`



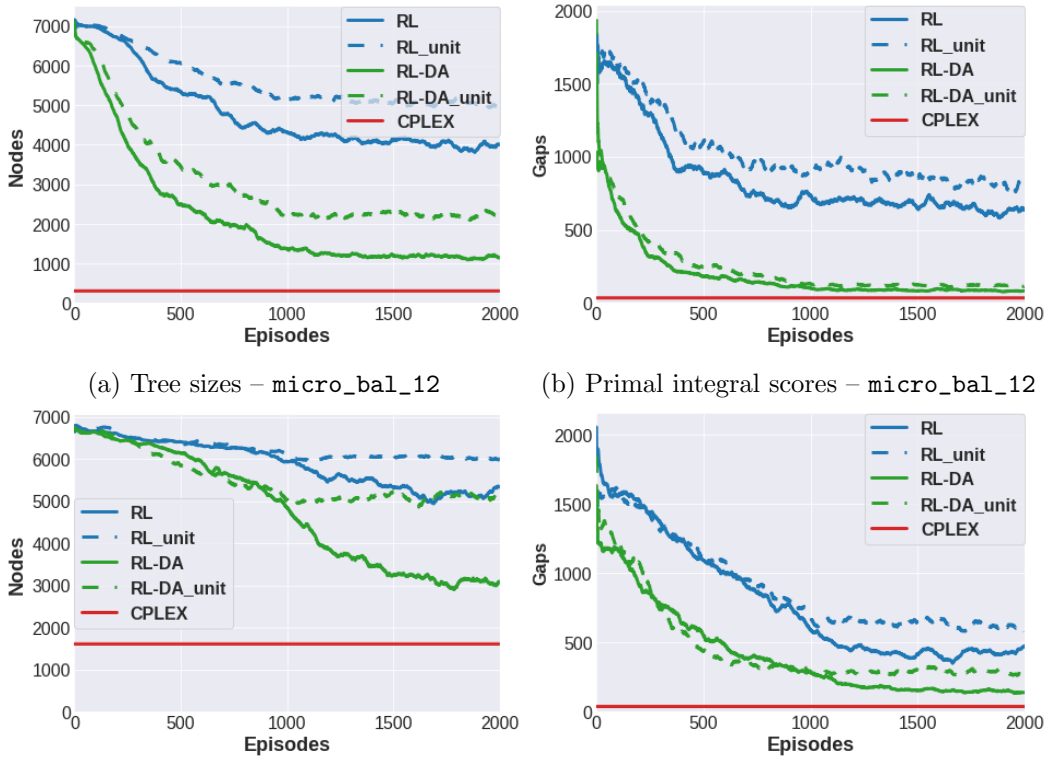
(g) Tree sizes – `hydro_var_2` (h) Primal integral scores – `hydro_var_2`

Figure 6.2: Training processes: comparison of complete strategies, using a branch-only agent as benchmark (`RL-branch`).

	RL		RL-branch		RL-DA		RL-OR	
micro_asym_12	+156%	+267%	**-5%**	**+ 2%**	+98%	+127%	+104%	+184%
micro_bal_12	+529%	+511%	**+111%**	**+141%**	+296%	+340%	+425%	+478%
hydro_fix_2	+63%	+79%	-1%	+ 0%	+0%	-2%	**-25%**	**-19%**
hydro_var_2	+270%	+300%	**-35%**	**-24%**	+275%	+219%	38%	+20%

Table 6.1: Performances on test instances against CPLEX for the best agents on train instances over 25 independent training processes.

Figure 6.3 shows a comparison with the training processes obtained when using the unitary cost model. As for the branching case, we see that performances are lower than those obtained under the subtree cost model.



(a) Tree sizes – micro_bal_12

(b) Primal integral scores – micro_bal_12

(c) Tree sizes – hydro_var_2

(d) Primal integral scores – hydro_var_2

Figure 6.3: Training processes: comparison between the unitary and subtree cost models when learning complete strategies.

Part III

Exploiting the Problems' Structure

This part briefly presents some attempts to leverage the structure of the considered problems to reduce the computational effort. To this end, we encompass different approaches such as heuristic branching, decomposition techniques and objective disruption. Although these axes are independent, they all share the same interest in using the underlying structure of the considered problems.

Chapter 7

A Graph Branching Heuristic

Content

Although the most used branching heuristics are LP-based (see Section 2.3.1), various branching strategies have been developed using a score based on the problem data. For example, orbital branching [98] leverages the problem information encoded in symmetry groups to build a new branching dichotomy. With a different approach, [91] builds a set of clauses from fathomed nodes to guide the branching decision toward early infeasible or sub-optimal nodes.
In this chapter, we propose a new kind of heuristic for performing variable selection in a B&B algorithm, based on a graph representation of the problem data. This graph representation appears to partly encode the underlying structure of the problem and thus may be used for branching. Section 7.1 proposes a graph representation for any node in a B&B tree and interpret it as an influence graph. Next, Section 7.2 presents a new branching heuristic based on this graph representation of the problem data. An alternative PCA-flavoured interpretation of such heuristic is also given. Last, we conduct some experiments in Section 7.3 and discuss briefly the impact and limitations inherent in this approach.

7.1 Leveraging the problem's structure through a graph representation

Taking some distances with LP-based heuristics, we propose a branching heuristic based on the problem's structure, using a graph representation of the interactions between variables. Using this representation, we later define a heuristic which can be interpreted in two distinct ways.

7.1.1 Graph representation

Graphs have already been used in the literature for the design of branching heuristics. For instance, a bipartite graph is used in [98] to compute symmetry groups for orbital branching. Here we propose different variants of a graph representation, that we use actively for performing variable selection.
Through this representation, our conviction is that we can roughly model the influence that branching on a variable has on other variables, in the sense of its tightening impact on the LP relaxation in the corresponding dimensions. Intuitively, this influence is the result of a complex combination of various effects, induced not only by the constraints but also by the objective function. The raw matrix representation of the constraint matrix is clearly not adequate to capture these effects (for instance, it is not invariant to indices' permutation), and we believe that a graph representation is more suited to this end. Let us first define what we mean by influence, and then propose a class of graphs used to

represent these influences for any MILP instance.

Definition 7.1.1. *Local influence*
We say that variable i has a non-zero local influence on variable j with respect to constraint k if $\mathbb{1}_{A_{ki}}\mathbb{1}_{A_{kj}} = 1$, with A_{ki} the coefficient of variable i in constraint k. At this point, the quantification of this influence ω_{ij}^k is not specified and may be any function of the problem data A, b, c – see Section 1, Problem (1.2), for notations.

Definition 7.1.2. *Direct influence*
We define the direct influence ω_{ij} of variable i on variable j as the sum over the constraints of the local influences:

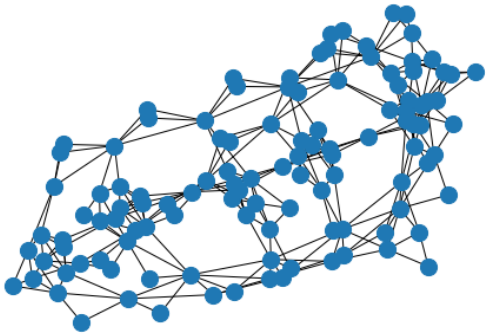
$$\omega_{ij} = \mathbb{1}_{i \neq j} \sum_{k=1}^{m} \omega_{ij}^k$$

Building on these definitions, we define the class of *influence graphs*, whose weight matrix is built according to the notion of influence as defined above.

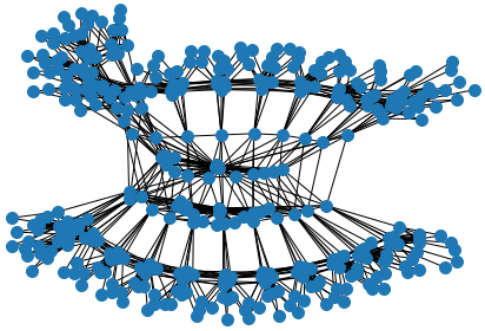
Definition 7.1.3. *Influence graph for MILP*
In the MILP setting, we say that a directed graph $G = (V, E, W)$ is an influence graph if $V = \{1, ..., n\}$, $E = V \times V$ and the edges' weights $W \in \mathbb{R}^{n \times n}$ satisfy the definition of direct influence.

By construction, an influence graph is a *primal graph* for the considered instance, *i.e.* a graph where nodes correspond to variables and an edge may exist between two variables only if they appear in a same constraint. Note that vertices are associated to variables, both binary and continuous.
We voluntarily keep this class of influence graphs large, so as to be in a position to consider different definitions of the direct influence.
However, regardless of the quantification of the weights, the definition of local influence and the additive nature of direct influence enforce a particular structure for influence graphs. Especially, theses graphs not only naturally exhibit the block structure often observed in the constraint matrix of real-world problems, but also the interconnections between these blocks. Figure 7.1 shows examples of influence graphs for two instances of the `microgrid` and `hydro` problems. We see in these examples that the influence graph representation allows to exhibit in our problems a "ladder structure", produced by temporal and spatial interconnections between variables.

7.1.2 Examples of influence graphs



(a) `binary_graph` for a `micro_asym_6` instance



(b) `binary_graph` for a `hydro_fix_2` instance

Figure 7.1: Examples of influence graphs.

We present here some examples of influence graphs, based on different definitions of local influence. Perhaps the most basic example of such graphs is obtained by defining the local influence of a variable i on variable j with respect to constraint k as

$$\omega_{ij}^k = \mathbb{1}_{A_k i}\mathbb{1}_{A_k j}$$

Using this definition, we obtain the `count_graph` where the weight of an edge between two nodes is the number of constraints linking the two corresponding variables. In a more agnostic manner, the `binary_graph` is obtained by setting

$$\omega_{ij}^k = \frac{\mathbb{1}_{A_k i}\mathbb{1}_{A_k j}}{\max\left\{1, \sum_{l=1}^m \mathbb{1}_{A_l i}\mathbb{1}_{A_l j}\right\}}$$

In this setting, the weight of an edge is the indicator of an existing constraint between the two considered variables.

Remark 1. Constraint-based influence graphs and B&B tree

An influence graph is a way of representing a MILP instance. As a consequence, it can be used to represent any node in a B&B tree. Along such tree, the evolution of the influence graphs associated to nodes is governed by the branching strategy. When considering pure constraint-based influence graphs as `binary_graph` and `count_graph`, they become sparser as the nodes are deeper. When a variable is fixed, the adjacent edges are removed. If we denote W_t the weight matrix of the influence

graph associated to some node ζ_t and $W_{t'}$ that of a descendant node, then a binary matrix $B_{t,t'}$ of dimension (m, n) exists such that $W_{t'} = B_{t,t'} \circ W_t$ with $\circ$ the Hadamard product. Especially, it implies that the adjacency matrix becomes sparser as we move down along a branch of the B&B tree.

The two graphs presented above only use the information provided by the constraints, without any consideration regarding the objective function. To encode this function in an influence graph, one can use LP-related data to weight the constraints. Rather than weighting equally the constraints, the idea is to put more mass on the binding constraints, that is to say on constraints which have a non-zero optimal value in the dual LP associated to the current node. Writing $y^* \in \mathbb{R}^m$ the optimal solution of the dual, the `binary_dual_graph` puts a zero mass on non-biding constraints and the others are weighted equally:

$$\omega_{ij}^k = \mathbb{1}_{A_k i} \mathbb{1}_{A_k j} \mathbb{1}_{y_k^* \neq 0}$$

A natural generalization is the `dual_graph`, obtained by setting

$$\omega_{ij}^k = \mathbb{1}_{A_k i} \mathbb{1}_{A_k j} \, |y_k^*|$$

Remark 2. Influence graphs and invariance to reformulation

Contrary to the matrix representation, influences graph are invariant to index permutations. However, we see in the examples using LP information that it is not invariant to a rescaling of the data A, b and c. To ensure a coherence in the constraints' weighting, some standardization may be performed whenever the constraint coefficients A_{ij} appear directly in the definition of the local influence. For instance, we may apply:

- $c \leftarrow \frac{c}{\sigma(c)}$ if $\sigma(c) \neq 0$ where $\sigma(c)$ is the standard deviation of the coefficients in c;
- $A_{k.} \leftarrow \frac{A_{k.}}{b_k}$ for any row $A_{k.}$ associated to a non-zero coefficient b_k;
- $A_{k.} \leftarrow \frac{A_{k.}}{\sigma(A_{k.})}$ if $\sigma(A_{k.}) \neq 0$ where $\sigma(A_{k.})$ is the standard deviation of the coefficients of the row $A_{k.}$ associated to a zero coefficient b_k.

7.1.3 Using the graph representation for branching

Naturally, when looking at Figure 7.1, it comes that not only a variable i has a direct influence on adjacent variables, but also indirect influence on any other variable j such that a path exists from i to j in the graph. Therefore, a natural heuristic for variable selection would be to branch on the integer variable with the highest influence on other integer variables. Depending on the chosen definition of the local influence, the stress may be put on the ability to create infeasible descendant nodes or early primal solutions.

This question of Influence Maximization on graphs (IM) has been widely studied in the past decades and is still an active field of research, especially with the emergence of social networks. In addition to the adequate definition of the direct influence for the problem of interest, one of the main challenges of IM is the definition of a diffusion process of the influence (see [99] for a recent survey). Note here that it has been proven for different diffusion processes that the IM problem is NP-hard.

Unfortunately, a requirement for a good branching strategy is its fast computation. Therefore, it seems useless to look for an exact solution of an ill-defined (*i.e.* heuristic) IM problem in this setting. This is the reason why we rather look for potentially non-accurate but fast approximation for IM. We propose to take inspiration from [100], where the authors use the MinCut algorithm for detecting influence communities in a social graph.
In the setting of variable selection, we want two variables to belong to the same community if branching on one of them has a strong impact of the second. If the influence is seen as a proxy for the tightening of the LP relaxation following a branching decision, branching consecutively on two variables of the same community would likely be inefficient. To avoid that, one can define K clusters with at most one representative by cluster and take them as candidates for branching.

7.2 Branching strategy

7.2.1 Selecting high-influential variables

As discussed above, using exact IM to perform variable selection in a B&B algorithm may be counterproductive. Therefore, we propose to approximate IM and select various high-influential variables in different locations of the influence graph.
To reach that objective, we use a two-stage approach. First, we exhibit K communities of comparable influence. This is done by approximating RatioCut through Spectral Clustering on a given influence graph, which allows to obtain clusters of comparable sizes and relatively independent from each other (see [101] for a detailed tutorial on Spectral Clustering). Next, we select in each group the integer variable with the highest total influence (see the definition below). These variables are then taken as candidates for branching, a natural ordering being given by their total influence.

Definition 7.2.1. *Total influence*
We define the total influence ω_i of variable i as the sum over its direct influences:

$$\omega_i = \sum_{j \neq i} \omega_{ij}$$

Formally, let W be the weight matrix of an influence graph associated with the current node of the B&B tree and $L = D - W$ the associated Laplacian with D the diagonal of W. Let $\mathbf{V} \in \mathbb{R}^{n \times K}$ be the matrix obtained by stacking the K eigenvectors associated to the K lowest eigenvalues of L ordered increasingly. The influence communities $(C_k)_{k=1}^K$ then form a partition of the index set $\{1, ..., n\}$ and are obtained by performing a K-means clustering on $\mathbf{V}$, *i.e.* is the solution of

$$\underset{\{C_1,...,C_K\} \in \mathcal{C}}{\arg\min} \sum_{k=1}^{K} \sum_{v_i \in C_k} ||v_i - \mu_k||_2^2 \tag{7.1}$$

with $\mathcal{C}$ the set of K-partitions of the index set, $(v_i)_{i=1,...,n}$ the rows of $\mathbf{V}$ and $\mu_k = \frac{1}{|C_k|} \sum_{v_i \in C_k}$.
Using these clusters, the set $\mathcal{S}_K$ of branching candidates is then defined as

$$\mathcal{S}_K = \left\{ \underset{i \in C_k \cap \mathcal{J}}{\arg\max}\, \omega_i \ ,\ k = 1, \ldots, K \right\}.$$

These variables are then ordered following the total influence of each variable: for two variables $i, j \in \mathcal{S}_K$, we say that i is prioritized over j (written $i \succeq j$) if it has a greater total influence

$$i \succeq j \iff \omega_i \geqslant \omega_j$$

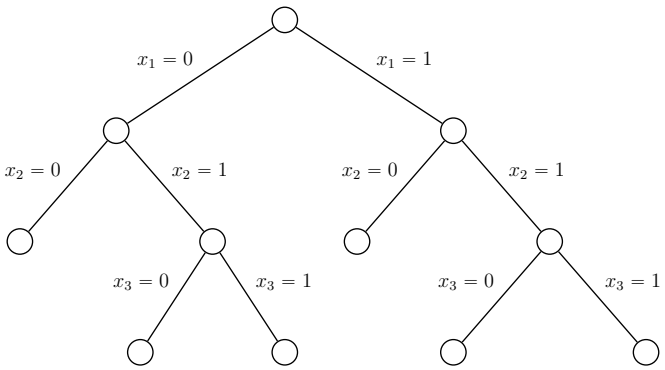


Figure 7.2: Branching decisions from the root node with $\mathcal{S}_K = \{x_1, x_2, x_3\}$ with $1 \geq 2 \geq 3$.

If we consider now $\mathcal{S}_K$ as an ordered set following the previous ordering rule, branching is performed on the first variable of such set at the current node, on the second at the immediate child nodes and so on in the subtree rooted in the current node. This is illustrated by Figure 7.2 Of course, one can set $K = 1$ and repeat the procedure at each node while bypassing the clustering part.

7.2.2 Alternative interpretation

Let us consider $X \in \mathbb{R}^{n \times m}$ a row-wise L^2-standardized matrix such that the dot product $W = XX^\top$ is a weight matrix associated to an influence graph. For instance, X may be the standardized version of the transpose of the constraint matrix A. A MILP instance is then represented by X as a scatter of n points living in $\mathbb{R}^m$. The L^2 standardization of each point can be seen as a way of uniforming the importance of each variable, thus preventing undesirable effects due to a potential bad conditioning of the problem formulation (see Remark 2).

Proposition 7.2.1. *Under the above assumptions, performing a K-means clustering on the K-dimensional projection of a non-centered PCA on X is equivalent to the K-spectral clustering on W.*

Proof.

Let us call $\mathbf{P} \in \mathbb{R}^{K \times m}$ *the projection matrix obtained from a non-centered K-PCA on X, i.e.* $\mathbf{P}$ *obtained by solving*

$$\begin{cases} \max\limits_{P \in \mathbb{R}^{K \times m}} Tr\left(PX^\top XP^\top\right) \\ s.t. \quad PP^\top = I_K \end{cases}. \tag{7.2}$$

It is well known that $\mathbf{P}$ *is obtained by stacking the K eigenvectors associated with the K largest eigenvalues of* $X^\top X$.

Let us set $\boldsymbol{U} = X\mathbf{P}^\top \in \mathbb{R}^{n\times K}$ *the matrix composed of the projected points* $u_1, ..., u_n$ *as rows. To assert the equivalence of the two methods, we shall demonstrate that* $\boldsymbol{U} = \boldsymbol{V}$ *with* $\boldsymbol{V}$ *previously mentioned. As stated above,* $\boldsymbol{V}$ *is obtained by stacking the* K *lowest eigenvectors of* $L = D - W = D - XX^\top$ *where* D *is the diagonal of* $XX^\top$*, hence is the solution of*

$$\begin{cases} \min\limits_{V\in\mathbb{R}^{n\times K}} Tr\left(V^\top\left(D - XX^\top\right)V\right) \\ s.t. \quad V^\top V = I_K \end{cases}. \tag{7.3}$$

As X *is row-wise standardized, we have* $D = I_n$*. Then* $\boldsymbol{V}$ *is the solution of*

$$\begin{cases} \max\limits_{V\in\mathbb{R}^{n\times K}} Tr\left(V^\top XX^\top V\right) \\ s.t. \quad V^\top V = I_K \end{cases}. \tag{7.4}$$

As a consequence, $\boldsymbol{V}$ *is obtained by stacking the* K *eigenvectors associated to the* K *largest eigenvalues of* $XX^\top$*. Since we want to show that* $\boldsymbol{V} = X\mathbf{P}^\top$*, it is sufficient to show that if* u *is an eigenvector of* $X^\top X$ *associated to the eigenvalue* λ*, then* $v = Xu$ *is an eigenvector of* $XX^\top$ *associated to the same eigenvalue* λ*. This result is trivial, since*

$$X^\top Xu = \lambda u \implies \left(XX^\top\right)Xu = \lambda Xu.$$

□

Proposition 7.2.1 has the merit to present a different representation of a MILP, not as a weighted graph but as a scatter of points assumed to live in an Euclidean space. However, a disadvantage of this representation is that incorporating LP information in the embedding is less straightforward.

7.3 Experiments

We ran experiments on the influence graphs presented above. To avoid the open question of selecting the pertinent heuristic along the Branch and Bound tree, we only use our heuristic at the root node. Hence the procedure is as previously illustrated in Figure 7.2:

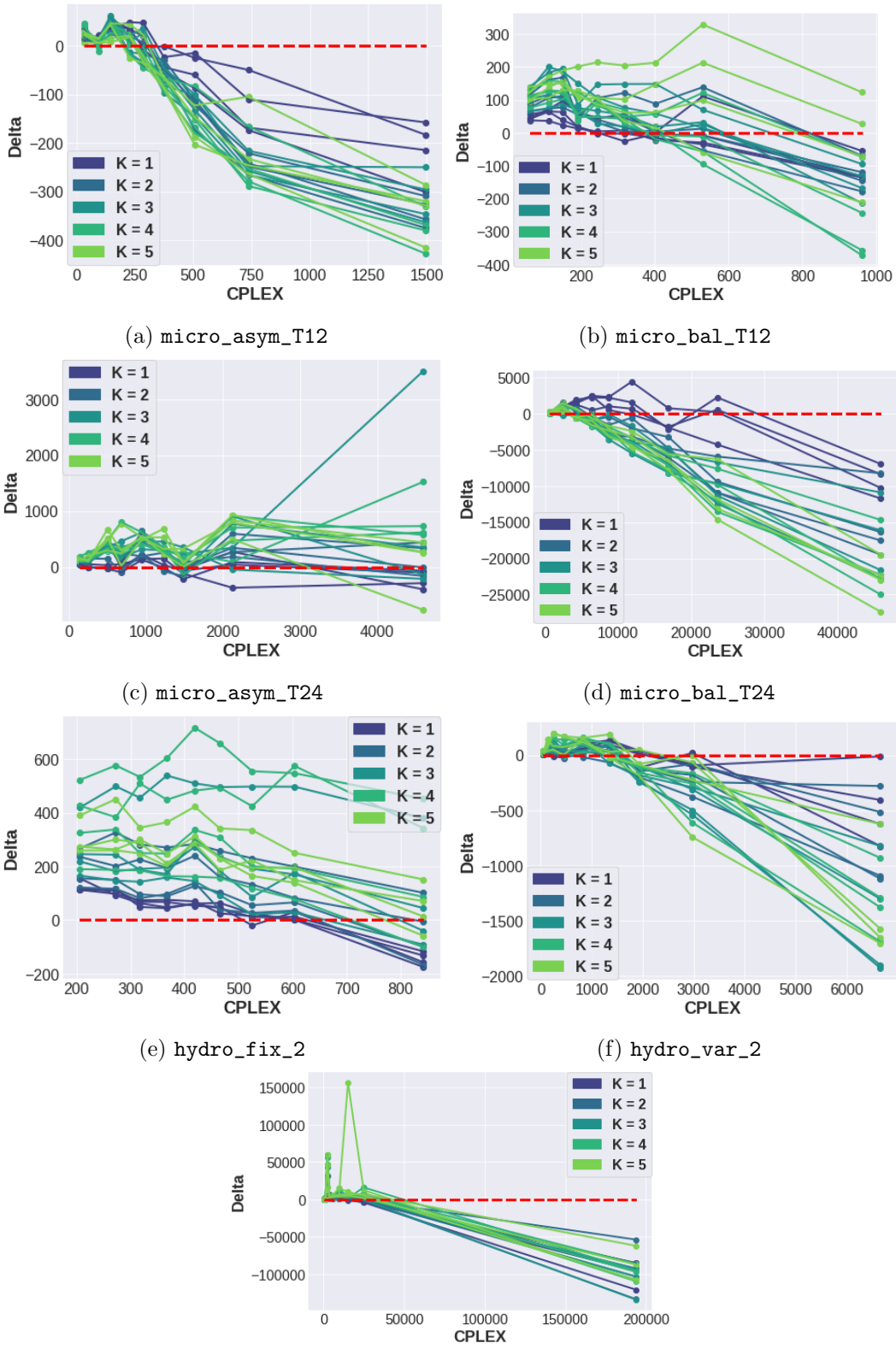
1. At the root node, compute the influence graph.
2. For a given K, compute the ordered set $\mathcal{S}_K$.
3. On each branch, branch successively on each variables in $\mathcal{S}_K$.

The averaged performances on 500 instances of `microgrid` and `hydro` problems are available in Table 7.1. More detailed results are presented in Figure 7.3. The first point that we highlight is that we did not observe any hierarchy in the performances obtained for each graph. However, the problem considered has a huge impact on the heuristic's performance. This illustrates the fact that our graphs do not fully model the influence that branching may have on each variable. For instance, if we consider two binary variables, the two constraints $x_i + x_j = 1$ and $x_i + x_j \leqslant 2$ are equally considered in `binary_graph` and `base_graph`, which can heavily pollute the modeling.
Except for `micro_asym_T24`, the proposed heuristic is all the more efficient as the instances are difficult to solve for `CPLEX`. This is a rather encouraging result. Naturally, there is not much one can do to downsize already small trees. However, the fact that the heuristic performs relatively well on difficult instances seems to advocate that it allows to take advantage of the problem's structure. This is supported by the fact that the best value of K for each graph is generally higher when the performances on the problem are good.

The results presented here seem to advocate that MILP instances can be represented by graphs. Actually, the interest in using graphs and especially graph neural networks to encode combinatorial tasks has now been identified by the community. A recent survey on the use of such graph neural networks for combinatorial tasks can be found in [69].

	CPLEX	binary	count	bin_dual	dual
micro_asym_T12	434.3	-93.7 (K=2)	-74.2 (K=2)	-101.2 (K=5)	**-103.6 (K=4)**
micro_bal_T12	329.8	**-6.9 (K=1)**	+21.2 (K=4)	+1.2 (K=1)	-0.7 (K=1)
micro_asym_T24	1325.8	-12.6 (K=1)	+142.7 (K=2)	+21.9 (K=1)	**-26.5 (K=1)**
micro_bal_T24	13474.9	**-6525.1 (K=5)**	-5625.8 (K=4)	-5530.4 (K=4)	-3765.7 (K=5)
hydro_fix_2	**446.0**	+37.4 (K=1)	+37.9 (K=1)	+35.1 (K=1)	+37.6 (K=1)
hydro_var_2	1628.6	-261.5 (K=3)	-257.4 (K=3)	-108.0 (K=3)	**-265.7 (K=5)**
hydro_var_3	28678.8	-10728.2 (K = 2)	**-12786.3** (K = 1)	-8748.5 (K = 1)	-9020.1 (K = 1)

Table 7.1: Average tree size difference between graph heuristics and `CPLEX`. For each problem, the best value of K is considered for each heuristic.



(a) `micro_asym_T12`

(b) `micro_bal_T12`

(c) `micro_asym_T24`

(d) `micro_bal_T24`

(e) `hydro_fix_2`

(f) `hydro_var_2`

(g) `hydro_var_3`

Figure 7.3: Average tree size difference between graph heuristics and `CPLEX`, displayed per quantile of the `CPLEX`'s distribution. Each curve represents a pair (influence graph, K). We do not identify each graph as no pattern emerges.

Chapter 8

A Decomposition-Coordination Approach

Content

Solving MILPs can turn out to be prohibitively computationally expensive. Decomposition methods may be used to leverage the structure of a particular problem, hence alleviating the computational burden. In this chapter, we introduce a decomposition-coordination approach designed to address problems with specific structure. This structure may be known *a priori* or discovered through Spectral Clustering [101] on the primal graph.

First, Section 8.1 details the specific structure to be leveraged in this chapter and introduce the reader with the Relax and Fix procedure. A parallel is drawn with Lagrangian Decomposition to introduce the philosophy of our approach, detailed in Section 8.2. Finally, Section 8.3 reports some experimental results.

8.1 Decoupling problems

8.1.1 Setting

Before going through the method, let us specify the scope of problems we are interested in and, at the same time, some notations which will be useful in the remainder.

The problems considered in this document, as presented in Chapter 3, can be seen as the juxtaposition of dependent subproblems, linked through constraints involving a set of *coupling variables*.

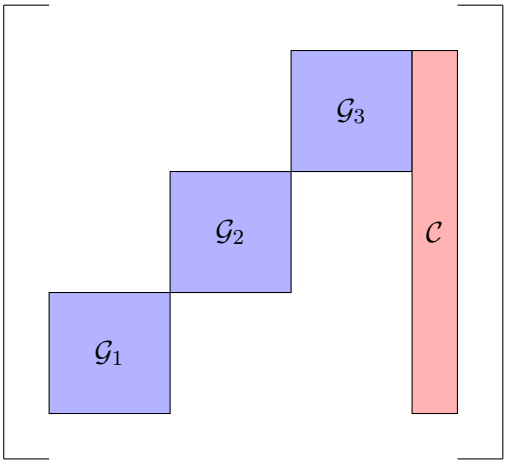
Mathematically, this structure is reflected by a block-diagonal constraint matrix, as represented in Figure 8.1. Formally, such problems can be written as

$$p \ : \begin{cases} \max\limits_{(x_{\mathcal{G}_k})_{k=1\dots K}, x_{\mathcal{C}}} & \sum\limits_{k=1}^{K} f_{\mathcal{G}_k}^\top x_{\mathcal{G}_k} + f_{\mathcal{C}}^\top x_{\mathcal{C}} \\ s.t. & A_{\mathcal{G}_k} \left[x_{\mathcal{G}_k} \ x_{\mathcal{C}}\right]^\top \leqslant b_{\mathcal{G}_k} \ , \ k \in \{1, ..., K\} \\ & x_{\mathcal{G}_k} \in X_{\mathcal{G}_k}, x_{\mathcal{C}} \in X_{\mathcal{C}} \ , \ k \in \{1, ..., K\} \end{cases} \tag{8.1}$$

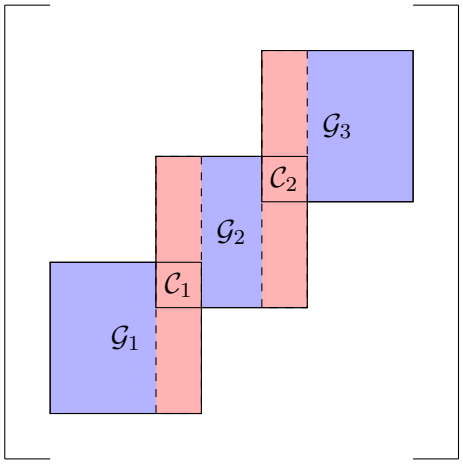
or, in a more specific case,

$$p \ : \begin{cases} \max\limits_{(x_{\mathcal{G}_k})_{k=1\dots K}, (x_{\mathcal{C}_k})_{k=1\dots K-1}} & \sum\limits_{k=1}^{K} f_{\mathcal{G}_k}^\top x_{\mathcal{G}_k} + f_{\mathcal{C}_k}^\top x_{\mathcal{C}_k} \\ s.t. & A_{\mathcal{G}_k} \left[x_{\mathcal{G}_k} \ x_{\mathcal{G}_{k+1}} \ x_{\mathcal{C}_k}\right]^\top \leqslant b_{\mathcal{G}_k} \ , \ k \in \{1, ..., K-1\} \\ & A_{\mathcal{G}_K} \left[x_{\mathcal{G}_K}\right]^\top \leqslant b_{\mathcal{G}_K} \\ & x_{\mathcal{G}_k} \in X_{\mathcal{G}_k}, \ k \in \{1, ..., K\} \\ & x_{\mathcal{C}_k} \in X_{\mathcal{C}_k}, \ k \in \{1, ..., K-1\} \end{cases} \tag{8.2}$$

where $X_{\mathcal{G}_k}$ defines the integer constraints for the set of variables $\mathcal{G}_k$: $X_{\mathcal{G}_k} = \mathbb{R}^{n_{\mathcal{G}_k}-|\mathcal{B}_{\mathcal{G}_k}|} \times \{0,1\}^{|\mathcal{B}_{\mathcal{G}_k}|}$ with $n_{\mathcal{G}_k}$ the number of variables belonging to block $\mathcal{G}_k$ and $\mathcal{B}_k$ the indices of integer variables in this same block. The variables $x_{\mathcal{C}}$ or $(x_{\mathcal{C}_k})_{k=1}^K$ in problem (8.2) are the so-called coupling variables. As an example, when a complex system is to be optimized once for multiple time steps, blocks may naturally be formed by regrouping the variables corresponding to the same time step. The coupling variables are in this case the variables appearing in the constraints which link the different time steps.
The methodology presented in this Section is inspired by these structures, but does not strictly require them to be applied.



(a) The coupling variables tie the blocks together, general case (8.1)



(b) The coupling variables tie consecutive blocks together, specific case (8.2)

Figure 8.1: Structures of interest in the constraint matrices of the considered problems. Case 8.1b is a specific case of 8.1a.

We write S the feasible set of p and P the polytope corresponding to the feasible set of the linear relaxation of p. $x_{\mathcal{C}}$ are seen here as complicating variables, in the sense that, if they were fixed, the problem would decouple as defined below.

Definition 8.1.1. *We say that a MILP decouples if its feasible set S can be rewritten as a cartesian product $S = \otimes_{k=1}^K S_k$ with $K > 1$.*
We call decomposition an ordered partition of the index set and we say that a decomposition $\mathcal{G} = \{\mathcal{G}_k\}_{k=1}^K$ decouples a MILP if its feasible set can be rewritten as
$S = \{x = [x_{\mathcal{G}_1} \ ... \ x_{\mathcal{G}_K}], \ x_{\mathcal{G}_k} \in S_k \ \forall k \in \{1, ..., K\}\}$.

The principal interest of a decoupling decomposition is that we have the equivalence between the

two problems $\max_{x \in \otimes_{k=1}^{K} S_k} f^\top x$ and $\sum_{k=1}^{K} \max_{x_{\mathcal{G}_k} \in S_k} f_{\mathcal{G}_k}^\top x_{\mathcal{G}_k}$.
The challenge addressed in the following is to take advantage of the structure exhibited in Equation (8.1) to heuristically solve decoupled sub-problems, in ways which do not harm too much the objective function. This comes with coordination and monitoring some trade-off between the granularity of the decomposition and the number of processed nodes.

8.1.2 Relax and Fix

For the sake of convenience, we introduce the notation $\rho\left(x_{\mathcal{I}_1} | x_{\mathcal{I}_2} = y\right)$ with $y \in \mathbb{R}^{|\mathcal{I}_2|}$, referring to the problem p while adding the constraint $x_{\mathcal{I}_2} = y$ for some index set $\mathcal{I}_2$ and relaxing the integrity constraints on any variable whose index does not belong to some index set $\mathcal{I}_1$:

$$\rho\left(x_{\mathcal{I}_1} | x_{\mathcal{I}_2} = y\right) \; : \begin{cases} \max\limits_{\substack{(x_{\mathcal{G}_i})_{i=1,\dots,K} \\ x_{\mathcal{C}}}} & \sum\limits_{i=1}^{K} f_{\mathcal{G}_i}^\top x_{\mathcal{G}_i} + f_{\mathcal{C}}^\top x_{\mathcal{C}} \\ s.t. & x \in P \\ & x_{\mathcal{I}_2} = y \\ & x_{\mathcal{I}_1} \in \{0,1\}^{|\mathcal{I}_1|} \end{cases} \tag{8.3}$$

$\rho\left(x_{\mathcal{I}_1}\right)$ will refer to the linear relaxation of p where only the binary variables in $\mathcal{I}_1$ are not relaxed and $\rho\left(.|x_{\mathcal{I}_2} = y\right)$ the linear relaxation of p with the additional constraint $x_{\mathcal{I}_2} = y$. This constraint is perceived here as a collection of individual constraints, and the reader should not be surprised to see notations such as $\rho\left(.|x_{\mathcal{I}_2} = y_2, x_{\mathcal{I}_1} = y_1\right)$ or again $\rho\left(.|\left\{x_{\mathcal{I}_2} = y_2\right\} \cup \left\{x_{\mathcal{I}_1} = y_1\right\}\right)$.
The Relax and Fix procedure (RF) is then a natural heuristic for finding a solution associated with, hopefully, a good lower bound. Algorithm 9 presents the basic RF procedure applied to p with any decomposition $\mathcal{G} = \{\mathcal{G}_k\}_{k=1}^{K}$ which partitions the integer variables set into disjunctive sets $\{\mathcal{B}_k\}_{k=1}^{K}$. By convention, we write $x^*(p)$ the optimal solution of a problem p obtained by any given method, and $z^*(p)$ its objective value, with $z^*(p) = -\infty$ if p is not feasible.

Algorithm 9 Relax-and-Fix: $RF(p, \mathcal{G})$

Input:
A decomposition $\{\mathcal{G}_k\}_{k=1}^K$ for a problem p with $\{\mathcal{B}_k\}_{k=1}^K$ its associated integer sets.
Initialization:
$x^{(1)} \leftarrow x^* \left(\rho \left(x_{\mathcal{B}_1}\right)\right)$
$z^{(1)} \leftarrow z^* \left(\rho \left(x_{\mathcal{B}_1}\right)\right)$
Procedure:
for k in 2...K:
$z^{(k)} \leftarrow z^* \left(\rho \left(x_{\mathcal{B}_k} | \left\{x_{\mathcal{B}_j} = x_{\mathcal{B}_j}^{(j)}\right\}_{j=1}^{k-1}\right)\right)$
if $z^{(k)} = -\infty$:
Return:
$z_{RF}^* \equiv -\infty$ and $x_{RF}^* \equiv$ `nan`
else:
$x^{(k)} \leftarrow x^* \left(\rho \left(x_{\mathcal{B}_k} | \left\{x_{\mathcal{B}_j} = x_{\mathcal{B}_j}^{(j)}\right\}_{j=1}^{k-1}\right)\right)$
end if
end for
Return:
$z_{RF}^* \equiv z^{(K)}$ and $x_{RF}^* \equiv x^{(K)}$.

A usual variant of RF is obtained by fixing only a subset of the integer variables of each block. Although it increases the odds of obtaining a feasible solution, it requires the selection of a relevant subset in each block, which is an open question (see for instance [102]). Note that this could be an opportunity for machine learning, but this axis is not developed here.
Considering the standard RF procedure as described in Algorithm 9, it provides a lower bound for p and turns out to reach optimality when the used decomposition decouples p. These points are stated in the following two propositions.

Proposition 8.1.1. *Let z^* be the optimal value of p. If $z^{(1)} > -\infty$, $z_{RF}^* \leqslant z^* \leqslant z^{(1)}$ holds.*

Proof. *Let S and S_1 be the feasible sets of respectively p and $\rho\left(x_{\mathcal{B}_1}\right)$. If $z^{(1)} > -\infty$, S_1 is non-empty. As $\rho\left(x_{\mathcal{B}_1}\right)$ is a relaxation of p, the two problems share the same objective function and $S \subseteq S_1$, which implies $z^{(1)} \geqslant z^*$. Conversely, with S_k the feasible set of $\rho\left(x_{\mathcal{B}_k} | \left\{x_{\mathcal{B}_j} = x_{\mathcal{B}_j}^{(j)}\right\}_{j=1}^{k-1}\right)$, we have* $S_k = \left\{x \in S \mid \left\{x_{\mathcal{B}_j} = x_{\mathcal{B}_j}^{(j)}\right\}_{j=1}^{k-1}\right\} \implies S_k \subseteq S \implies z^* \geqslant z_{RF}^*$. □

Proposition 8.1.2. *When a problem p decouples, a decomposition $\mathcal{G}$ exists such that $RF(p, \mathcal{G})$ is optimal.*

Proof. *Let S be the feasible set for p and $\mathcal{G} = \{\mathcal{G}_k\}_{k=1}^K$ be a decoupling decomposition such that $S = \otimes_{k=1}^K S_{\mathcal{G}_k}$ and $P_{\mathcal{G}_k}$ is the set $S_{\mathcal{G}_k}$ without integrity constraints. Considering the case $K = 2$ and only considering bounded sets, the RF problem for p can be rewritten as*

$$\max_{x_{\mathcal{G}_2} \in S_{\mathcal{G}_2}} \left\{ f_{\mathcal{G}_2}^\top x_{\mathcal{G}_2} + \max_{x_{\mathcal{G}_1} \in S_{\mathcal{G}_1}, y \in P_{\mathcal{G}_2}} \left\{ f_{\mathcal{G}_1}^\top x_{\mathcal{G}_1} + f_{\mathcal{G}_2}^\top y \right\} - f_{\mathcal{G}_2}^\top \tilde{y} \right\}$$

with $\tilde{y}$ the solution for group $\mathcal{G}_2$ in the inner problem. $\mathcal{G}$ decouples the inner problem when it decouples p, therefore it is equivalent to

$$\begin{aligned}
& \max_{x_{\mathcal{G}_2} \in S_{\mathcal{G}_2}} \left\{ f_{\mathcal{G}_2}^\top x_{\mathcal{G}_2} + \max_{x_{\mathcal{G}_1} \in S_{\mathcal{G}_1}} \left\{ f_{\mathcal{G}_1}^\top x_{\mathcal{G}_1} \right\} + f_{\mathcal{G}_2}^\top \tilde{y} - f_{\mathcal{G}_2}^\top \tilde{y} \right\} \\
\iff & \max_{x_{\mathcal{G}_2} \in S_{\mathcal{G}_2}} \left\{ f_{\mathcal{G}_2}^\top x_{\mathcal{G}_2} + \max_{x_{\mathcal{G}_1} \in S_{\mathcal{G}_1}} \left\{ f_{\mathcal{G}_1}^\top x_{\mathcal{G}_1} \right\} \right\} \\
\iff & \max_{x_{\mathcal{G}_2} \in S_{\mathcal{G}_2}} \left\{ f_{\mathcal{G}_2}^\top x_{\mathcal{G}_2} \right\} + \max_{x_{\mathcal{G}_1} \in S_{\mathcal{G}_1}} \left\{ f_{\mathcal{G}_1}^\top x_{\mathcal{G}_1} \right\} \\
\iff & \max_{x \in S} \left\{ f_{\mathcal{G}_1}^\top x_{\mathcal{G}_1} + f_{\mathcal{G}_2}^\top x_{\mathcal{G}_2} \right\} \quad \textit{(as } \mathcal{G} \textit{ decouples } p\textit{)}
\end{aligned}$$

The result can be extended to any $K > 2$ by induction. □

8.1.3 Parallel with Lagrangian Decomposition

In the following, we introduce a link between Relax and Fix and a constrained version of Lagrangian Decomposition (LD [103]), which will help us to introduce the philosophy of our methodology. We take $K = 2$ for the sake of conciseness.

Without assuming any special structure, any problem p can always be rewritten as

$$p \ : \ \begin{cases} \max\limits_{x} & f^\top x \\ s.t. & x \in S_1 \cap S_2 \end{cases} \tag{8.4}$$

with $S_k = \{x \in P \mid x_{\mathcal{G}_k} \in X_{\mathcal{G}_k}\}$ for $k = 1, 2$ with $\{\mathcal{G}_1, \mathcal{G}_2\}$ a partition of the index set. The Lagrangian decomposition on the two sets S_1, S_2 comes down to solving the problem

$$(LD) \ : \ \min_{\lambda \in \Lambda \equiv \mathbb{R}^n} \left(\begin{cases} \max\limits_{x} & (f - \lambda)^\top x \\ s.t. & x \in S_1 \end{cases} + \begin{cases} \max\limits_{x} & \lambda^\top x \\ s.t. & x \in S_2 \end{cases} \right)$$

It is straightforward to see that LD provides an upper bound for $z^*(p)$. Let us assume that λ exists such that $z^*\left(\{\max_x (f - \lambda)^\top x \mid x \in S_1\}\right) + z^*\left(\{\max_x \lambda^\top x \mid x \in S_2\}\right) < z^*\left(\{\max_x f^\top x \mid x \in S_1 \cap S_2\}\right)$. Taking x^* the solution of the right-hand-side problem and acknowledging that $x^* \in S_1 \cap S_2$, we obtain

the contradiction $(f-\lambda)^\top x^* + \lambda^\top x^* < f^\top x^*$.
Note that the set $\Lambda = \mathbb{R}^n$ can be safely restricted to $\Lambda = \{\lambda \in \mathbb{R}^n,\ \lambda \geqslant 0\}$.

Lagrangian decomposition has two main drawbacks. First, it gives no guarantee of finding an integer solution satisfying both S_1 and S_2, it only aims at providing an upper bound for the solution in S. Second, the number of multipliers (the dimension of λ, n here) can be large enough to make the search over λ prohibitive.
In this context, Relax and Fix may appear as a way of tackling heuristically the first issue. Writing $p_1 : \left\{\max_x f^\top x \mid x \in S_1\right\}$, RF can be formulated as

$$0 \times \begin{cases} \max\limits_x & \frac{1}{2} f^\top x \\ s.t. & x \in S_1 \end{cases} + 2 \times \begin{cases} \max\limits_x & \frac{1}{2} f^\top x \\ s.t. & x_{\mathcal{B}_1} = x^*(p_1)_{\mathcal{B}_1} \\ & x \in S_2 \end{cases} \tag{8.5}$$

They are three differences compared with Lagrangian decomposition. First, the search for λ over $\mathbb{R}^n$ is discarded by considering only the case $\lambda = \frac{f}{2}$ (or more generally $\lambda = \alpha f$, $\alpha > 0$). Second, the constraint $x_{\mathcal{B}_1} = x^*(p_1)_{\mathcal{B}_1}$ is added to enforce the feasibility of the potential solution. Last, the two problems are weighted so as to give all the weight to the second problem. Note that the feasibility constraint on its own already invalids the upper bound property.
We see through this comparison that Relax and Fix is a very drastic way to obtain a lower bound, especially by heavily restricting the search over the set of multipliers λ. Besides, one has no guarantee of actually finding such lower bound, as $x^*(p_1)_{\mathcal{G}_1}$ may not be compatible with S_2. Therefore, such approach may not be efficient in many cases. In the following, we investigate a Relax and Fix scheme similar to Equation (8.5) but using a less restrictive multiplier set. The multipliers are seen as a potential way of guiding the solution of the first problem towards the optimal solution, giving rise to the optimization problem

$$\max_{\lambda \in \Lambda} \left(0 \times \begin{cases} \max\limits_x & (f-\lambda)^\top x \\ s.t. & x \in S_1 \end{cases} + 1 \times \begin{cases} \max\limits_x & f^\top x \\ s.t. & x_{\mathcal{B}_1} = x^*\left(p_1(\lambda)\right)_{\mathcal{B}_1} \\ & x \in S_2 \end{cases} \right) \tag{8.6}$$

where $p_1(\lambda)$ refers to the first inner problem for a given value of λ. As stated in Proposition 8.1.3, this is equivalent to p when $\Lambda = \mathbb{R}^n$.

Lemma 8.1.1. *Let $x \in \{0,1\}^n \times \mathbb{R}^m$ and $\mathcal{S} = \left\{x^{(j)} \in [0,1]^n\right\}_{j=1,\dots,q} \times \mathcal{F}$ with $\mathcal{F}$ some countable and finite set in $\mathbb{R}^m$, then $x \in conv(\mathcal{S})$ implies that an extreme point y of $conv(\mathcal{S})$ exists such that $y_{:n} = x_{:n}$ where $x_{:n}$ refers to the first n coordinates of x.*

Proof of Lemma 8.1.1. *Let us consider wlog that $x = \sum_{j=1}^{q} \lambda^{(j)} y^{(j)}$ with $\lambda^{(j)} \in (0,1]$, $\sum_{j=1}^{q} \lambda^{(j)} \leqslant 1$, and $y^{(j)} \in \mathcal{E}(\mathcal{S})$ for any $j \in \{1,\dots,q\}$ with $\mathcal{E}(S)$ the set of extreme points of $conv(S)$ and show that j exists such that $y^{(j)}_{:n} = x_{:n}$.*
Considering the k-th coordinate with $k \leqslant n$:

1. *if $x_k = 1$, take $i \in \{1,\dots,q\}$, then $1 = \lambda^{(i)} y_k^{(i)} + \sum_{j\neq i} \lambda^{(j)} y_k^{(j)} \implies \lambda^{(i)} y_k^{(i)} = 1 - \sum_{j\neq i} \lambda^{(j)} y_k^{(j)}$. Besides, $y_k^{(i)} > 0$ as otherwise we would have $1 = \sum_{j\neq i} \lambda^{(j)} y_k^{(j)} \implies 1 = \sum_{j\neq i} \lambda^{(j)} \implies \lambda^{(i)} = 0$ which is impossible by assumption.*
   If $y_k^{(i)} < 1$, we then have $\lambda^{(i)} > 1 - \sum_{j\neq i} \lambda^{(j)} y_k^{(j)} > 1 - \sum_{j\neq i} \lambda^{(j)}$ as $y_k^{(j)} \leqslant 1$ for any j, hence $\sum_{j=1}^{q} \lambda^{(j)} > 1$ which is impossible. Therefore $y_k^{(i)} = y_k^{(j)} = x_k = 1$ for any $j = 1,\dots,q$.

2. *if $x_k = 0$, take $i \in \{1,\dots,q\}$, then $0 = \sum_{j=1}^{q} \lambda^{(j)} y_k^{(j)} \geqslant \lambda^{(i)} y_k^{(i)}$ which implies $y_k^{(i)} = 0$. Therefore $y_k^{(i)} = y_k^{(j)} = x_k = 0$ for any $j = 1,\dots,q$.*

□

Proposition 8.1.3. *Problems (8.6) and p as defined in (8.4) are equivalent when $\Lambda = \mathbb{R}^n$.*

Proof. *Let us denote the two inner problems in program (8.6) $p_1(\lambda)$ and $p_2(\lambda)$ so that it can be rewritten as $\max_{\lambda\in\mathbb{R}^n} \{0 \times z^*(p_1(\lambda)) + 1 \times z^*(p_2(\lambda))\}$.*
First, we know that problem (8.6) yields a lower bound for p as the feasible set of $p_2(\lambda)$ is a subset of S for any $\lambda \in \mathbb{R}^n$. Let us show that a specific value λ^* gives an optimal solution.
By construction, $x^* \in S_1$ so we also have $x^* \in conv(S_1)$. In other words, x^* is a convex combination of the extreme points $\mathcal{E}(S_1)$ of $conv(S_1)$. Lemma 8.1.1 implies that $\mathcal{E}(S_1) \cap \left\{x \in \mathbb{R}^n \mid x_{\mathcal{B}_1} = x^*_{\mathcal{B}_1}\right\} \neq \varnothing$. In that case, taking λ^* such that $\lambda^*_j = 1$ (resp. $\lambda^*_j = -1$) if $x^*_j = 1$ (resp. $x^*_j = 0$) for any j in $\mathcal{B}_1$ and 0 elsewhere necessarily gives $x^*(p_1(\lambda^*))_{\mathcal{B}_1} = x^*_{\mathcal{B}_1}$. As a consequence, $x^*(p_2(\lambda^*)) = x^*$ by construction.

□

Combining Relax and Fix and Lagrangian decomposition, we obtained an equivalent formulation of any MILP which is decomposed in two subproblems, supposed to be easier to solve than p. However,

the cost of such approach is that one has to search over the multipliers space Λ to guarantee optimality, which is in general a very high cost to pay. In the following, we derive an approach to reduce such search in our setting of nearly decoupled problems.

8.2 Decouple, Relax and Fix

8.2.1 The generic methodology

Taking as starting point the problem (8.6), we aim at reducing the search space Λ to a finite set of interesting points, if possible countable with low cardinality. This would allow us to perform a full exploration of this set, hence avoiding to rely on methods such as gradient descent, the efficiency of which may be uncertain on high dimensional spaces.
We consider in the following with no loss of generality the case of a problem p formulated as (8.1) which we can write as

$$p \; : \begin{cases} \max\limits_{(x_{\mathcal{G}_k})_{k=1\dots K}, x_{\mathcal{C}}} & \sum\limits_{k=1}^{K} f_{\mathcal{G}_k}^{\top} x_{\mathcal{G}_k} + f_{\mathcal{C}}^{\top} x_{\mathcal{C}} \\ s.t. & x_{\mathcal{G}_k} \in S_k(x_{\mathcal{C}}) = P_{\mathcal{G}_k}(x_{\mathcal{C}}) \cap X_{\mathcal{G}_k} \\ & x_{\mathcal{C}} \in X_{\mathcal{C}} \end{cases} \tag{8.7}$$

with $P_{\mathcal{G}_k}(x_{\mathcal{C}}) = \left\{ x \in \mathbb{R}^{|\mathcal{G}_k|} \mid A_{\mathcal{G}_k} \left[x_{\mathcal{G}_k} \; x_{\mathcal{C}} \right]^{\top} \leqslant b_{\mathcal{G}_k} \right\}$.
The challenge addressed in the following is that of reducing the initial search space $\Lambda = \mathbb{R}^n$ while keeping the solution optimal, or at least not decreasing it too much. To this end, we switch the lens used so far and, rather than using multipliers to guide the first inner problem toward the optimal solution, we will directly search over the variables' domain. We introduce in Proposition 8.2.1 the generic formulation of the program we will consider in the remainder. Named Decouple, Relax and Fix (DRF), this program is designed to be equivalent to the initial problem p due to the structure of p.

Proposition 8.2.1. *When considering $K = 2$, problem p as defined equivalently in Equations* (8.7)

and (8.6) *is equivalent to the Decouple, Relax and Fix problem (DRF):*

$$(DRF) \;:\; \max_{x_c \in X_{\mathcal{C}}} \left(0 \times \left\{ \begin{array}{ll} \max\limits_{x} & f^\top x \\ s.t. & x_{\mathcal{C}} = x_c \\ & x_{\mathcal{G}_2} \in P_{\mathcal{G}_2}(x_c) \\ & x_{\mathcal{G}_1} \in S_1(x_c) \end{array} \right. + 1 \times \left\{ \begin{array}{ll} \max\limits_{x} & f^\top x \\ s.t. & x_{\mathcal{B}_1} = x^*(p_1)_{\mathcal{B}_1} \\ & x_{\mathcal{C}} = x_c \\ & x_{\mathcal{G}_2} \in S_2(x_c) \end{array} \right. \right) \tag{8.8}$$

Proof. *Let us denote $p_1(x_c)$ and $p_2(x_c)$ the two inner problems in DRF so that it can be rewritten as* $\max_{x_c \in X_{\mathcal{C}}} \{0 \times z^*(p_1(x_c)) + 1 \times z^*(p_2(x_c))\}$.
First, we know that DRF yields a lower bound for p as the feasible set of $p_2(x_c)$ is a subset of S for any $x_c \in X_{\mathcal{C}}$. Let us show that if we take $x_c = x^*_{\mathcal{C}}$, then (DRF) yields an optimal solution of p with x^* any optimal solution for p.
When considering a fixed value x_c for the coupling variables, $p_1(x_c)$ can be rewritten as
$\max_{x_{\mathcal{G}_1}, x_{\mathcal{G}_2}} \left\{ f^\top_{\mathcal{G}_1} x_{\mathcal{G}_1} + f^\top_{\mathcal{G}_2} x_{\mathcal{G}_2} + f^\top_{\mathcal{C}} x_c \mid [x_{\mathcal{G}_1}\ x_{\mathcal{G}_2}] \in S_1(x_c) \times P_{\mathcal{G}_2}(x_c) \right\}$ *and p as*
$\max_{x_{\mathcal{G}_1}, x_{\mathcal{G}_2}} \left\{ f^\top_{\mathcal{G}_1} x_{\mathcal{G}_1} + f^\top_{\mathcal{G}_2} x_{\mathcal{G}_2} + f^\top_{\mathcal{C}} x_c \mid [x_{\mathcal{G}_1}\ x_{\mathcal{G}_2}] \in S_1(x_c) \times S_2(x_c) \right\}$.
Hence, $\{\mathcal{G}_1, \mathcal{G}_2\}$ decouples these two problems, which are thus respectively equivalent to
$\left\{ f^\top_{\mathcal{C}} x_c + \max_{x_{\mathcal{G}_1}} \left\{ f^\top_{\mathcal{G}_1} x_{\mathcal{G}_1} \mid x_{\mathcal{G}_1} \in S_1(x_c) \right\} + \max_{x_{\mathcal{G}_2}} \left\{ f^\top_{\mathcal{G}_2} x_{\mathcal{G}_2} \mid x_{\mathcal{G}_2} \in P_{\mathcal{G}_2}(x_c) \right\} \right\}$ *and*
$\left\{ f^\top_{\mathcal{C}} x_c + \max_{x_{\mathcal{G}_1}} \left\{ f^\top_{\mathcal{G}_1} x_{\mathcal{G}_1} \mid x_{\mathcal{G}_1} \in S_1(x_c) \right\} + \max_{x_{\mathcal{G}_2}} \left\{ f^\top_{\mathcal{G}_2} x_{\mathcal{G}_2} \mid x_{\mathcal{G}_2} \in S_2(x_c) \right\} \right\}$.
The two problems in $x_{\mathcal{G}_1}$ are identical and thus yield solutions with equal objective values for a given x_c. Considering the value $x^*_{\mathcal{C}}$, we obtain $f^\top_{\mathcal{G}_1} x^* (p_1(x^*_{\mathcal{C}}))_{\mathcal{G}_1} = f^\top_{\mathcal{G}_1} x^*_{\mathcal{G}_1}$ and thus $f^\top x^* (p_2(x^*_{\mathcal{C}})) = f^\top x^*$ as the problem (8.7) *is decoupled by $\{\mathcal{G}_1, \mathcal{G}_2\}$ once the coupling variables are fixed.* □

The only difference with *RF* is the introduction of the master problem over $X_{\mathcal{C}}$, which guarantees the equivalence with our initial structured problem. If we write $p(x_c)$ the problem p augmented with the constraint $x_{\mathcal{C}} = x_c$, *DRF* applied to p with the decomposition $\mathcal{G}$ can be written $\max_{x_c \in X_{\mathcal{C}}} \{RF(p(x_c), \mathcal{G})\}$.
To properly define a *DRF* procedure, one still needs to characterize the search over $X_{\mathcal{C}}$. In the following, we incorporate the *RF* procedure in a Branch and Bound algorithm on the set of coupling variables to solve the *DRF* problem (8.8). For the sake of simplicity, we restrict to the cases where the coupling variables are either binary or continuous.

8.2.2 Decouple, Relax and Fix with binary coupling variables

From now on, the decomposition used is the one naturally induced by the formulations (8.1) or (8.2), then $RF(s)$ will stand for $RF\left(s, \mathcal{G} = \{\mathcal{G}_k\}_{k=1}^{K}\right)$ with s a subproblem of p as defined in Equation (8.7). Considering the case where coupling variables are only binary, we propose in this section an algorithm, referred to as Decouple, Relax and Fix for Binary coupling variables ($DRFB$) to solve the DRF problem (8.8). The search over $X_{\mathcal{C}}$ is embedded in a B&B procedure by first branching on coupling variables and solving the decoupled sub-problems separately, as presented in Algorithm 10. It is illustrated in Figure 8.2.

Definition 8.2.1. *Decoupling tree.*
We call decoupling tree the subtree rooted in p and built from branching on the set of binary coupling variables $\mathcal{C}$ only. We write $\mathcal{L}_{\mathcal{C}}$ the set of leaves of such tree.

Algorithm 10 Decouple, Relax and Fix for Binary Coupling Variables (DRFB)

Branch only on binary coupling variables $\mathcal{C}$ to build the decoupling tree by B&B.
When a decoupling leaf s is met, set $z_s = RF(s)$ and fathom s.
Continue until the entire decoupling tree is fathomed and set $\mathcal{L}_{\mathcal{C}}$ the set of leaves.
$z^* = \max_{s \in \mathcal{L}_{\mathcal{C}}} \{z_s\}$.
Return: z^*.

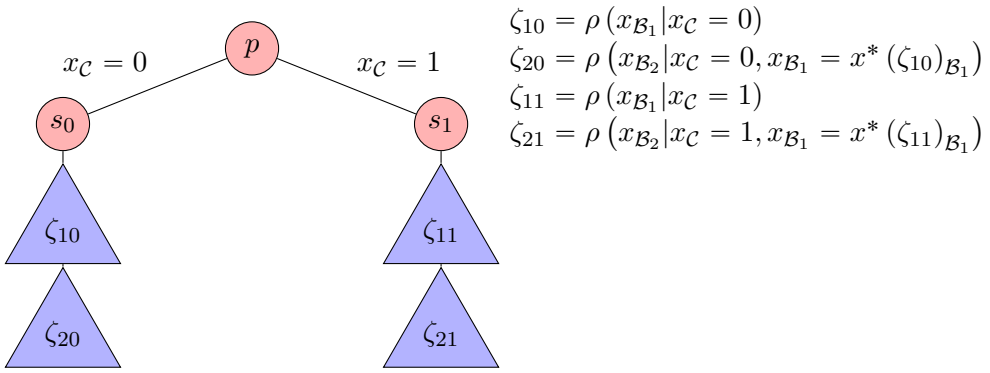


Figure 8.2: Tree created by $DRFB$ in case of a unique binary coupling variable for $K = 2$. Red circles represent nodes of the decoupling tree and blue triangles are rooted subproblems solved separately by Branch and Bound.

We make here some remarks on this algorithm, which naturally stems from the use of B&B in our setting.

Remark 1. Optimality guarantee

When coupling variables are only binary, the problem solved by $DRFB$ is exactly DRF, as the feasible set $X_{\mathcal{C}}$ is entirely explored unless suboptimality or infeasibility is proven. Hence $DRFB$ is optimal by Proposition 8.2.1.

However, it is important to note that the optimality guarantee is lost as soon as some coupling variables are continuous. This is shown by the following counter example, with $\mathcal{C} = \{0\}$, $\mathcal{G}_1 = \{1\}$, $\mathcal{G}_2 = \{2\}$.

$$p : \begin{cases} \max\limits_{x_0,x_1,x_2} & 0.5x_0 + 2x_1 - 2x_2 \\ s.t. & x_1 + x_0 \leqslant 1 \\ & x_2 + x_0 \geqslant 0.5 \\ & (x_1, x_2) \in \{0,1\}^2, x_0 \in [0,1] \end{cases}$$

The solution for $RF\,(p, \{1,2\})$ is [0 1 1] with value 0, whereas the solution for p is [1 0 0] with value 0.5.

Remark 2. Primal bounds

The LP solution of any visited node in the whole procedure can be used for pruning. Let us exhibit the different cases with $K = 2$, taking s the subproblem associated to:

1. a node ζ_1 in the decoupling tree. Any RF procedure under ζ_1 only solves LP relaxations of subproblems of s. Thus the LP solution at ζ_1 is a valid upper bound to be used for pruning;

2. a node ζ_2 in the second stage of a RF procedure (when $k = 2$ in Algorithm 9, *i.e.* when solving the second problem in the procedure, the variables of the first block being fixed), the LP solution is valid as usual in B&B;

3. a node ζ_3 in the first stage of a RF procedure. Two cases appear: either the optimal solution of this stage is or is not to be found at a descendant node of ζ_3. If it is not, it can be pruned safely. If it is, it is enough to remark that first stage's descendants are naturally associated to subproblems of ζ_3, but also are all nodes of the second stage.

8.2.3 Decouple, Relax and Fix with continuous coupling variables: $K = 2$

We now consider the case where the number of blocks is $K = 2$ and the coupling variables are only continuous. In this context, $X_{\mathcal{C}}$ is an interval in $\mathbb{R}^{|\mathcal{C}|}$ and therefore cannot be explored exhaustively as done previously. A natural solution for guaranteeing the optimality of the solution is to discretize the continuous coupling variables onto a sufficiently narrow grid $\mathcal{H} \subset \mathbb{R}^{|\mathcal{C}|}$, which gives rise to the Decouple, Relax and Fix for Continuous coupling variables ($DRFC_{\mathcal{G},\mathcal{C}}$), presented in Algorithm 12. In the following, $GRF_{\mathcal{G},\mathcal{C}}$ refers to the function defined by Algorithm 11 when applied to a problem p and a grid $\mathcal{H}$ with decomposition $\mathcal{G}$ and coupling variables $\mathcal{C}$. When the choice of a decomposition and coupling variables raises no doubt, we omit it to alleviate the notations and only refer to $DRFC$ and GRF.

Algorithm 11 Guided Relax-and-Fix ($GRF_{\mathcal{G},\mathcal{C}}$)

Input:
$u \in \mathcal{H}$, $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, $\mathcal{C}$
Procedure:
$z^*_{GRF} \leftarrow z^* \left(\rho \left(x_{\mathcal{B}_1} | x_{\mathcal{C}} = u\right)\right)$.
if $z^*_{GRF} = -\infty$:
Return z^*_{GRF}
else:
$x^1 \leftarrow x^* \left(\rho \left(x_{\mathcal{B}_1} | x_{\mathcal{C}} = u\right)\right)$
$z^*_{GRF} \leftarrow z^* \left(\rho \left(x_{\mathcal{B}_2} | x_{\mathcal{B}_1} = x^1_{\mathcal{B}_1}\right)\right)$.
Return: z^*_{GRF}
end if

Algorithm 12 Decouple, Relax and Fix for Continuous Coupling Variables ($DRFC_{\mathcal{G},\mathcal{C}}$)

Input: $\mathcal{H}$, $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, $\mathcal{C}$
for u in $\mathcal{H}$ **do:**
$z^u_{GRF} \leftarrow GRF_{\mathcal{G},\mathcal{C}}(u)$
end for
$z^* \equiv \max_{u \in \mathcal{H}} z^u_{GRF}$.
Return: z^*

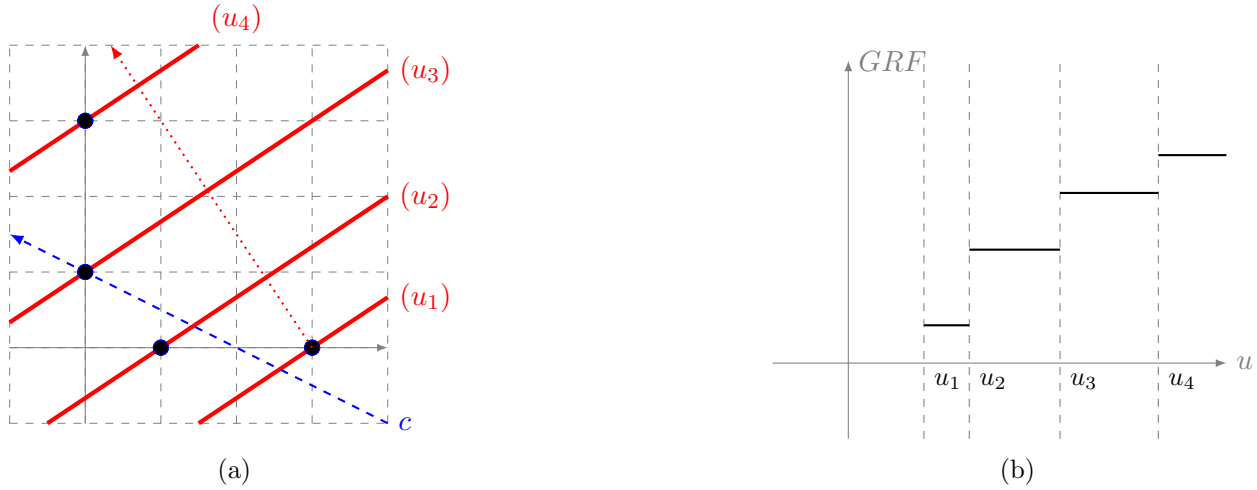
Such approach suffers heavily from the curse of dimensionality. If we consider a uniform grid $\mathcal{H}$ of step-size L, the number of GRF evaluations becomes $|\mathcal{H}| = L^{|\mathcal{C}|}$. Therefore, our interest in the following will be to reduce the decoupling grid size $|\mathcal{H}|$ used in $DRFC$. To this aim, let us exhibit some characteristics of GRF as a function of the fixed value $x_{\mathcal{C}} = u$.

Sensitivity Analysis

First, let us focus on the linear relaxation $\rho(.|x_{\mathcal{C}} = u)$ of the first problem that is to be solved when calling GRF. The consequence of modifying the constraint $x_{\mathcal{C}} = u$ is a translation of the right-hand side, in a single or most likely multiple constraints. Sensitivity analysis in Linear Programming tells us that the effect of such translation depends on whether the affected constraints are binding or not. The case of a single affected constraint is simpler to interpret. If the constraint is binding, a change in u will always (except in some degenerated cases) affect the solution of $\rho(.|x_{\mathcal{C}} = u)$, decreasing its value if the constraint is tightened and vice versa. If the affected constraint is non-binding, the value may be decreased if the tightening is strong enough to make the constraint becomes binding – on the contrary, the value is not affected if the change in u is small enough. The constraints' translation is linear in u, which in turn makes the function $z^*(\rho(.|x_{\mathcal{C}} = u))$ continuous (Lipschitz continuity is stated by Theorem 2.4 in [104]), piecewise linear and concave in u (see Proposition 2.3 in [105] and Figure 8.4 for an illustration).

Let us see how this information can be leveraged in the context of the GRF procedure. In Algorithm 11, the only effect of a change in u which matters is its impact on the integer solution of $\rho(x_{\mathcal{B}_1}|x_{\mathcal{C}} = u)$. Since the constraints translation is linear and $z^*(\rho(x_{\mathcal{B}_1}|x_{\mathcal{C}} = u))$ is piecewise linear in u, the integer solution $x^*(\rho(x_{\mathcal{B}_1}|x_{\mathcal{C}} = u))_{\mathcal{B}_1}$ is piecewise constant due to the integrity constraints (assuming the considered solver will always find the same solution in case multiple optimal solutions exist). As a consequence, $GRF(u)$ is also piecewise constant as the second inner problem in GRF (*i.e.* $\rho(x_{\mathcal{B}_2}|x_{\mathcal{B}_1} = y_{\mathcal{B}_1})$) remains identical for two values of u which give the same integer solution $x^*(\rho(x_{\mathcal{B}_1}|x_{\mathcal{C}} = u))_{\mathcal{B}_1}$. This is illustrated in Figure 8.3 and a comparison with the LP case on a concrete example is given in Figure 8.4 for an instance of `micro_asym_T12`.

Note here that, contrary to what is suggested by Figure 8.3, GRF is not necessarily monotonic (see Figure 8.4). Besides, two different plateaus may *a priori* have the same value and discontinuities are not determined only by new feasible integer solutions, as detailed in the following.



(a) (b)

Figure 8.3: Illustration of the fact that GRF is piecewise constant. Black dots in Figure 8.3a represent the feasible set for a maximisation problem, the objective direction being represented by the blue arrow. The effect of changing the value of u is a translation of some linear constraints, represented by the red lines discarding any point above. In Figure 8.3b, GRF jumps whenever a new solution induced by the translated constraints is found.

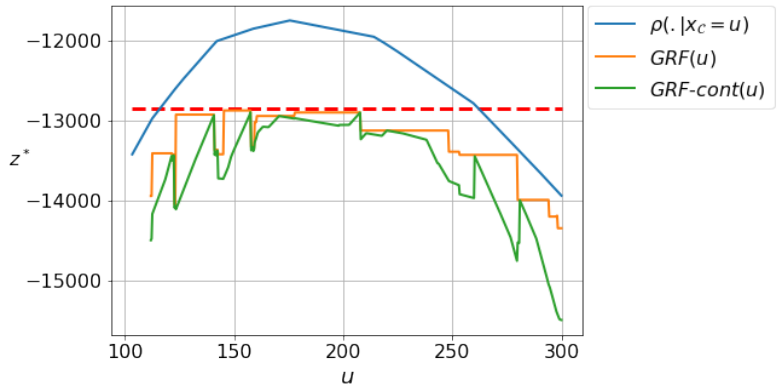


Figure 8.4: This figure illustrates on a `micro_asym_T12` instance the effect of fixing a unique continuous coupling variable to some value u (namely the mid-horizon inventory when considering a two-block temporal partition). `LP` is the value of the linear relaxation as a function of u, `GRF` is the value returned by Algorithm 11 (*i.e.* $z^*\left(\rho\left(x_{\mathcal{B}_2}|x_{\mathcal{B}_1} = y_{\mathcal{B}_1}\right)\right)$) and `GRF-cont` is the value of Algorithm 11 when adding the constraint $x_{\mathcal{C}} = u$ in the second inner problem (*i.e.* $z^*\left(\rho\left(x_{\mathcal{B}_2}|x_{\mathcal{B}_1} = y_{\mathcal{B}_1}, x_{\mathcal{C}} = u\right)\right)$).

Our method will rely on an exploration of a discrete subset of the space $X_{\mathcal{C}}$ guided by the structure of the GRF function. To simplify the exposition, we only consider the case of a unique continuous coupling variable $\mathcal{C} = \{0\}$. If we write α, β the bounds of x_0, we seek a grid $\mathcal{H}$ in $[\alpha, \beta]$ of limited length which allows to find an optimal solution in $DRFC$. In other words, we want to guide the search

in $X_{\mathcal{C}}$ with the aim of evaluating once $GRF(u) \in \max_{v\in[\alpha,\beta]} GRF(v)$ while keeping low the number of evaluations.

At this point, we acknowledge that the choice of discretizing $X_{\mathcal{C}}$ may harm the quality of the solution in some extreme cases. This is illustrated by Proposition 8.2.2, which states that some plateaus may actually be isolated points, even when considering the interior of $X_{\mathcal{C}}$.

Proposition 8.2.2. *One can build pathological cases where some plateaus of GRF measure zero. As a consequence, discretizing $X_{\mathcal{C}}$ does not guarantee to find an optimal solution through DRFC.*

Proof. *Let us consider the problem*

$$\begin{cases} \max\limits_{x,u} & x_1 + x_2 - x_3 + x_4 \\ s.t. & x_1 + x_2 + x_3 \geqslant u \\ & x_1 + x_2 + x_3 \geqslant 4 - u \\ & x_4 \leqslant u \\ & u \in [1,3]\ , x \in \{0,1\}^4 \end{cases}$$

which is decoupled by $\{\mathcal{G}_1 = \{1,2,3\}, \mathcal{G}_2 = \{4\}\}$ when u is fixed. In this case, we obtain

$$GRF(u) = \begin{cases} 3 & \text{if } u = 2 \\ 2 & \text{if } u \in [1,2) \cup (2,3] \end{cases}$$

□

Exploration of $X_{\mathcal{C}}$

In order to build a relevant grid limited in size, we take advantage of the fact that GRF is piecewise constant on $[\alpha, \beta]$. This structure is convenient for our case of study as, in this setting, a point of $\mathcal{H}$ does not need to be evaluated as soon as another point belonging to the same plateau has already been evaluated. Thus, the objective is to run GRF a minimal number of times per plateau. Such methodology guarantees optimality by Proposition 8.2.3 provided GRF exhibits no isolated point.

Proposition 8.2.3. *Let z^* be the optimal value for a problem p with $\mathcal{C} = \{0\}$, then it exists $u \in [\alpha, \beta]$ such that $GRF(u) = z^*$, with $X_{\mathcal{C}} = [\alpha, \beta]$.*

Proof. *The problem is decoupled by $\{\mathcal{G}_1, \mathcal{G}_2\}$ as soon as $x_{\mathcal{C}}$ is fixed and DRFC is optimal, see Proposition 8.2.1.* □

If the locations of the flat regions were to be known, $DRFC$ would be optimal using a non-uniform grid of size N, N being the number of plateaus. As knowing the plateau locations is unlikely, our aim is then to design a strategy for evaluating GRF on a (non-uniform) grid $\mathcal{H}$ whose size compares well with respect to N.

We know by Proposition 8.2.3 that a plateau associated with an optimal integer solution for $\mathcal{B}_1$ exists, and this plateau is found as soon as one of its extremities is found. Our method relies on the reasons why the discontinuities of GRF appear. GRF may jump for two reasons: the integer solution becomes either (i) infeasible or (ii) sub-optimal after translating the constraints. Formally, for any $u \in [\alpha, \beta]$, let $\bar{x}_{\mathcal{B}_1} = x^* \left(\rho \left(x_{\mathcal{B}_1} | x_0 = u\right)\right)_{\mathcal{B}_1}$. The first point of discontinuity of GRF at the right of u is $u + \delta$ for $\delta \in (0, \beta - u]$, with $\delta = \min\left(\delta_1, \delta_2\right)$ and

$$(i)\ \ \delta_1 = \min\left\{\lambda \in (0, \beta - u) \mid P_{\bar{x}_{\mathcal{B}_1}}\left(u + \lambda\right) = \varnothing\right\}$$

$$(ii)\ \ \delta_2 = \min\left\{\lambda \in (0, \beta - u) \mid \exists x_1 \in S_1(u + \lambda),\ \eta\left(x_1, u + \lambda\right) > \eta\left(\bar{x}_{\mathcal{B}_1}, u + \lambda\right)\right\}$$

Here, $P_{\bar{x}_{\mathcal{B}_1}}(u)$ denotes the polytope of the LP $\rho\left(. | x_0 = u, x_{\mathcal{B}_1} = \bar{x}_{\mathcal{B}_1}\right)$, and $\eta(\bar{x}_{\mathcal{B}_1}, u)$ is the optimal value of such LP: $\eta\left(\bar{x}_{\mathcal{B}_1}, u\right)) \equiv z^*\left(\rho\left(. | x_0 = u, x_{\mathcal{B}_1} = \bar{x}_{\mathcal{B}_1}\right)\right)$.
We call *first-order discontinuities* the points satisfying (i). They are easy to exhibit, in the sense that we only need to check if a polytope is empty or not (see Remark 3), which is negligible compared with evaluating GRF. On the contrary, those satisfying (ii), referred to as *second-order discontinuities*, are expensive to find, as they require a MILP solving.

Algorithm 13 finds first-order discontinuities and works as follows. For any given integer solution $\bar{x}_{\mathcal{B}_1}$ of $\rho\left(x_{\mathcal{B}_1} | x_0 = u\right)$, the left and right discontinuity points are found by checking the emptiness of $P_{\bar{x}_{\mathcal{B}_1}}(v)$ with v moving away from u. At each discontinuity, a new integer solution is computed and the process is iterated until no more new points are found.
If we write $N_1 \leqslant N$ the number of plateaus exhibited by Algorithm 13, the latter performs in the worst case $N_1 \left|H\right|$ emptiness checks and this number can be reduced to $N_1 \log_2\left(\left|H\right|\right)$ using a dichotomic search.
Note that searching for the closest discontinuity points is sufficient as $dom(\eta(x, .))$ is convex (see Proposition 8.2.4), which ensures that $P_{\bar{x}_{\mathcal{B}_1}}(u)$ cannot be empty at $u = u_2$ and non-empty at $u = u_1$

and $u = u_3$ if $u_1 < u_2 < u_3$.

Proposition 8.2.4. *Let $\bar{x}_{\mathcal{B}_1}$ be a feasible integer solution for $\rho(x_{\mathcal{B}_1})$, then $dom(\eta(\bar{x}_{\mathcal{B}_1}, .))$ is convex.*

Proof. *Let $P_{\bar{x}_{\mathcal{B}_1}}$ be the polytope of $\rho\{.|x_{\mathcal{B}_1} = \bar{x}_{\mathcal{B}_1}\}$, we must show that if $\eta(\bar{x}_{\mathcal{B}_1}, u)$ and $\eta(\bar{x}_{\mathcal{B}_1}, v)$ are defined, so is $\eta(\bar{x}_{\mathcal{B}_1}, \tau u + (1-\tau)v)$ for any τ in $[0,1]$.*
If $\eta(\bar{x}_{\mathcal{B}_1}, u)$ and $\eta(\bar{x}_{\mathcal{B}_1}, v)$ are defined, then $a \in P_{\bar{x}_{\mathcal{B}_1}}(u)$ and $b \in P_{\bar{x}_{\mathcal{B}_1}}(v)$ exist. As $P_{\bar{x}_{\mathcal{B}_1}}(w) = P_{\bar{x}_{\mathcal{B}_1}} \cap \{x \in \mathbb{R}^n \mid x_0 = w\}$ for any $w \in [\alpha, \beta]$, we have $y = \tau a + (1-\tau)b \in P_{\bar{x}_{\mathcal{B}_1}}$ by convexity of $P_{\bar{x}_{\mathcal{B}_1}}$. In addition, $y_0 = \tau a_0 + (1-\tau)b_0 = \tau u + (1-\tau)v$, i.e. $y \in P_{\bar{x}_{\mathcal{B}_1}}(\tau u + (1-\tau)v) = P_{\bar{x}_{\mathcal{B}_1}} \cap \{x \in \mathbb{R}^n \mid x_0 = \tau u + (1-\tau)v\}$, which is a bounded convex set. Thus $\eta(\bar{x}_{\mathcal{B}_1}, \tau u + (1-\tau)v)$ is defined. □

Algorithm 13 Finding first-order discontinuities for coupling variable x_0

Input:
A coupling variable x_0 with domain $[\alpha, \beta]$
Initialization:
$B = \{\alpha\}$, $V = \varnothing$
while $B \neq \varnothing$ **do**:
Let u be an element of B and set $B = B \backslash \{u\}$, $V = V \cup \{u\}$
Let $x = x^*(\rho(x_{\mathcal{B}_1}|x_0 = u))$
Set $u_1 = \max_{v \in \mathcal{H} \cap [\alpha, u)} \left\{ P_{x_{\mathcal{B}_1}}(v) = \varnothing \right\}$ ← dichotomic or linear search
if u_1 is defined and $u_1 \notin B \cup V$:
$B = B \cup \{u_1\}$
end if
Set $u_2 = \min_{v \in \mathcal{H} \cap (u, \beta]} \left\{ P_{x_{\mathcal{B}_1}}(v) = \varnothing \right\}$ ← dichotomic or linear search
if u_2 is defined and $u_2 \notin B \cup V$:
$B = B \cup \{u_2\}$
end if
end while
Return: V

Remark 3. Emptiness check for polyhedra

Checking the emptiness of a polyhedron is here considered as simple in the sense that it only requires to solve a linear program. A basic way of assessing if a polyhedron is empty is to use a variant of Farkas' lemma, also know as *Theorem of the Alternatives*: either the system $Ax \leqslant b$ for $x \in \mathbb{R}^n$ has a solution, or the system $A^\top y = 0$, $b^\top y < 0$ for $y \geqslant 0$ has a solution. Then, assessing if the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ is empty is done by solving the second linear system.

Remark 4. Bound tightening

The above considerations use the bounds of the coupling variable α, β. A naive application would be to use the bounds provided by the formulation of the considered problem. However, poor bounds may be provided and thus mechanically increase the number of emptiness checks. In practice, we use a bound tightening technique, known as OBBT (Optimization Based Bounds Tightening [106]). This method consists in maximizing and minimizing the value of each variable in the linear relaxation of the initial problem.

Remark 5. Algorithm 13 does not exhibit all first-order discontinuities

Situations may arise when a first-order discontinuity appears at an integer solution associated with a second-order discontinuity, not found by Algorithm 13. This situation is illustrated in Figure 8.5.

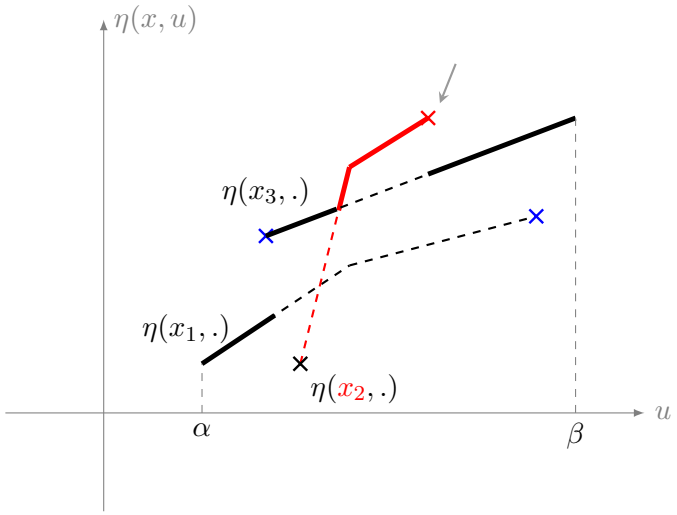


Figure 8.5: Illustration of $\eta(x,.)$ functions for three optimal integer solutions x_1, x_2, x_3. x_1 and x_3 are found by Algorithm 13 along with the associated first-order discontinuities labelled by blue crosses. However, the first-order discontinuity labelled by a red cross is not found, thus neither is x_2. This is due to the fact that the change to the plateau corresponding to x_2 when moving along u is perceived as a second-order discontinuity by the algorithm, using either x_1 or x_3 as a starting point.

Remark 6. Approximation

The number of emptiness checks can be reduced by considering Algorithm 14, which limits the search of new points to the area between already found discontinuities (rather than considering the whole grid each time).

Algorithm 14 Approximation for Algorithm 13

Input:
A coupling variable x_0 with domain $[\alpha, \beta]$
Initialization:
$B = \{\alpha\}$, $V = \varnothing$
while $B \neq \varnothing$ **do**:
Let u be an element of B and set $B = B \backslash \{u\}$, $V = V \cup \{u\}$
Let $x = x^*\left(\rho\left(x_{\mathcal{B}_1} | x_0 = u\right)\right)$
Set $v_1 = \min_{v \in \mathcal{H}} \{v > \max_{w \in V}\{w < u\}\}$, $v_2 = \max_{v \in \mathcal{H}} \{v < \min_{w \in V}\{w > u\}\}$
if v_1 is defined:
Set $u_1 = \max_{v \in \mathcal{H} \cap [v1, u)} \{P_x(v) = \varnothing\}$ ← dichotomic or linear search
if u_1 is defined and $u_1 \notin B \cup V$:
$B = B \cup \{u_1\}$
end if
end if
if v_2 is defined:
Set $u_2 = \min_{v \in \mathcal{H} \cap [v1, u)} \{P_x(v) = \varnothing\}$ ← dichotomic or linear search
if u_2 is defined and $u_2 \notin B \cup V$:
$B = B \cup \{u_2\}$
end if
end if
end while
Return: V

Detecting second-order discontinuities will then allow us to find the remaining discontinuities, as it will also enable to solve the problem raised in Remark 5. However, we argue that it is not a very concerning issue, as this type of discontinuities does not appear that often in practice compared to first-order ones if the problem has a significant level of constraints. This intuition is confirmed on a `microgrid` example in Figure 8.6.

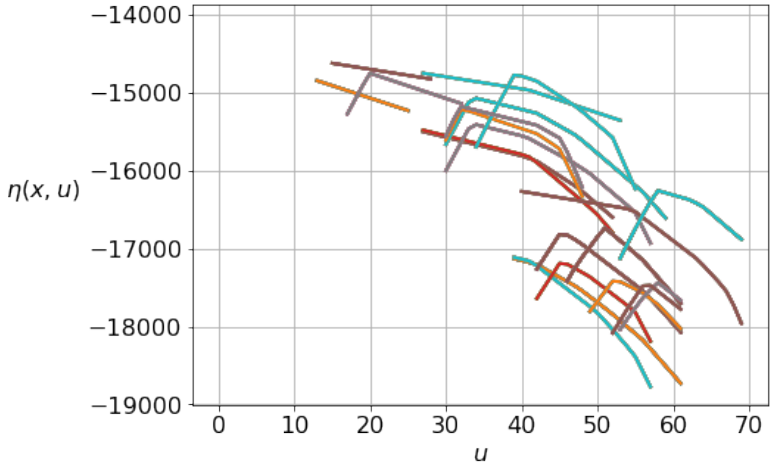


Figure 8.6: $\eta(x, u)$ for 250 different integer solutions of a `micro_asym_T12` instance, taking the mid-period inventory level as coupling variable.

Remark 7. $\eta(x, .)$ **is concave**

As suggested by Figure 8.6, $\eta(x, .)$ is concave on its domain as we show here.

We know that there exists $\lambda \geqslant 0$ such that $\eta(x, u) \equiv \max_z \left\{c^\top z | z \in P_x \cap \{x \in \mathbb{R}^n \mid x_0 = u\}\right\}$ can be rewritten as $\eta_s(x, u) = \max_z \left\{c^\top z - \lambda (x_0 - u)^2 | z \in P_x\right\} = \min_z \{f(z, u) | z \in P_x\}$ with P_x a convex set and f concave and defined over P_x. Then

$$\begin{aligned}
\eta(x, \tau u_1 + (1-\tau)u_2) &= \max_{z \in P_x} f(z, \tau u_1 + (1-\tau)u_2) \\
&\geqslant f(z, \tau u_1 + (1-\tau)u_2) \ \ \forall z \in P_x \\
&\geqslant f(\tau z_1 + (1-\tau)z_2, \tau u_1 + (1-\tau)u_2) \ \ \forall z_1, z_2 \in P_x \qquad \text{as } P_x \text{ is convex} \\
&\geqslant \tau f(z_1, u_1) + (1-\tau) f(z_2, u_2) \, \forall z_1, z_2 \in P_x \qquad \text{as } f \text{ is concave} \\
&\geqslant \tau \eta(x, u_1) + (1-\tau)\eta(x, u_2)
\end{aligned}$$

8.2.4 Generalization of $DRFC$ to any K

We saw in the previous section that continuous coupling variables is a complication for our decoupling approach, as it requires to explore a continuous space when solving DRF. Thus, we resort to approximations by considering only a finite subset of points. Similarly, DRF highly suffers from the curse of dimensionality. Therefore, an additional approximation should be made to make $DRFC$ tractable when $K > 2$. Concretely, we regard our problems using formulation (8.2) and only consider

one continuous coupling variable between two consecutive blocks. The others are associated to groups of the decomposition. Our belief is that this approximation should not be too harmful provided the considered coupling variable is selected carefully and implies structural asymmetries.

With a slight abuse of notations, we write $\mathcal{C}_k$ the selected coupling variable between groups k and $k+1$. Writing $\{\alpha_k, \beta_k\}_{k=1}^{K-1}$ the associated bounds, we then explore a subset of $X_{\mathcal{C}} = \underset{k=1}{\overset{K-1}{\times}} [\alpha_k, \beta_k]$ by using a B&B-like approach, presented in Algorithm 15. It progressively solves the relaxed subproblems while branching on the continuous coupling variables, as illustrated in Figure 8.7. To make the parallel with B&B perfectly transparent, the tree nodes are here associated to relaxed subproblems $\rho\left(x_{\mathcal{B}_k} | \left\{x_{\mathcal{B}_j} = x_{\mathcal{B}_j}^{(j-1)}\right\}_{j=1}^{k-1}, x_{\mathcal{C}_k} = u\right)$ where $x^{(j)}$ is the solution found in the ascendant node at depth j and u is a discontinuity point found by Algorithm 14 when considering $\mathcal{G}_k$ as one set of a two-member partition. We use a DFS node selection, the branching strategy being defined by the order of the decomposition. Note here that, in the same way as $DRFB$, pruning by bound is performed in $DRFC\text{-}K$.

Algorithm 15 $DRFC\text{-}K(\mathcal{G}, \mathcal{C}, k = 1, z^* = -\infty, H = \varnothing)$

Procedure:
  if $k = K$:
    $z = z^*\left(\rho\left(x_{\mathcal{G}_K} | H\right)\right)$
    $z^* = \max\{z, z^*\}$
    Return: z^*
  else (if $k < K$):
    Let $\mathcal{G}'$ be the partition $\mathcal{G}' = \left\{\mathcal{G}_k; \bigcup_{j \neq k} \mathcal{G}_j\right\}$
    Let $\mathcal{U}$ be the first-order discontinuities for $GRF_{\mathcal{G}', \mathcal{C}_k}$
    for u in $\mathcal{U}$ **do**:
      Set $z = z^*\left(\rho\left(x_{\mathcal{B}_k} | H \cup \{x_{\mathcal{C}_k} = u\}\right)\right)$
      if $z \leqslant z^*$:
        pass ← pruning by bound or infeasibility
      else:
        Set $x^{(u)} = x^*\left(\rho\left(x_{\mathcal{B}_k} | H \cup \{x_{\mathcal{C}_k} = u\}\right)\right)$
        $z_u^* = DRFC\text{-}K\left(\mathcal{G}, \mathcal{C}, k+1, z^*, H \cup \left\{x_{\mathcal{B}_k} = x_{\mathcal{B}_k}^{(u)}\right\}\right)$
      end if
    end for
    Set $z^* \in \min_{u \in \mathcal{U}} z_u^*$
    Return: z^*
  end if

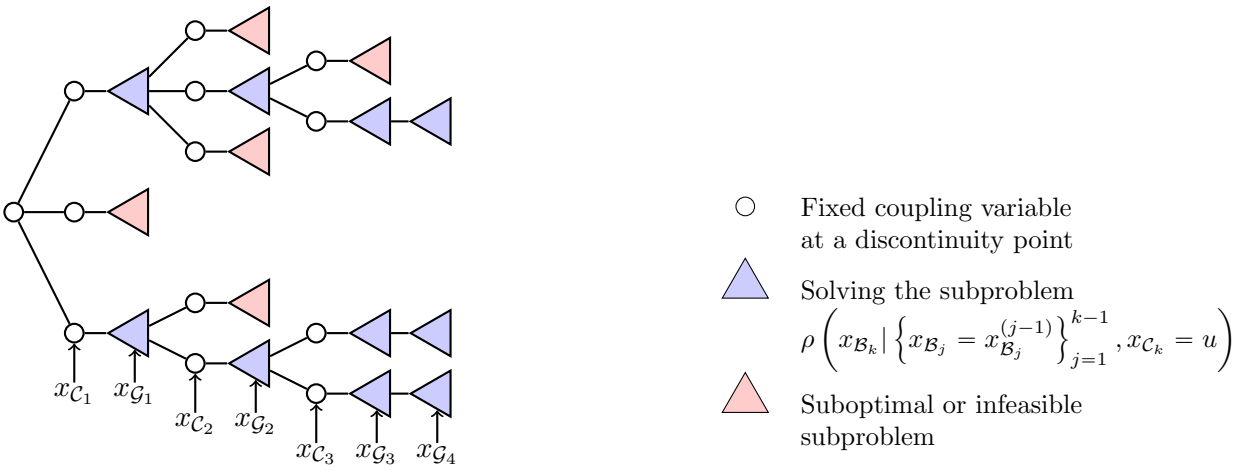


Figure 8.7: Illustration for *DRFC-K* with $K = 4$

8.2.5 Heuristics for decomposition

The *DRF* problem and its adaptations to the different cases presented above rely on a decomposition, and so is their efficiency. Of course, the problem of finding an "optimal" decomposition is not at stake, since the very notion of optimality is subject to debate when considering approximate methods. Indeed, two criteria may be taken into consideration: the number of nodes and the lower bound induced by the decomposition.

In the following, we propose heuristics for defining a decomposition which may fit the structure of problems encountered by EDF.

8.2.5.1 Temporal decomposition

The vast majority of EDF's MILPs carries an important temporal structure. A system, whatever it may be (*e.g.* a microgrid, a nuclear plant, a hydroelectric valley, etc.), is to be optimized over multiple time steps. In this setting, variables are often indexed by time steps and it is really natural to associate a group of variables in a decomposition with a set of time steps.

Formally, we call *temporal decomposition* a decomposition $\mathcal{G} = \{\mathcal{G}_k\}_{k=1}^{K}$ which satisfies

$$\left(x_i \in \mathcal{G}_k,\ x_j \in \mathcal{G}_{k'},\ k < k'\right) \iff t_i < t_j$$

where t_i refers to the time step of variable i.

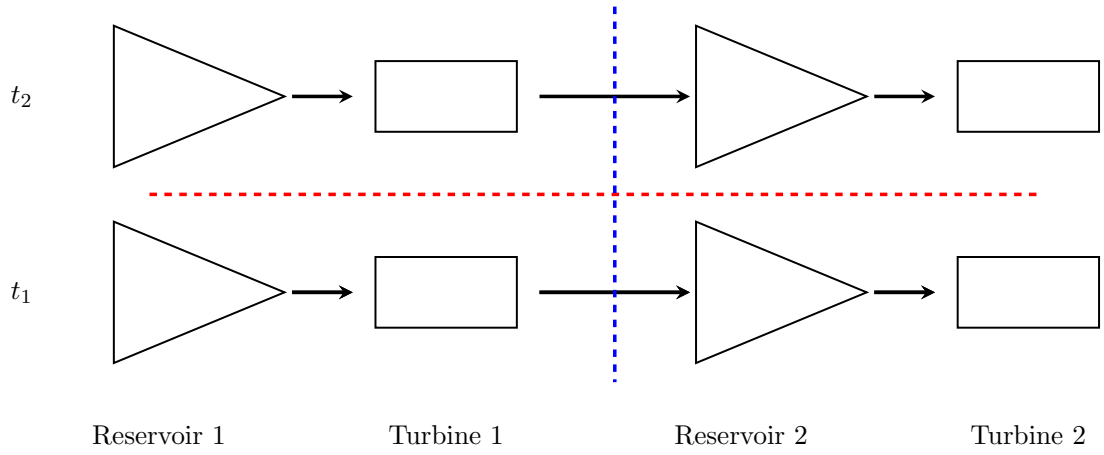


Figure 8.8: Spatial (blue dashed line) vs temporal (red dashed line) decomposition

8.2.5.2 Spatial decomposition

Sometimes, time may not be as structural as space, and using the system's units may be more justified than time steps. Hydroelectric valleys (see Section 3.2) is a sound example of such phenomenon. Consider for simplicity the case where $K = T = 2$ with two power units (see Figure 8.8). If one considers a temporal decomposition, many coupling variables will appear in various locations of the valley due to the temporal inertia of the problem, specified by the water flowing and the unit temporal constraints. In the opposite, if we consider a decomposition using the power units to define blocks, the water flow between units account for the main coupling variables. Of course, the number of such variables will grow as the number of considered time steps increases, and considering such a decomposition when T is extremely high may not be wise.

In other words, time may not be considered as a relevant criterion for decomposition if it does not create a bottleneck, allowing a single (or limited amount of) variable(s) to impact two consecutive blocks, as illustrated in Figure 8.8. Keeping that in mind, an alternative for temporal decomposition is *spatial decomposition*, which is formally a decomposition $\mathcal{G} = \{\mathcal{G}_k\}_{k=1}^{K}$ which satisfies

$$\left(x_i \in \mathcal{G}_k,\ x_j \in \mathcal{G}_{k'},\ k \neq k'\right) \iff e_i \neq e_j$$

where e_i refers to the equipment associated with variable i.

8.2.5.3 Spectral decomposition

When introducing spatial decomposition, we mentioned an intuitive criterion, similar to a "thumb rule", for using either temporal or spatial decomposition. In the following, we take further this idea of selecting the decomposition which induces few and discriminating coupling variables, by casting the decomposition process into a clustering task. In the most general case (*i.e.* $DRFC\text{-}K$), the process to be automated is then (*i*) build a K-partition of the index set by clustering, (*ii*) order heuristically this partition and (*iii*) select the coupling variables of interest. This is the object of Algorithm 16, that we explain in the following.

Partitioning the index set

We can list different attributes that we may want our decomposition to have. First, as said above, the number of coupling variables between groups should be low. Second, the groups should be of similar size to avoid solving large subproblems. Third, we want to select coupling variables which have a strong impact on the consecutive blocks. Such criteria are similar to those considered in the previous chapter. As a consequence, we propose to partition the index set again by Spectral Clustering (see Chapter 7) on a primal graph.

Selecting the coupling variables and ordering the decomposition

Let $\{\mathcal{G}_1, ..., \mathcal{G}_K\}$ be the partition obtained by Spectral Clustering, we still need to extract relevant coupling variables and order the groups. Denoting W_{ij} the weight between variables i and j in the considered primal graph, we define a local score

$$s\left(\mathcal{G}_{k_1}, \mathcal{G}_{k_2}\right) = \sum_{i \in \mathcal{G}_{k_1}} \sum_{j \in \mathcal{G}_{k_2}} W_{i,j}$$

which quantifies the intensity of the links (in the sense of the considered primal graph) between $\mathcal{G}_{k_1}$ and $\mathcal{G}_{k_2}$. We then define a non-oriented complete *aggregated graph* where vertices are associated to the groups $\mathcal{G}_1, \ldots, \mathcal{G}_K$, the weight of an edge between two vertices associated with $\mathcal{G}_{k_1}, \mathcal{G}_{k_2}$ being $s\left(\mathcal{G}_{k_1}, \mathcal{G}_{k_2}\right)$. The ordering is then obtained by finding the maximal hamiltonian chain on this graph, and the selected coupling variable $\mathcal{C}_{k_1}$ between two consecutive groups k_1, k_2 is defined by

$$\mathcal{C}_{k_1} = \underset{i \in \mathcal{G}_{k_1} \setminus \mathcal{B}_{k_1}}{\arg\max} \sum_{j \in \mathcal{G}_{k_2}} W_{ij}$$

Note that the problem of finding a maximal hamiltonian chain is NP-hard. However, it can be solved by complete enumeration as, in practice, K is kept low.

Algorithm 16 Spectral Decomposition

Input:
A number groups K
Procedure:
Build a primal graph of weights $(W_{ij})_{i,j=1}^{n}$
Apply Spectral Clustering to obtain a partition $\{\mathcal{G}_1, ..., \mathcal{G}_K\}$
Build the aggregated graph associated to $\{\mathcal{G}_1, ..., \mathcal{G}_K\}$
Find a maximal hamiltonian chain on the aggregated graph and set $\{r_1, ..., r_K\}$ the indices of its nodes
Set $\mathcal{C}_{r_k} = \arg\max_{i \in \mathcal{G}_{r_k} \setminus \mathcal{B}_{r_{k+1}}} \sum_{j \in \mathcal{G}_{r_{k+1}}} W_{ij}$ for k in $1, \ldots, K-1$
Return:
$\mathcal{G} = \{\mathcal{G}_{r_1}, \ldots, \mathcal{G}_{r_K}\}$
$\mathcal{C} = \{\mathcal{C}_{r_1}, \ldots, \mathcal{C}_{r_K}\}$

Remark 8. Some remarks on the hyperparameter K

In the previous procedure, K is considered as a given hyperparameter and a trade-off actually appears when setting its value. On the one hand, increasing the value of K allows to handle lower subproblems. On the other hand, it also increases the search space $X_{\mathcal{C}}$, which may increase the number of subproblems to solve and/or decrease the lower bound.

8.3 Experiments

To evaluate the DRF methodology, we compare the two variants $DRFB$ and $DRFC$ with RF and `CPLEX`. The three decomposition methods (temporal, spatial and spectral) are tested on `micro_bal_T12`, `micro_bal_T12`, `hydro_var_2`, `hydro_var_3` and `hydro_var_4`, except for the spatial decomposition which is not used on `microgrid` problems, as it makes less sense than for `hydro` problems. In $DRFC$, we select a unique coupling variable and look for first-order discontinuities using a grid of length 200. We heuristically select the coupling variable by maximizing the score

$$\sigma_{u \in [\alpha, \beta]} \left(\rho(.|x_c = u) \right) \tag{8.9}$$

with α_c, β_c the bounds obtained by OBBT for the coupling variable $c \in \mathcal{C}$ and $\sigma_{u \in [\alpha, \beta]} \left(f(u) \right))$ the standard deviation of f values on a uniform grid over $[\alpha, \beta]$. The idea of this heuristic is to select a

coupling variable which has a strong influence on the objective.

Table 8.1 and Table 8.2 present the results obtained on 500 instances of each problem when following either the order proposed in the spectral decomposition or the natural order for temporal and spatial decompositions, *i.e.* solving first the subproblem associated with lower time steps or the one associated with the upstream units. $DRFB$ is only used with the temporal decomposition as the spatial decomposition only contains continuous coupling variables in `hydro` problems. Likewise, spectral decomposition may fall into the same case.

	`micro_bal_T12`	`micro_bal_T24`
`CPLEX`	(336, 0.0, 0.0)	(13629, 0.0 , 0.0)
`RF-temporal`	(71, 0.0, -5.14)	(726, 0.0 , -3.54)
`DRFC-temporal`	(**858**, **0.0**, **-2.78**)	(**4608**, **0.0** , **-1.46**)
`DRFB-temporal`	(**310**, **0.0**, **-0.8**)	(**2678**, **0.0** , **-0.57**)
`RF-spectral`	(80, 0.0, -16.22)	(697, 0.49, -17.78)
`DRFC-spectral`	(928, 0.0, -14.27)	(4872, 0.0 , -15.43)

Table 8.1: Respectively nodes, proportions of unsolved instances and relative optimality gaps on `microgrid` problems. Bold characters point out the best approaches regarding the proportion of unsolved instances, with an optimality gap lower than 5%. $K = 2$.

	`hydro_var_2`	`hydro_var_3`	`hydro_var_4`
`CPLEX`	(1650, 0.0, 0.0)	(28679, 0.0, 0.0)	(46339, 0.0, 0.0)
`RF-temporal`	(100, 5.4, -0.79)	(501, 2.6, -0.62)	(592, 10.6, -0.54)
`DRFC-temporal`	(**792**, **0.0**, **-0.85**)	(3496, 0.2, -0.91)	(**3454**, **0.0**, **-0.84**)
`DRFB-temporal`	(641, 0.8, -0.39)	(4028, 0.4, -0.18)	(18688, 1.0, -0.21)
`RF-spectral`	(141, 5.0, -0.35)	(1303, 44.2, -0.68)	(1158, 2.6, -0.32)
`DRFC-spectral`	(**244**, **0.0**, **-0.63**)	(222215, 11.8, -1.56)	(8216, 0.2, -0.59)
`RF-spatial`	(126, 12.4, -0.41)	(1402, 2.4, -0.12)	(905, 13.6, -0.3)
`DRFC-spatial`	(**844**, **0.0**, **-1.15**)	(**3955**, **0.0**, **-0.77**)	(3809, 0.6, -0.7)

Table 8.2: Respectively nodes, proportions of unsolved instances and relative optimality gaps on `hydro` problems. Bold characters points out the best approaches regarding the proportion of unsolved instances, with an optimality gap lower than 5%. $K = 2$.

We see that both $DRFB$ and $DRFC$ systematically decrease the proportion of unsolved instances and the gap compared to RF. These gains are obtained at the cost of a higher number of processed nodes, which is nonetheless much lower than the nodes processed by `CPLEX` on the majority of the problems considered. The temporal decomposition appears to have relatively more stable performances than its competitors. We note that the spectral decomposition exhibits very poor performances on

`hydro_var_3` when setting $K = 2$.

Generally, $DRFB$ appears to find better solutions that $DRFC$. However, it fails more often to obtain a feasible solution. This can be explained by the fact that $DRFB$ explores the entire set of possibilities for all the binary coupling variables, whereas $DRFC$ only explores a subset of the possible values of a single coupling variable. Thus, the exploration is more thorough in the $DRFB$ case. However, the choices made are non reversible in $DRFB$ as we actually branch on the coupling variables, whereas $DRFC$ fixes their value only temporarily.

Table 8.3 presents the results for $DRFC$-K using the spectral and spatial decompositions, where K is set to the number of units of the considered problem. We see that the effect is ambiguous, as it makes the spectral decomposition more efficient on `hydro_var_3` and conversely on `hydro_var_4`. Otherwise, results are less convincing but still comparable to that of Table 8.2.

	`hydro_var_3`	`hydro_var_4`
`CPLEX`	(28679, 0.0, 0.0)	(46339, 0.0, 0.0)
`RF-spectral`	(297, 2.4, -0.15)	(421, 25.0, -0.6)
`DRFC-spectral`	(**2442**, **0.0**, **-0.51**)	(**33351**, **2.0**, **-1.82**)
`RF-spatial`	(297, 2.4, -0.15)	(220, 24.8, -0.44)
`DRFC-spatial`	(**2290**, **0.0**, **-0.82**)	(7685, 3.6, -1.7)

Table 8.3: Respectively nodes, proportions of unsolved instances and relative optimality gaps on `hydro` problems. Bold characters points out the best approaches regarding the proportion of unsolved instances, with an optimality gap lower than 5%. K is set to the number of units in `hydro` problems.

Table 8.4 and Table 8.5 present the results when considering the same decomposition with a reversed order compared with Table 8.1 and Table 8.2 respectively. We see that it yields lower performances as it allows to find fewer solutions in case of `hydro` problems or higher gaps on `microgrid`, whether we consider RF or DRF approaches. These results confirm that the order matters in such methods, and that the natural order induced by the formulation is more relevant than the opposite.

	micro_bal_T12	micro_bal_T24
CPLEX	(336, 0.0, 0.0)	(13629, 0.0, 0.0)
RF-temporal	(89, 0.0, -9.85)	(2578, 0.0, -14.76)
DRFC-temporal	(861, 0.0, -6.08)	(9144, 0.0, -7.2)
DRFB-temporal	(**276**, **0.0**, **-1.76**)	(**5405**, **0.0**, **-3.0**)
RF-spectral	(88, 0.0, -9.85)	(20143, 0.0, -14.75)
DRFC-spectral	(805, 0.0, -6.03)	(26560, 0.0, -7.2)

Table 8.4: Results when using the reverse order compared to Table 8.1. $K = 2$.

	hydro_var_2	hydro_var_3	hydro_var_4
CPLEX	(1650, 0.0, 0.0)	(28679, 0.0, 0.0)	(46339, 0.0, 0.0)
RF-temporal	(87, 9.2, -1.06)	(428, 34.2, -1.2)	(528, 46.2, -0.67)
DRFC-temporal	(622, 0.4, -0.75)	(4027, 0.8, -2.36)	(**10908**, **1.8**, **-1.03**)
DRFB-temporal	(775, 1.2, -0.53)	(5206, 3.2, -0.5)	(76882, 2.6, -0.31)
RF-spectral	(126, 12.4, -0.41)	(1402, 2.4, -0.12)	(1180, 13.2, -0.15)
DRFC-spectral	(1266, 2.2, -0.43)	(**2539**, **0.0**, **-0.16**)	(8549, 5.6, -0.18)
RF-spatial	(141, 5.0, -0.35)	(1303, 44.2, -0.68)	(612, 49.4, -0.8)
DRFC-spatial	(**599**, **0.0**, **-1.14**)	(29497, 2.2, -2.25)	(34732, 6.6, -0.99)

Table 8.5: Results when using the reverse order compared to Table 8.2. $K = 2$.

Chapter 9

Perturbation of the Objective Function

Content

This chapter presents an attempt to learn an efficient objective function's disruption for a given problem. First, Section 9.1 casts this objective as a black-box optimization problem and presents some observations in that matter. The impact of a disruption on the optimal value is also questioned. Section 9.2 presents the methodology used to solve this problem, especially using auto-encoders to downsize the search space and therefore to handle the curse of dimensionality. Last, Section 9.3 displays some brief experiments.

9.1 Preliminaries

9.1.1 Objective - BBO

One of the identified sources of difficulties for solving MILPs by means of a B&B procedure is the presence of symmetries, e.g. when the problem contains identical variables (same objective coefficients, same constraints). As explained in [107], these symmetries considerably increase the size of the search space. For instance, if a problem exhibits N interchangeable variables, breaking symmetry could reduce the size of the search space by a factor N.
A variable symmetry is a permutation of the variables that preserves the solution's value. In other words, a bijection $\sigma : \{1, \dots, n\} \to \{1, \dots, n\}$ on the n indices of the variables exists such that if for every $i, x_i = d_i$ is a solution, then $x_{\sigma(i)} = d_i$ is a solution as well with the same value [108]. At a high level, one can then see objective disruptions as a way to produce an ordering for B&B nodes with same initial LP values, breaking these symmetries and thus, hopefully, producing smaller B&B trees.

Perturbing the objective function has already been identified as a way to break the symmetries of a problem. However, as pointed out in [108], it turns out to be less efficient than problem-specific methods (see [109] for recent advances on the subject). Besides, a major inconvenient is that one cannot know in advance if the perturbation will reduce the B&B tree or if, on the contrary, it will increase its size. When facing repeated problems coming from an unknown distribution, one may be tempted to learn the optimal perturbation for this distribution.
This task is naturally defined as a Black-Box Optimization (BBO) problem. Let $\Omega \subset \mathbb{R}^n$ be the set of admissible perturbations for a given instance distribution $\mathcal{L}$ (*i.e.* a given problem). Given a

perturbation $\varepsilon \in \Omega$, the perturbed (or disrupted) problem refers in the following to

$$p_\varepsilon : \begin{cases} \min\limits_{x} & (c+\varepsilon)^\top x \\ s.t. & Ax \leqslant b \ ; \ x \in \{0,1\}^{|\mathcal{J}|} \times \mathbb{R}^{n-|\mathcal{J}|} \end{cases} \tag{9.1}$$

One can define a *black-box evaluation function* μ such that

$$\mu \ : \ \begin{cases} \Omega \to \mathbb{R} \\ \varepsilon \mapsto \mu(\varepsilon) = \mathbb{E}_{p\sim\mathcal{L}}\left[f(p,\varepsilon)\right] \end{cases}$$

where $f(p,\varepsilon)$ can be any statistic (e.g. the number of nodes) produced on the instance p by applying a disruption ε on the objective function. Such definition remains valid under any configuration of the B&B solver considered. In this setting, we are looking for the optimal perturbation ε^* which satisfies

$$\varepsilon^* \in \arg\min_{\varepsilon\in\Omega} \mu(\varepsilon). \tag{BBO}$$

for a given distribution $\mathcal{L}$. As the analytic form of μ is obviously not known, ε^* is to be approximated by sampling techniques.

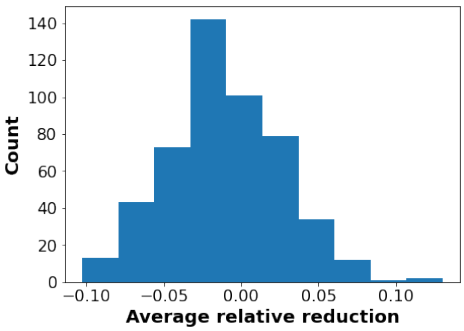
A more ambitious objective would be to generate individual perturbations, *i.e.* one optimal perturbation per instance. This task would then formally be defined as finding an optimal generator $\gamma^* : p \mapsto \gamma^*(p) = \varepsilon \in \Omega$, which satisfies

$$\gamma^* \in \arg\min_{\gamma\in\Gamma} \mathbb{E}_{p\sim\mathcal{D}}\left[f\left(p,\gamma(p)\right)\right] \tag{BBO2}$$

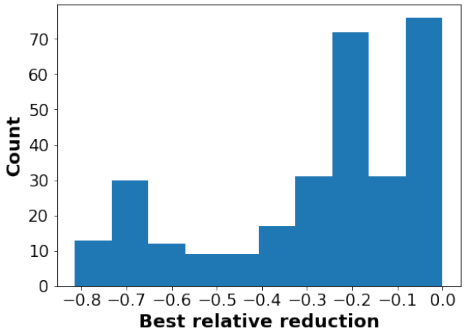
where Γ is some function space mapping the set of instances to Ω.

9.1.2 Observations

Figure 9.1 shows the histogram of the averaged performance over 300 instances of `micro_asym_T6` for $M = 500$ normalized random perturbations, generated from a gaussian distribution. More formally, Figure 9.1 displays the histogram of $\hat{\mu}(\varepsilon_j) \equiv \frac{1}{N}\sum_{i=1}^{N} f(p_i,\varepsilon_j)$ with $f(p,\varepsilon) = \frac{|\mathcal{T}(p,\varepsilon)|-|\mathcal{T}(p,0)|}{|\mathcal{T}(p,0)|}$, $|\mathcal{T}(p,\varepsilon)|$ denoting the size of the B&B tree for the instance p perturbed with ε and N the number of instances in the selected set. This metric will be referred as the *relative performance* in the following.



(a) Averaged relative performance by perturbation.



(b) Relative performance of the best perturbation by instance.

Figure 9.1: Histograms of the relative performance over 500 random perturbations on 300 instances of the `micro_asym_T6` problem. The L^2 norm of the perturbations is 10^{-6}.

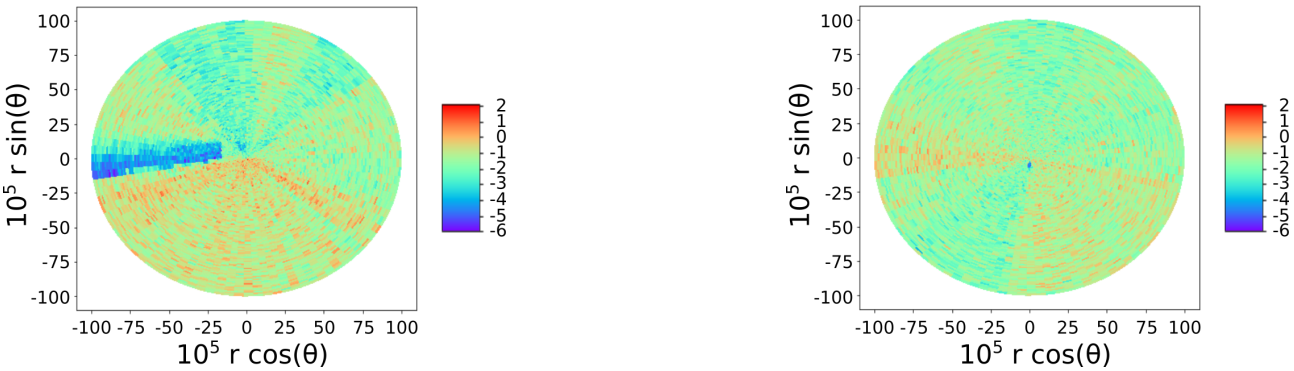
Figure 9.1 lets us think that there is actually some potential in the subject. The black-box optimization program (BBO) is typically addressed by designing a sampling scheme towards the left tail of the histogram in Figure 9.1a, *i.e.* to design a perturbation which performs well in average. As for (BBO), its objective is to reach similar results as those of Figure 9.1b, where the perturbation is dependent on the instance. Although this approach seems more promising, it may be harder to obtain a satisfying perturbation generator.

A major issue is to be tackled before using standard techniques to solve the optimization problem (BBO): the so-called curse of dimensionality. Indeed, BBO relies on sampling procedures in Ω, which size directly depends on the number of variables n. Hence, before applying BBO techniques, it is important to make sure that one could find a low-dimensional space where the evaluation function is smooth.

Let us leverage the problem's structure to visualize the black-box function in a low-dimensional space $\mathcal{S} \subset \Omega$. We take the weight matrix of an influence graph as defined previously (see Chapter 7) and draw perturbations on a η-sphere in the span of the eigenvectors associated with the two highest eigenvalues of this graph. More concretely, let W be the weight matrix of an influence graph and v_1, v_2 the two eigenvectors associated with its two highest eigenvalues. We then consider η-spheres in the span of (v_1, v_2), i.e.

$$\Omega_\eta = \left\{ \varepsilon \in \mathbb{R}^n,\ \varepsilon = \eta \frac{\cos(\theta) v_1 + \sin(\theta) v_2}{||\cos(\theta) v_1 + \sin(\theta) v_2||} \right\}.$$

Figure 9.2 shows the average node reduction of such perturbations using `coun_graph`, for different values of η and compare it with some drawn in a random 2D space. Two observations can be made. First, the performance of a perturbation seems to be relatively independent of its norm, at least in such low-dimensional spaces, which advocates for considering only spherical spaces. Second, the evaluation function μ appears to be relatively smooth in these low-dimensional spaces, hence allowing to encompass BBO techniques.



(a) Projection in the 2VP space

(b) Projection in a 2D random space

Figure 9.2: Average nodes reduction for 2D random perturbations on the `micro_asym_T6` problem

Apart from these observations, it appeared that heterogeneous characteristics influence the performance of a perturbation. For instance, regarding the `microgrid` problem, disrupting only binary variables seems to be more efficient than disrupting all variables or only continuous ones. Besides, applying the same perturbation to every coefficient belonging to the same time step turned out to be as efficient as random perturbations.

9.1.3 Is it legitimate to disrupt the objective function?

Let us restrict the perturbations to the η-sphere in $\mathbb{R}^n$, that is to say $\Omega = \{\varepsilon \in \mathbb{R}^n,\ ||\varepsilon|| = \eta\}$ where $||.||$ stands for the standard Euclidean norm. Such a choice is made as we consider that the tuning of the norm of the perturbation is independent to that of its shape (here, two colinear perturbations are said to have the same shape). In other words, we assume that if we can optimize the perturbation's shape for a given norm, the optimization process still stands for any desired norm. This assumption is based on the previous observations.

Before tackling the question of the tuning of the shape of the perturbation, it is necessary to ensure that one can disrupt the objective function without downgrading the quality of the solution. In case of a pure Binary Linear Problem (BLP), Propositions 9.1.1 and 9.1.2 state that the solution's value is kept unchanged provided that the norm of the perturbation is small enough. As a consequence, the perturbation's shape can be safely tuned as soon as one controls the norm of the perturbation.
Here, the original problem (P) and its perturbed counterpart (P_ε) are written

$$(P)\ \begin{cases} \min\limits_{x} & c^\top x \\ s.t. & Ax \leqslant b,\ x \in \{0,1\}^n \end{cases} \qquad (P_\varepsilon)\ \begin{cases} \min\limits_{x} & (c+\varepsilon)^\top x \\ s.t. & Ax \leqslant b,\ x \in \{0,1\}^n \end{cases} \tag{9.2}$$

Proposition 9.1.1. *For any BLP (P), a positive real number η exists such that for any ε lying on the hypercube $\Omega_\eta = \{\varepsilon \in \mathbb{R}^n,\ ||\varepsilon||_\infty \leqslant \eta\}$, the solution of (P) and that of the disrupted problem (P_ε) have the same objective value in (P).*

Proof. *Let us select an appropriate value for η and show the proposition by contradiction. We write $\Delta = \{-1,0,1\}^n$, $\Delta^+ = \{\delta \in \Delta,\ c^\top \delta > 0\}$ and set η such that*

$$0 < \eta < \min_{\alpha \in \Delta^+} \frac{c^\top \alpha}{n}.$$

Such value trivially exists except for $c = 0$, but in that case the proof is direct as any feasible solution have the same objective value in (P).
Let x^* and x be two optimal solutions of respectively (P) and (P_ε), and assume $x \neq x^*$ so that $x = x^* + \delta$ with $\delta \in \Delta \backslash \{0\}$ and that $c^\top x^* \neq c^\top x$.
As x^* is the solution of (P), we have that $c^\top x^* < c^\top (x^* + \delta)$ since $x^* + \delta$ is a feasible solution for both problems, so $\delta \in \Delta^+$. Besides, as x is the solution of (P_ε), we have

$$\begin{aligned} & (c+\varepsilon)^\top x \leqslant (c+\varepsilon)^\top x^* \\ \Longrightarrow\ & c^\top \delta \leqslant -\varepsilon^\top \delta \leqslant \left|\varepsilon^\top \delta\right| \leqslant |\varepsilon|^\top |\delta| \\ \Longrightarrow\ & c^\top \delta \leqslant \sum_{j=1}^{n} |\varepsilon_j| \leqslant n\eta \end{aligned}$$

which is in contradiction with the definition of η. □

Proposition 9.1.2. *For any BLP (P), a positive real number η exists such that for any ε lying on the ball $\Omega_\eta = \{\varepsilon \in \mathbb{R}^n,\ ||\varepsilon|| \leqslant \eta\}$, the solution of (P) and that of the disrupted problem (P_ε) have the same objective value in (P).*

Proof. *The proof directly stems from Proposition 9.1.1 as a non-empty hypercube always contains a non-empty ball.* □

In the following, Ω_η will refer to the sphere of radius η.

Remark 1. Disrupting the constraints' right-hand side seems less conceivable

Note that disrupting the objective function is less hazardous than disrupting the right-hand side (rhs). Indeed, the latter may discard integer solutions and potentially heavily harm the value of the found solution. Think for example of a constraint $x_i + x_j \leqslant 1$ with $i, j \in \mathcal{J}$: if the disrupted rhs value was lower than 1, the solutions $(x_i = 1, x_j = 0)$ and $(x_i = 0, x_j = 1)$ would be removed from the feasible set. More generally, constraints involving integer variables are often designed to be tight so as to find earlier feasible solutions. This characteristic is hardly compatible with disruptions.

9.1.4 Choice of the statistic f

So far, we have not discussed the choice of the statistic f and only mentioned the absolute performance, *i.e.* the average node reduction. Actually, this natural idea may be a poor choice. Figure 9.3 shows the histogram of the number of nodes for several instances of a given problem. Naturally, trying to solve the problem (BBO) using the absolute performance would lead to overfitting to the training set, as the few difficult instances (right tail) are the one with the highest potential decrease. An alternative is to consider the relative decrease as statistic of interest $f(p, \varepsilon) = \frac{|\mathcal{T}(p,\varepsilon)| - |\mathcal{T}(p,0)|}{|\mathcal{T}(p,0)|}$. It appears to be a more robust measure of performance, as it gives more weight to the easier instances, which are more numerous thus more statistically significant. However, it is important to acknowledge that this measure does not fully reflect the objective pursued in the rest of this document.
Another approach would be to consider the hardest instances as outliers and discard them from the analysis. Again, it does not seem satisfying in our context.

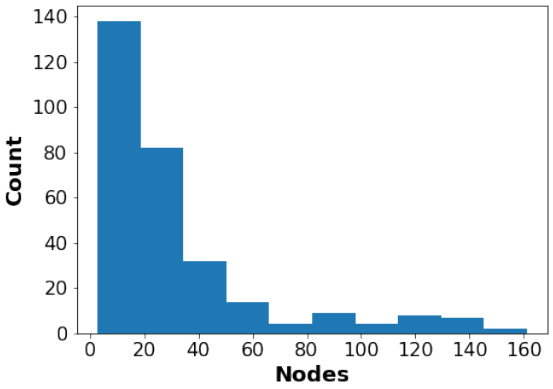


Figure 9.3: Histogram of the number of nodes without perturbation for 500 instances of the `micro_asym_T6` problem.

9.2 Dimension reduction and surrogate models

9.2.1 BBO with dimension reduction

BBO aims at finding a minimum of a black-box function, without any assumption on such function. A plethora of methodologies have been introduced on this matter (see for instance [110, 111, 112]), and we restrict ourselves to those designed to handle the case of expensive functions. In a BBO setting, we call expensive a black-box function which requires many resources (*i.e.* time) to be evaluated. This is the case of the performance as defined above, as it requires to solve as many MILPs as the size of the training set. BBO usually works by sampling and evaluating in an iterative manner new points in the search space. The difficulty of the task is to design an efficient sampling procedure, allowing to hopefully find the optimum in few iterations. In this context, one of the most used strategies is to use surrogate models and merit functions to guide the exploration of the search space. Let us precise these notions.

A surrogate model is a predictor $h : \Omega \to \mathbb{R}$ of the black-box evaluation function μ. This predictor allows to make fast approximations of the black-box function and thus to reduce the number of expensive calls. At each iteration, new points are sampled and evaluated by the black-box function μ. This sampling is usually performed through the use of a merit function $\nu : \Omega \to \mathbb{R}$, which gives a score to any new point using its expected performance based on the surrogate model h. Usually, merit functions define an *exploration-exploitation trade-off*, arbitrating between sampling promising points and exploring new areas of the search space. Note here the parallel with RL, where the same trade-off

appears (see [113] for a parallel between the two approaches).

Algorithm 17 presents a generic iterative procedure for performing BBO in the context of program (BBO).

Algorithm 17 BBO - Generic

Input:
$\mu, \nu, h : \Omega \to \mathbb{R}$
$(\varepsilon_i, \mu(\varepsilon_i))_{i=1}^{n_0}$
Procedure:
for t in $0, \ldots, T-1$ **do:**
Calibrate h using $(\varepsilon_i, \mu(\varepsilon_i))_{i=1}^{n_t}$
Sample N new points $(\varepsilon_i)_{i=n_t+1}^{n_{t+1}=n_t+N}$ in Ω using ν and h
Compute the evaluations of the new sampled points $(\mu(\varepsilon_i))_{i=n_t+1}^{n_{t+1}}$
end for
Return:
The best ε obtained.

BBO techniques suffer from the curse of dimensionality as sampling must be performed in the search space Ω. To overcome this issue, we opted for projecting the perturbations into a low-dimensional and structured space $\mathcal{S}$ through a projection (or encoding) function ϕ

$$\phi \ : \ \begin{cases} \Omega \to \mathcal{S} \\ \\ \varepsilon \mapsto \phi(\varepsilon) = u \end{cases}$$

Assuming that this projection admits a decoding function ϕ^{-}, the sampling is to be performed in the low-dimensional space $\mathcal{S}$ to mitigate the curse of dimensionality. Algorithm 18 presents the procedure we use for performing BBO with dimension reduction.

Algorithm 18 BBO with dimension reduction - Generic

Input:
$\mu, \nu, h : \mathcal{S} \to \mathbb{R}$
$\phi : \Omega \to \mathcal{S}$, $\phi^- : \mathcal{S} \to \Omega$
$(\varepsilon_i, \mu(\varepsilon_i))_{i=1}^{n_0}$
Procedure:
for t in $0, \dots, T-1$ **do:**
Calibrate h using $(\phi(\varepsilon_i), \mu(\varepsilon_i))_{i=1}^{n_t}$
Update ϕ and ϕ^- if needed
Sample N new points $(u_i)_{i=n_t+1}^{n_{t+1}=n_t+N}$ in $\mathcal{S}$ using ν and h and compute the corresponding perturbations $(\varepsilon_i = \phi^-(u_i))_{i=n_t+1}^{n_{t+1}}$
Compute the evaluations of the new sampled points $(\mu(\varepsilon_i))_{i=n_t+1}^{n_{t+1}}$
end for
Return:
The best ε obtained.

In the following, we explore the use of auto-encoders to perform this embedding in a low-dimensional space.

9.2.2 Supervised auto-encoder

First, we implemented a Supervised Auto-Encoder (SAE [114]), which is an auto-encoder with the addition of a supervised loss on the representation layer. As shown in Figure 9.4a, SAE is made of an encoding function ϕ, its decoding counterpart ϕ^- and a predictor in the encoding space h. The two losses governing SAE are the supervised loss (l_s) as well as the classical reconstruction error (l_r)

$$\begin{cases} l_s(h, \phi) = \mathbb{E}_{\varepsilon \sim \rho}\left[||(h \circ \phi)(\varepsilon) - \mu(\varepsilon)||_2^2\right] \\ l_r\left(\phi, \phi^-\right) = \mathbb{E}_{\varepsilon \sim \rho}\left[||\left(\phi \circ \phi^-\right)(\varepsilon) - \varepsilon||_2^2\right] \end{cases} \tag{SAE}$$

with ρ the distribution of the so far sampled perturbations. Thus, the calibration step in Algorithm 18 comes down to fitting the SAE model through the losses in equation (SAE).

One may wonder if the reconstruction loss of (SAE), *i.e.* the Euclidean distance between a perturbation and its reconstruction, is appropriate to our application. Implicitly, it means that it matters to reconstruct the perturbation in each of its coordinates. This makes sense in many applications: for instance in *information retrieval* or *data compression*, the only purpose of the encoding is to account for the variations of the data. This is not the case in our application. As stated in Algorithm 18, the BBO

procedure with Dimension Reduction will require to sample in the encoding space $\mathcal{S}$ some points u with expected high performance and, then, decode these new points to obtain valid perturbations with the same expected performance. As a consequence, we rather care about $||(h \circ \phi \circ \phi^-)(u) - h(u)||_2^2$, as $h \circ \phi$ predicts the performance of the decoded perturbation $\phi^-(u)$. Indeed, as the black-box function may not be smooth in Ω, the distance between perturbations and their images is not so relevant. On the contrary, one wish to ensure that, when sampling points in $\mathcal{S}$, the decoded perturbation performs similarly as is expected for the encoded sample. Nonetheless, such metric contains relatively little information and does not really help to build a rich encoding. Actually, as h will be kept linear, we use the richer metric in the encoding space $||(\phi \circ \phi^-)(u) - u||_2^2$.

Using this metric, we define a new neural network architecture, SAEL (for Supervised Auto-Encoder Looped), specially designed for our BBO task. The architecture is presented in Figure 9.4b and the losses governing the training are

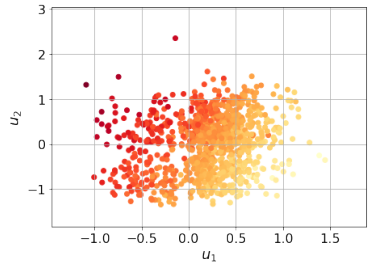
$$\begin{cases} l_s(h, \phi) = \mathbb{E}_{\varepsilon \sim \rho}\left[||(h \circ \phi)(\varepsilon) - \mu(\varepsilon)||_2^2\right] \\ l_r\left(\phi, \phi^-\right) = \mathbb{E}_{\varepsilon \sim \rho}\left[||\left(\phi \circ \phi^- \circ \phi\right)(\varepsilon) - \phi(\varepsilon)||_2^2\right] \end{cases}. \tag{SAEL}$$

(a) SAE (b) SAEL

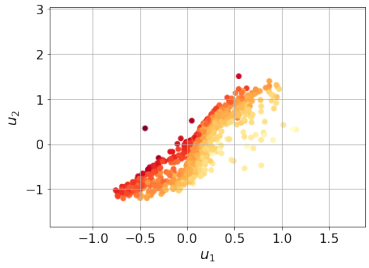
Figure 9.4: Neural Network Architectures. The orange layers are output layers. In SAE, the supervised loss are backpropagated through h and ϕ, while the reconstruction information is sent through ϕ^- and ϕ. In SAEL, the difference lies in the fact that the reconstruction loss is backpropagated through ϕ, ϕ^- and again ϕ.

9.3 Experiments

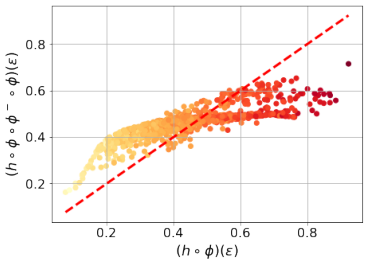
First and foremost, we investigate the coherence of the proposed models. Figure 9.5 and Figure 9.6 show the projections as well as the predicted performance for SAE and SAEL on a set of random perturbations. Figure 9.7 is the equivalent when using the less informative metric $||(h \circ \phi \circ \phi^{-})(u) - h(u)||_2^2$ for the reconstruction loss. The colors represent the actual performance μ of these perturbations.
Figures 9.5a, 9.6a and 9.7a show the projections in a 2D space: $u = (u_1, u_2) = \phi(\varepsilon)$. Thus, we expect to observe a colour gradient as an illustration of the appropriate structure of the embedding space.
In Figures 9.5b, 9.6b and 9.7b, the "looped" projections $(\phi \circ \phi^{-} \circ \phi)(\varepsilon)$ are represented. These projections are expected to be similar to the basic projections $\phi(\varepsilon)$ displayed in the previous figures. Indeed, after sampling in the 2D space $\mathcal{S}$, the actual perturbations are obtained by applying ϕ^{-}. But the only way to ensure that these decoded new points are consistent with their expected performance estimated by h in $\mathcal{S}$ is to project them again in $\mathcal{S}$, as it is the space in which the evaluation is smooth.
Pursuing the same objective, Figures 9.5c, 9.6c and 9.7c display the expected performance of the perturbations $(h \circ \phi)(\varepsilon)$ and their "looped" counterpart $(h \circ \phi \circ \phi^{-} \circ \phi)(\varepsilon)$. We thus expect to see points on the first bisector.
As expected, we can see that SAEL is more coherent than SAE, both in term of encoding reconstruction and predictions' stability. Our hope is then to produce perturbations which have the same value as were expected for their encodings when using SAEL, hence improving the sampling efficiency in Algorithm 18.



(a) Encoded perturbations $\phi(\varepsilon)$ in a 2D space.

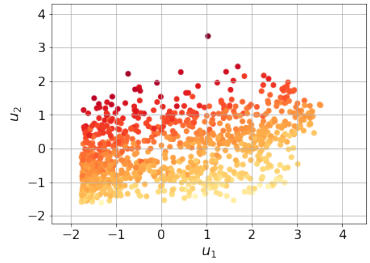


(b) Encoded, decoded and re-encoded perturbations $(\phi \circ \phi^- \circ \phi)(\varepsilon)$ in a 2D space.

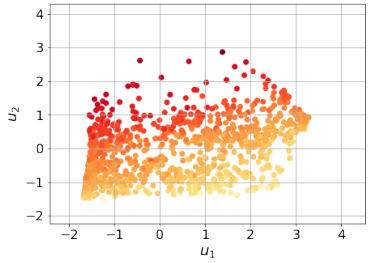


(c) Prediction's robustness to encoding/decoding.

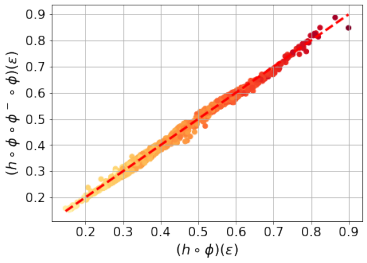
Figure 9.5: Coherence of SAE.



(a) Encoded perturbations $\phi(\varepsilon)$ in a 2D space.

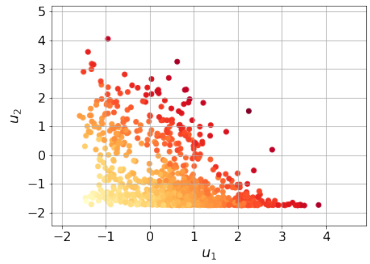


(b) Encoded, decoded and re-encoded perturbations $(\phi \circ \phi^- \circ \phi)(\varepsilon)$ in a 2D space.

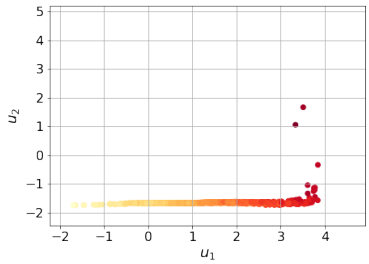


(c) Prediction's robustness to encoding/decoding.

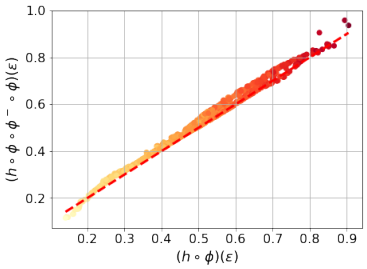
Figure 9.6: Coherence of SAEL.



(a) Encoded perturbations $\phi(\varepsilon)$ in a 2D space.



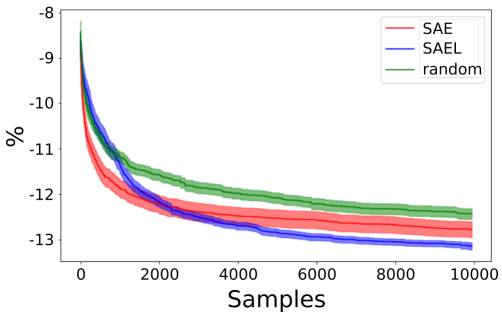
(b) Encoded, decoded and re-encoded perturbations $(\phi \circ \phi^- \circ \phi)(\varepsilon)$ in a 2D space.



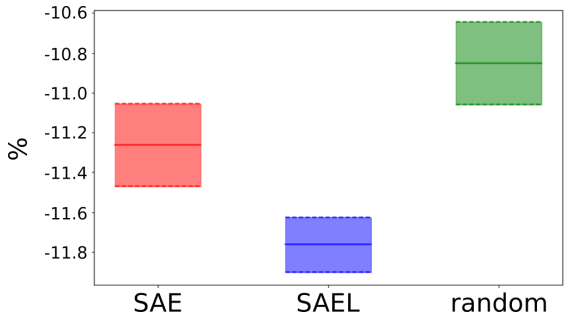
(c) Prediction's robustness to encoding/decoding.

Figure 9.7: Coherence of SAEL with the less informative metric $||(h \circ \phi \circ \phi^-)(u) - h(u)||_2^2$ in the reconstruction loss.

Due to the computational effort of evaluating the black-box function μ, we only ran experiments on `micro_asym_T6`. For sampling in the encoding space, we draw randomly points on a cartesian grid in that space, the probability of sampling a point u on this grid being proportional to the exponential of the expected performance $h(u)$. The metric used is the relative performance, and the results are averaged over 100 independent runs of Algorithm 18 on 200 random training instances and 300 testing sets. Figure 9.8 shows the average performance (with gaussian confidence intervals) on train and test sets. The L^2 norm of perturbations is set to 10^{-6}.



(a) Performances on train instances for the best perturbation found along the training process.



(b) Results on test after 10000 samples.

Figure 9.8: Monte Carlo estimates for BBO relative performance on `micro_asym_T6`.

Results are difficult to analyze. Even if we observe better performances, the little improvement obtained using SAE and SAEL compared with a random exploration questions the utility of the methodology. In addition to the high computational time needed to evaluate the black-box function, this explains why we did not investigate further on this approach and focused more on more promising ones, such as reinforcement learning.

These poor performances may come from many sources, such as the sampling method, the dimension reduction technique, or again the fact that we tested the approach on an easy-to-solve problem.

Chapter 10

Conclusion and perspectives

Although we principally investigated the reinforcement learning (RL) approach, this work has also been an opportunity to consider different paradigms, such as imitation learning, clustering and black box optimization. For each of these paradigms, the integration in the combinatorial optimization framework, and more specifically in the Branch and Bound algorithm (B&B), has faced limitations. Some of them has been lifted, but others remain. We take here a critical look at the different contributions displayed in this document and elaborate on various perspectives for future work.

A preliminary remark

Perhaps the first observation to be made is that our committed position was to omit an important characteristic of the industrial context related to the solving of repeated instances. In practice, a common scenario is to face overlapping instances. For instance, one may imagine the case where a system is optimized at time t for the period going from t to $t+h$, and again at time $t+k$ for the period from $t+k$ to $t+k+h$ with $0 < k < h$. In this setting, it may seem inefficient to conceal the result of the first optimization and start the second one from scratch. For instance, one idea may be to search for solutions of the second instance in some neighbourhood of the solution(s) found during the first optimization. This approach, known as *reoptimization*, may be combined with machine learning, for instance to guide the neighbourhood search.

Reinforcement learning for branching

As presented in this document, we rather chose to consider instances as independent outcomes of a single random variable. In this setting, we developed a reinforcement learning methodology

to discover strategies by trials-and-errors on training instances. To this end, the learning task of discovering efficient B&B strategies have been cast into a Markov Decision Process (MDP). Focus is put on discovering oracle strategies, *i.e.* strategies which minimize the B&B tree size. In this case, learning sequential policies is generally not trivial, and the success of reinforcement learning methods often lies in an informative cost (or reward) signal and a meaningful state representation. In the context of learning the branching strategy for tree size minimization, meeting these two criteria is not a straightforward task.

Regarding the first point, different solutions have been tested. We first introduced the notion of h-ahead branching heuristics and tree-based transitions. Under tree-based transitions, we presented our generic approach as a way of learning in a tractable way these new heuristics. The choice of the cost model is often crucial in RL, and the focus is frequently put on designing heuristic intermediary costs, which allow to limit the credit assignment problem by observing as much as possible cost signals. However, we experimentally showed that costs based on classic branching heuristic scores are not suitable for our task of learning oracle strategies. This observation highlights that classic branching heuristics are based on scores which may reflect the lack of mathematical understanding of the branching dynamics. We proposed an alternative to such heuristic cost models and proved that solving the MDP with a unitary cost model yields an oracle strategy (*i.e.* a strategy which minimizes the B&B tree size). This property is always valid under classic trajectory-based transitions but requires a *depth-first search* (DFS) node selection strategy to hold under tree-based transitions. We experimentally observed that this necessary condition is not restrictive in practice, and that using a non-DFS node selection strategy does not harm the performances. In this setting, tree-based transitions make the value of a state equals to the size of the subtree rooted in the corresponding node. We observed better results under this setting, due to more informative cost signals which alleviate the credit assignment problem inherent to classic trajectory-based transitions. Although solving the MDP with a unitary cost model ensures to minimize the tree size, the value function exhibits a high volatility which makes the learning task difficult. To overcome this issue, we considered a generalization of this cost model which allows to stabilize the targets, and consequently improve the performances of the agent.

As for the second point, we leveraged features presented in prior related works and proposed additional features to build a meaningful state representation. However, when comparing a generic agent designed

to perform well on the whole problem's distribution with the idealized case of specialized agents trained on a single instance, we see that the learnt generic strategy has difficulties to adapt to the specifics of each instance. This observation seems to advocate for a better and unified representation of a state in the specified setting. To obtain such unified representation, we tried to embed the problem's structure in the neural network architecture through constraint-wise convolutions. However, the experiments did not met the expectations associated with this representation. Although using a graph representation seems appealing, for instance by embedding this graph in a neural network (see [68, 69]), our experiments in that matter have not been conclusive so far. The idea, simple to express but more difficult to achieve in practice, is to provide the agent with the means to leverage the structure of the considered problem, both in terms of constraints and optimal locations in the feasible set. Through all the experiments performed, we feel that the representation of a state in the considered MDPs is something that merits to be improved in the future.

Widening the scope to node selection

The reinforcement learning methodology proposed for discovering a branching strategy has been adapted to the case of the node selection strategy and the complete strategy under DFS – both branching and node selection. Under some assumptions, we proved the oracle property of an intuitive node selection strategy, which can directly be learnt by imitation. To improve the sample efficiency of the reinforcement learning approach, we also leveraged demonstrations from this oracle strategy to give additional feedbacks on non-taken actions so as to augment the gradient provided to the agent. The results obtained when learning only the node selection strategy appear to be better than those previously observed on the branching strategy. This is due to the fact that not only the learning task is easier (both the action and state spaces are reduced), but bad decisions are also less penalizing, the node selection strategy being generally less important than branching.

The combination of the two strategies generates open questions, such as coordination. For instance, we noticed that the agent was able to produce better primal integral scores when focusing only on node selection than when learning the complete strategy. This observation suggests that the synergy between the two policies may be improved.

Some perspectives on the reinforcement learning training process

Regarding the use of reinforcement learning for discovering B&B strategies, we already mentioned

the need for improving the state representation and the potential coordination between multiple agents if any. Globally, we observe that our approach struggles to scale with the size of the problem. In that matter, one could think of different ideas to improve its scalability.

First, one could try to improve the exploration. In this work, we encompassed the use of different experts in the exploration and/or learning phase. Another approach which has been considered is to use sampling methods such as MCTS (Monte Carlo Tree Search [97]) to improve the quality of the exploration. However, the experiments performed in this direction were not conclusive and may deserve a more in-depth study. One of the main challenge faced when considering sampling techniques in this context is the dramatically huge size of the search space.

Rather than improving its quality, an orthogonal idea is to augment the amount of resources dedicated to the exploration. We attempted to use asynchronous methods, where parallel agents explore different instances simultaneously, but the learning process appeared to be destabilized by this approach. So far, we do not have insights on the causes of this phenomenon. However, this question of scaling up the samples collection for learning deserves to be looked into much deeper, as the greatest successes of reinforcement learning often involve a massive amount of data.

From a different perspective, it is tempting to use the strong temporal structure of the considered problems to improve the scalability of our methodology. For instance, one may learn strategies on low dimensional problems and marginally adapt them to bigger ones, using some kind of transfer learning [115]. This subject has not be treated in this work but certainly merits to be looked into.

With a similar idea of starting the learning by easy tasks, one could focus first on learning strategies at the bottom of the trees by sampling branching constraints. The number of sampled constraints would decrease through the learning process so as to gradually consider larger trees. This approach may seem natural, just as beginners first practice checkmates and endgames when learning to play chess. However, different issues may appear in our setting when considering this kind of approach. First, the sampling of the branching constraints is crucial, as the agent should learn good strategies for states which are likely to be visited by its future versions. Besides, the scales of the target, strongly related to the tree size in our work, will probably get wider as bigger trees are considered, which may cause some troubles in the training process. This kind of issue was encountered for instance when using expert's demonstrations when learning the branching strategy.

Last, it appears naturally more complex to learn strategies at a problem level than at an instance level.

To obtain more specialized strategies, one may consider to clusterize the instances of a problem so as to train one agent per cluster. Of course, such idea would suffer from the classic pitfall of clustering, which is the use of a heuristic metric to guide the clustering. In a similar way, the training process could be focusing on a restricted number of chosen instances to decrease the variance in the samples, then progressively enlarge the scope to the rest of the training instances.

Considering more ambitious strategies

During this thesis, the focus has principally been put on the branching strategy, as it is often recognized as one of the most important strategies in B&B algorithms. It is legitimate to wonder whether there is some potential in widening the scope by considering generalized branching decisions. In our work, we designed branching policies which split some set S into the two disjunctive sets $\{x \in S;\ x_j = 0\}$ and $\{x \in S;\ x_j = 1\}$ when branching on variable j. When considering generalized branching strategies, child nodes can be created by *general disjunctions* of the form $\left\{x \in S;\ \sum_{j \in \mathcal{J}} a_j x_j \leqslant b\right\}$ and $\left\{x \in S;\ \sum_{j \in \mathcal{J}} a_j x_j > b\right\}$ (see [116]). Such method has the potential for drastically reducing the size of the tree, but would also heavily increase the search space in our reinforcement learning approach. Also, errors may lead to serious increases of tree sizes. To restrict the search, one may perhaps focus on partial assignments, guided by a predictor of optimal solutions.

Graph branching

Using a different lens, we presented a branching heuristic based on a graph representation of a MILP, integrating the notion of variable influences. Trivially, the limitation of such approach is its heuristic nature, not only regarding the definition of influence but also the influence maximization scheme adopted. However, its capacity to drastically reduce the size of B&B trees for difficult instances is interesting, and suggests that graphs are relevant tools for encoding the structure of a problem. Note however that among the various definitions of influence experimented, there is no clear hierarchy regarding the performances. Therefore, different types of graphs may be considered to encode this structure, other than primal graphs – the results in [68] seem to advocate for using a bipartite graph. To overcome the fact that using this branching heuristic may be detrimental according to the considered instance, an option would be to learn when to use it, for instance by training a classifier (similarly to the work of [86], where the authors train a classifier to decide whether or not a decomposition should be used). This learning could be performed at the instance level or, more ambitiously, at the node

level. More generally, one could consider using a pool of branching heuristics as the action space in the reinforcement learning methods presented in this work. In a spirit similar to the approach introduced in [74], it would allow to drastically reduce the size of the action space and thus the exploration cost. In addition, experiments showed that one can drastically reduce the tree size only by taking control of branching decisions at the beginning of the tree. To go further in the parallel with reinforcement learning, one may use the methodologies presented in Part II to learn only decisions near the root node. On the one hand, it may heavily reduce the search space and allow to consider larger problems. On the other hand, it relies on the efficiency of the solver for any other decisions.

Decomposition-coordination by decoupling

To decrease the computational effort necessary to solve large and difficult instances, we presented a decomposition-coordination approach using the problem's structure. Doing so, we lose the optimality guarantee provided by B&B. This method relies on different heuristic choices, such as the selection of the coupling variables for which different values are to be explored. However, the choice of these variables is crucial for the method to find a good solution if any. One may think of learning this selection, but this task seems difficult for two reasons. First, doubts may be raised regarding the existence of a smooth mapping (or at least suitable for learning), between instances and relevant coupling variables. Second, two objectives have to be considered (the number of nodes and the quality of the found solution), the trade-off between the two being arbitrary.

Disrupting the objective function

Last, we addressed the problem of disrupting the objective function so as to decrease the B&B tree size. This approach can be thought as exploiting the structure of the problem, such as symmetries, to provide the B&B algorithm with an ordering for nodes with initially similar LP values. The focus has been put on the discovery of an efficient perturbation for a given problem, which produced mitigated results. We saw that the alternative of learning a generator so as to obtain an individual perturbation for each instance of a same problem is much more ambitious. This axis may call for additional research.

A last word

Through all the discussions and reflections carried out during this thesis, we touched on the plethora of possible ways to leverage machine learning in a combinatorial optimization context. This emerging

field of research has been actively developed over the past three years, and the scientific community is now fully acknowledging its potential. Dedicated teachings, competitions, and even programming modules [117] have recently been created, which reflects the enthusiasm around these questions. All these initiatives promise a durable interest in the topic, with undoubtedly answers provided to the questions raised by our work and, hopefully, new questions to come.

Appendix

Résumé substantiel

Cette thèse a pour but d'utiliser des techniques d'apprentissage automatique pour la résolution de problèmes d'optimisation combinatoire. De par sa position de premier producteur français d'électricité, Electricité de France (EDF) doit continuellement piloter différents sites de production, ce qui se traduit mathématiquement par la résolution de problèmes linéaires en nombres entiers (en anglais *Mixed Integer Linear Programming problems*). A cet égard, EDF doit régulièrement résoudre des instances issues de ces problèmes, définies par des données stochastiques. Actuellement, ces instances sont résolues par un algorithme de Branch and Bound (B&B), sans tirer profit des potentielles similarités entre l'instance courante et celles résolues par le passé. Afin de conserver la garantie d'optimalité fournie par l'algorithme de B&B, nous nous proposons d'exploiter ces similarités pour un problème donné et d'apprendre différentes stratégies au sein de cet algorithme, comme par exemple la stratégie de branchement (sélection de variable) ou de sélection de nœud. Le principal critère utilisé afin d'évaluer la performance des stratégies proposées est la taille de l'arbre de B&B généré.

L'approche majoritairement développée dans ce travail est l'utilisation d'apprentissage par renforcement pour découvrir de telles stratégies par essais/erreurs sur les instances historiques. Afin de s'adapter à l'environnement induit par l'algorithme de B&B, nous définissons un nouveau type de transitions au sein de processus de décision markoviens (en anglais *Markov Decision Processes*, MDPs), basées sur la structure d'arbre binaire. Par ailleurs, nous étudions différents modèles de coût au sein de ces MDPs. Du point de vue de la minimisation de la taille des arbres de B&B, nous prouvons l'optimalité du modèle de coût unitaire sous le modèle de transition classique ainsi que sous le modèle de transition binaire, dans l'apprentissage non seulement de la stratégie de branchement mais également de la stratégie de sélection de nœud. Pour autant, les expérimentations menées pour la stratégie de branchement suggèrent qu'il peut être préférable d'incorporer un biais dans le modèle de coût afin d'améliorer la stabilité du processus d'apprentissage. En ce qui concerne l'apprentissage de la stratégie

de sélection de nœud, nous démontrons l'optimalité d'une stratégie explicitement définie, qui peut être apprise plus efficacement de manière supervisée.
En plus des approches mentionnées, nous proposons une stratégie de décomposition-coordination afin de potentiellement permettre le passage à l'échelle de l'apprentissage par renforcement sur des problèmes de plus grande dimension. Une heuristique de branchement basée sur une représentation par graphe d'un nœud de l'arbre de B&B est également proposée. Cette représentation peut par ailleurs être utilisée afin de guider automatiquement la décomposition précédemment mentionnée. Enfin, nous présentons une approche dédiée à l'apprentissage de perturbations de la fonction objectif, afin notamment de briser d'éventuelles sources de symétrie.
Les différentes méthodes proposées sont évaluées sur des problèmes réels, fournis par EDF. Pour chaque problème, deux configurations sont envisagées afin de renforcer la robustesse des résultats fournis.

Les pages suivantes fournissent un résumé détaillé du contenu du présent manuscrit. La formalisation est restreinte afin de permettre un accès simple aux idées présentées.

Partie 1 : Introduction

Chapitre 1 : Introduction générale

Le contexte de cette thèse CIFRE est donné par Electricité de France (EDF), qui doit régulièrement optimiser des systèmes, qu'ils soient de production, de transport, ou plus généralement d'allocation de ressources. Un même système doit donc être optimisé de manière répétée, ce qui produit des problèmes d'optimisation similaires en termes à la fois de structure et de dimensionnement. Pour un système donné, les multiples instances diffèrent du fait de la variabilité des données qui les définissent. A titre d'exemple, EDF peut être intéressée par l'optimisation journalière d'une centrale électrique. Tous les jours, un plan de production pour cette centrale doit être défini. La réalité matérielle de la centrale est supposée permanente, et il en est donc de même pour la structure du problème d'optimisation sous-jacent, le nombre et type de variables ou de contraintes. Pour autant, le contexte dans lequel le système doit être optimisé peut différer selon le jour considéré, par exemple du fait de l'évolution de la demande, des prix, des coûts, etc. Ces données variables induisent donc des instances différentes

d'un même problème, associé à un système physique permanent.

Concrètement, nous considérons des problèmes d'optimisation linéaire mixte en nombres entiers (en anglais *Mixed Integer Linear Programming problems*, MILPs), écrits

$$p : \begin{cases} \min\limits_{x} & c^\top x \\ s.c. & Ax \leqslant b \ ; \ x \in \{0,1\}^{|\mathcal{J}|} \times \mathbb{R}^{n-|\mathcal{J}|} \end{cases}$$

avec $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, m le nombre de contraintes, n le nombre de variables et $\mathcal{J}$ l'ensemble des indices des variables binaires.

Ces problèmes sont communément résolus à l'aide d'un algorithme de Branch and Bound (B&B), basé sur la connaissance d'algorithmes efficaces permettant de résoudre la relaxation linéaire de p. Un algorithme de B&B consiste en l'expansion d'un arbre, où les nœuds sont associés à des sous-problèmes relâchés de l'instance initiale p. A chaque itération de l'algorithme, la politique de sélection de nœud choisit un nœud parmi ceux n'ayant pas encore été visités. Si le sous-problème associé est infaisable ou peut être démontré sous-optimal, le nœud est fermé et un nouveau nœud doit être exploré. Dans le cas contraire, la politique de branchement sélectionne une variable binaire $j \in \mathcal{J}$ et crée deux nœuds enfants en rajoutant respectivement au sous-problème courant les contraintes $x_j = 0$ ou $x_j = 1$. L'algorithme, illustré par la Figure ci-dessous, se termine en garantissant l'optimalité de la solution trouvée lorsque tous les nœuds ouverts ont été visités.

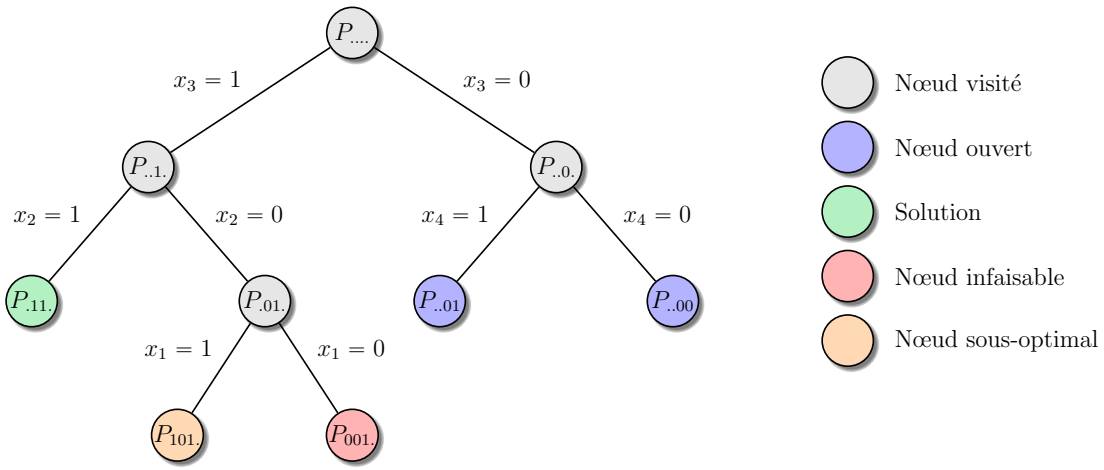


Figure: Illustration d'un arbre de B&B en cours d'expansion avec $\{1, \ldots, 4\} \subseteq \mathcal{J}$. La stratégie de sélection de nœud doit choisir un nœud à visiter parmi les deux nœuds ouverts. Si ce nœud ne correspond pas à un problème infaisable ou ne peut pas être démontré sous-optimal, deux nœuds enfants seront créés par la politique de branchement.

Actuellement, la création de l'arbre de B&B par des solveurs commerciaux est guidée par de nombreuses stratégies heuristiques. Pour prendre en compte les similarités des différentes instances d'un même problème, nous nous intéressons dans ce travail à la possibilité de créer de nouvelles stratégies de B&B, efficaces pour le problème considéré. Le critère retenu pour juger de l'efficacité d'une stratégie est la taille de l'arbre généré. C'est en effet un proxy classique du temps de calcul, communément utilisé car indépendant de l'implémentation et plus généralement de toute considération informatique. En considérant les instances d'un même problème comme les réalisations d'une variable aléatoire suivant une loi inconnue $\mathcal{L}$, le problème que nous chercherons à résoudre peut se formuler comme un problème d'optimisation *boîte noire* où la fonction à minimiser est

$$\mu_{\mathcal{P}} : \begin{cases} \Pi \to \mathbb{R} \\ \pi \mapsto \mathbb{E}_{p \sim \mathcal{L}} \left[\mu \left(p, \pi \right) \right] \end{cases}$$

avec Π l'ensemble des stratégies de B&B d'intérêt et $\mu\left(p, \pi\right)$ la taille de l'arbre de B&B généré sur l'instance p par la politique π.

Le reste du manuscrit est construit de la manière suivante. Les Chapitres 2 et 3 introduisent les notions nécessaires pour la compréhension du document ainsi que les cas d'usage utilisés pour les expérimentations. La Partie 2, composée des Chapitres 4, 5, et 6 présente une approche basée sur

l'apprentissage par renforcement pour la découverte de stratégies oracles, c'est-à-dire minimisant la taille des arbres générés. La Partie 3, quant à elle, explore différentes méthodes permettant d'exploiter la structure des problèmes considérés et ainsi de réduire la taille des arbres de B&B.

Chapitre 2 : Background

Ce chapitre rappelle les principales notions nécessaires à la compréhension des contributions scientifiques du manuscrit, issues à la fois de l'apprentissage statistique et de l'optimisation combinatoire. Il donne également un aperçu de la littérature concernée par l'apprentissage de stratégies de B&B.

La principale approche étudiée dans la littérature consiste à apprendre des stratégies heuristiques par imitation [63, 65, 68, 70]. L'apprentissage par imitation revient à observer les actions produites par un expert en certains états de l'environnement considéré, et d'apprendre à les copier, par exemple en entraînant un classifieur. La limite intrinsèque de ces approches est donc naturellement qu'elles n'ont pas vocation à produire des arbres de taille inférieure à ceux générés par l'expert. Par conséquent, nous nous tournerons plus volontiers vers de l'apprentissage par renforcement [21] (en anglais *Reinforcement Learning*, RL), afin de s'affranchir des stratégies heuristiques existantes et, si possible, de produire des politiques plus efficaces.

Le lecteur est renvoyé au manuscrit pour une introduction plus complète de l'apprentissage par renforcement. La différence principale avec l'apprentissage par imitation réside dans le fait que l'on n'observe pas les actions prises par un quelconque expert. A l'inverse, un modèle de coût est défini et l'objectif de l'apprentissage est alors de trouver une politique optimale, qui permet de minimiser les coûts associés aux actions prises dans les différents états visités. On cherchera alors à explorer l'environnement, à collecter les différents états visités, actions prises et coûts associés afin d'apprendre par essais/erreurs une politique visant à minimiser ces derniers.
Formalisons tout cela au moyen de l'exemple classique de l'algorithme DQN [28], basé sur l'apprentissage de la fonction Q-valeur. Considérons un Processus de Décision Markovien (en anglais *Markov Decision Process*, MDP) $< \mathcal{S}, \mathcal{A}, T, c, \gamma >$ avec $\mathcal{S}$ l'espace d'états, $\mathcal{A}$ l'espace d'actions, T un modèle de transition, c un modèle de coût et $\gamma \in [0, 1]$ un facteur d'escompte. Une *fonction valeur* V^π associée

à une politique π est définie comme l'espérance de la somme escomptée des coûts futurs que l'agent collectera en suivant la politique π depuis l'état courant s_t :

$$V^{\pi}(s_t) = \mathbb{E}_{\Delta^{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k C_{t+k+1} | s_t\right]$$

avec Δ^{π} la distribution des états visités en suivant π et C_t le coût observé au temps t. De manière similaire, la *fonction Q-valeur* est définie comme

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\Delta^{\pi}}\left[\sum_{k=0}^{\infty} \gamma^k C_{t+k+1} | s_t, a_t\right]$$

Avec ce cadre, nous pouvons formellement définir l'objectif de l'apprentissage par renforcement, qui est de trouver une *politique optimale* π^*, *i.e.* une politique qui minimise la valeur de tout état:

$$V^*(s) = \min_{\pi \in \Pi} V^{\pi}(s)$$

Pour atteindre cet objectif, l'algorithme DQN approxime la fonction Q-valeur optimale par un réseau de neurones $\hat{Q}(.,.;\theta)$ et met à jour les poids θ de ce réseau en minimisant à chaque itération i l'équivalent empirique de la fonction de perte

$$L_i^{DQN}(\theta_i) = \mathbb{E}_{(s,a,c,s') \sim \Delta^i}\left[\left(c + \gamma\Big[\min_{a'} \hat{Q}(s', a'; \theta_i^-)\Big] - \hat{Q}(s, a; \theta_i)\right)^2\right]$$

avec c le coût observé depuis la paire état/action (s, a), θ_i^- une version fixée des poids mis à jour périodiquement et Δ^i la distribution de probabilité des expériences collectées dans un buffer au cours des précédentes itérations.

Chapitre 3 : Cas d'usage

Les méthodes proposées sont évaluées sur deux problèmes fournis par EDF. Afin d'augmenter la robustesse des expérimentations, différentes configurations sont envisagées pour chacun des deux problèmes.

Le premier problème, appelé `microgrid`, correspond à l'optimisation d'un réseau de chaleur et d'électricité. L'objectif est de maximiser le profit sur le marché de l'électricité tout en satisfaisant une demande en chaleur variable sur un horizon donné. Les moyens de production sont deux chaudières à

gaz et une unité de cogénération, produisant simultanément de la chaleur et de l'électricité. La chaleur produite par les trois unités peut être stockée avec perte, et la cogénération ne peut être allumée que durant un certain temps.
La dimension du problème `microgrid` est définie par le nombre de pas de temps considérés. Par ailleurs, deux configurations sont envisagées, `micro_asym` et `micro_bal`, les chaudières de la première étant plus dissemblables que celles de la seconde. Ainsi, le degré de symétries est plus élevé dans `micro_bal`.
Pour chaque configuration, les données variables sont les prix du gaz, de l'électricité et la demande en chaleur à chaque pas de temps.

Le second problème considéré, `hydro`, retranscrit la gestion d'une vallée hydraulique. L'objectif est de maximiser le profit sur différents marchés de l'électricité sur un horizon donné. L'énergie est produite par différentes unités de production, composées de turbines, à partir du flot descendant et contrôlé par des réservoirs.
La dimension du problème `hydro` est définie par le nombre d'unités de production considérées. Les deux configurations envisagées, `hydro_fix` et `hydro_var`, diffèrent par leurs politiques de gestion des réservoirs. Contrairement à `hydro_fix`, la politique de gestion du volume des réservoirs à mi-période ainsi qu'en fin d'horizon est variable dans la configuration `hydro_var`. Par ailleurs, le nombre de turbines dans chaque unité de production diffère entre les deux configurations.
Les données variables sont alors les différents prix de l'électricité et, pour la seconde configuration, les contraintes de volume à mi-période et en fin de période.

Partie 2 : Apprentissage de stratégies oracles dans un algorithme de B&B

Dans cette partie, nous proposons d'utiliser l'apprentissage par renforcement pour découvrir des stratégies de branchement et de sélection de nœud efficaces au sein d'un algorithme de B&B. L'accent est mis sur la recherche de stratégies oracles, c'est-à-dire qui minimisent la taille des arbres générés. Nous considérons un algorithme de B&B théorique où seules les stratégies de branchement et de sélection de nœud gouvernent l'expansion de l'arbre (Algorithme 4). Les expérimentations sont effectuées

en utilisant `CPLEX`, et les heuristiques de presolve et de génération de coupes sont désactivées pour permettre une comparaison équitable des stratégies étudiées.

Chapitre 4 : Apprentissage d'une stratégie de branchement dynamique

Ce chapitre vise à définir une méthodologie permettant la découverte d'une stratégie de branchement oracle. Pour ce faire, la dynamique inhérente à une telle stratégie doit être prise en compte. En effet, il est aisé de comprendre que l'efficacité d'une décision de branchement au nœud courant dépend des choix effectués dans le reste de l'arbre de B&B.

Processus de décision markovien – Dans un premier temps, il convient de définir un cadre permettant l'apprentissage d'une telle stratégie. Pour ce faire, nous définissons un processus de décision markovien (MDP) $< \mathcal{S}, \mathcal{A}, T, c, \gamma >$ régissant les interactions entre la stratégie de branchement et l'algorithme de B&B (environnement). L'espace d'états $\mathcal{S}$ est défini comme l'ensemble des possibles observations effectuées au moment de prendre une décision de branchement. Les actions $\mathcal{A}$ sont naturellement associées à la sélection de variables binaires ($\mathcal{A} \equiv \mathcal{J}$), candidates pour le branchement. Les transitions T sont supposées déterministes (Hypothèse 4.1.1) et peuvent être de deux ordres, *trajectory-based* (Définition 4.1.1) comme classiquement en RL ou *tree-based* (Définition 4.1.2). Les transitions *tree-based* sont ici proposées afin de mieux prendre en compte la structure d'arbre binaire de l'environnement, peu commune en RL. Le modèle de coût c reste à définir à ce stade, et le facteur d'escompte γ est un hyperparamètre à optimiser.

Heuristique de branchement à horizon h ($h-$*ahead branching heuristic*) – En utilisant les transitions *tree-based*, nous définissons une classe de stratégies de branchement basées sur une connaissance du futur, c'est-à-dire des nœuds enfants créés consécutivement à une décision de branchement. Notamment, cette classe contient et généralise des stratégies proposées dans la littérature, comme par exemple la stratégie de Strong Branching [51]. L'intérêt de ces stratégies est justifié par le caractère dynamique de toute stratégie de branchement. Pour autant, mettre en place une heuristique de branchement à horizon $h > 1$ n'est pas réaliste, et il en est de même pour l'apprentissage supervisé hors ligne de telles stratégies. Dès lors, une approche par renforcement est privilégiée.

Apprentissage par renforcement avec transitions *tree-based* – Nous développons et étudions une méthodologie de renforcement sous le modèle de transition *tree-based* déterministe proposé plus tôt. Nous montrons notamment que l'on ne peut obtenir une politique optimale à partir de la fonction valeur optimale, contrairement au cadre classique de l'apprentissage par renforcement (Proposition 4.1.1). Pour autant, il est possible de redéfinir un algorithme d'itération sur les valeurs (en anglais *value iteration*) afin de trouver une fonction valeur approchée $V^{\sim}$ (Théorème 4.1.1), solution de l'équation de programmation dynamique

$$V^{\sim}(s) = \min_{j \in \mathcal{J}} c\left(s, j\right) + \gamma\Big[V^{\sim}\left(D_0^{\sim}(s, j)\right) + V^{\sim}\left(D_1^{\sim}(s, j)\right)\Big]$$

Une politique gloutonne peut alors être définie par $\tilde{\pi}(s) = \arg\min_{j \in \mathcal{J}} c\left(s, j\right) + \gamma\Big[V^{\sim}\left(D_0^{\sim}(s, j)\right) + V^{\sim}\left(D_1^{\sim}(s, j)\right)\Big]$, avec $c(s, j)$ le coût associé au couple état/action (s, j) et $D_0^{\sim}(s, j)$ (resp. $D_1^{\sim}(s, j)$) les états enfants de s associés à l'ajout de la contrainte $x_j = 0$ (resp. $x_j = 1$) en suivant ladite politique gloutonne.

Pour apprendre une telle politique, la taille de l'espace de recherche nous force à utiliser des approximations, basées sur la méthodologie de Q-learning approximé. La fonction Q-valeur est approchée par un réseau de neurones, entraîné pour minimiser une fonction de perte construite pour prendre en compte le caractère épisodique et déterministe du MDP précédemment défini (Équation (4.12)). Une fois l'apprentissage de la Q-valeur effectué (Algorithme 6), une politique gloutonne sera utilisée, définie par

$$\pi_\theta(s) = \arg\min_{j \in \mathcal{J}} \hat{Q}(s, j; \theta)$$

avec $\hat{Q}(s, j; \theta)$ la Q-valeur prédite par le réseau de neurones paramétré par θ pour le couple état/action (s, j).

Dans un premier temps, cette approche est testée sur des modèles de coût heuristiques, basés sur des stratégies de branchement pré-existantes. Les résultats sont peu compétitifs par rapport aux performances de `CPLEX`, mais indiquent une supériorité des transitions *tree-based*.

Apprentissage par renforcement avec modèle de coût oracle – Par la suite, nous proposons d'utiliser un modèle de coût oracle, c'est-à-dire un modèle de coût tel que l'obtention d'une politique optimale pour le MDP associé garantisse la définition d'une stratégie oracle, minimisant la taille des arbres de B&B. Nous montrons que le modèle de coût unitaire, défini par $c(s, j) = 1$ pour tout couple état/action

(s, j), est un modèle oracle.

Sous les transitions *trajectory-based*, ce résultat est inconditionnel (Proposition 4.2.1). Sous les transitions *tree-based*, nous montrons que l'utilisation d'une stratégie de sélection de nœud *Depth-First Search* (DFS) ainsi qu'un facteur d'escompte unitaire $\gamma = 1$ est une condition suffisante pour obtenir l'équivalence entre l'obtention d'une politique optimale et la définition d'une stratégie oracle (Proposition 4.2.4). Dans le MDP correspondant, la valeur d'un état est égale à la taille du sous-arbre à l'état courant généré par la politique utilisée. Cela permet notamment l'obtention de signaux plus "localisés" dans l'espace, ce qui réduit drastiquement le problème de *credit assignment* induit par les transitions *trajectory-based*.

Les performances obtenues abondent dans ce sens, l'utilisation du modèle de coût unitaire et des transitions *tree-based* permettant d'obtenir des performances significativement meilleures.

Apprentissage par renforcement avec modèle de coût biaisé – Jusqu'à présent, nous avons défini des espaces d'états et d'actions cohérents avec la politique de branchement et proposé un nouveau modèle de transitions, adapté à l'environnement induit par l'algorithme de B&B. Par ailleurs, nous avons proposé un modèle de coût qui permet non seulement de garantir théoriquement l'obtention d'une stratégie oracle, mais également d'améliorer les performances empiriques. Le dernier élément du MDP qu'il nous reste à étudier est alors le facteur d'escompte γ.

Notamment, nous montrons comment l'utilisation d'un facteur d'escompte $\gamma < 1$ permet de réduire la volatilité de la fonction valeur et donc des cibles utilisées lors de l'apprentissage. En effet, cela permet de réduire la dépendance des cibles à (i) la position de l'état dans l'arbre de B&B, (ii) la qualité de l'agent, et (iii) l'instance considérée. Par ailleurs, cela encourage l'agent à générer des arbres déséquilibrés, favorisant l'apparition d'arbres en profondeur et non en largeur.

Pour autant, diminuer le facteur d'escompte comporte également des inconvénients. Théoriquement, l'équivalence entre politique optimale et stratégie oracle ne tient plus lorsque l'on considère $\gamma < 1$ sous des transitions *tree-based*. Empiriquement, le facteur d'escompte déséquilibre la répartition des valeurs, ce qui rend difficile la différenciation des états, spécialement près du nœud racine. Pour pallier ce problème, nous proposons un nouveau modèle de coût, non oracle, qui permet l'utilisation d'un facteur d'escompte $\gamma < 1$ et la standardisation des cibles au cours de l'apprentissage. Les expérimentations montrent que cela améliore significativement les performances, rendant les stratégies découvertes

compétitives avec celles produites par `CPLEX` sur des instances de taille moyenne. Cependant, ces résultats passent difficilement à l'échelle du fait du fléau de la dimension, subi de manière inhérente par toute approche d'apprentissage par renforcement.

Variations – De nombreuses pistes peuvent être explorées en complément du travail effectué sur le MDP, et nous en présentons quelques unes.
Le choix de la fonction de perte est questionné et confirmé empiriquement par la mise en comparaison avec la fonction de perte classique de DQN.
Par ailleurs, nous montrons l'importance des features utilisées pour la représentation des états. L'architecture du réseau utilisée, un MLP (Multi-Layer Perceptron) dense à 4 couches cachées, a également été investiguée sans succès. Une des raisons évoquées pour justifier la difficulté d'optimisation de l'architecture est la forte dépendance des méthodes d'apprentissage profond par renforcement (*Deep Reinforcement Learning*) aux nombreux hyperparamètres.
Afin d'améliorer l'exploration de l'espace de recherche, nous évaluons empiriquement l'utilisation de `CPLEX` comme expert, au moment de l'exploration (sélection des actions durant l'apprentissage) ou de la régression (augmentation de l'information). Les stratégies obtenues ne semblent pas significativement meilleures sur les expérimentations menées.
De même, l'utilisation de certaines bornes théoriques dans la fonction de perte ne permet pas d'améliorer les performances. Au contraire, les bornes utilisées empêchent l'agent d'apprendre à différencier les actions. Cela illustre notamment le fait que l'important en RL n'est pas de bien évaluer les différentes actions mais bel et bien de les classer efficacement.
Enfin, nous montrons que le fait d'entraîner des agents spécialisés sur chaque instance d'un même problème est ambivalent. Sur certains problèmes, cela améliore les performances observées, comme attendu du fait de la simplification de la tâche. Sur d'autres, les performances ne sont pas significativement améliorées.

Chapitre 5 : Apprentissage de la stratégie de sélection de nœud

Après avoir étudié l'apprentissage d'une stratégie de branchement, nous nous intéressons à l'apprentissage de la stratégie de sélection de nœud. Cet apprentissage s'avère plus aisé que celui de la stratégie de branchement, et ce pour deux raisons principales. D'une part, l'espace de recherche considéré est

moins vaste. D'autre part, nous sommes en mesure d'exhiber formellement une stratégie oracle.

Définition d'une stratégie oracle – Comme précédemment, nous effectuons l'hypothèse de transitions déterministes, et plus précisément celle d'une stratégie de branchement déterministe. La conséquence majeure de cette hypothèse est que l'on peut garantir l'existence et formellement exhiber une stratégie de sélection de nœud DFS oracle (Proposition 5.1.3). Dès lors, nous pouvons nous restreindre aux stratégies DFS et donc considérer un espace d'actions binaires au sein de notre MDP, une action consistant à choisir un nœud à visiter parmi les deux nœuds enfants du nœud courant. De plus, contrairement au cas de la stratégie de branchement, nous pouvons calculer hors ligne les choix oracles, comme la sélection du nœud enfant contenant la solution optimale au sous-problème associé au nœud courant.

Stratégies d'apprentissage – Nous proposons d'utiliser les choix oracles de plusieurs manières. Tout d'abord, ils peuvent être utilisés hors ligne, comme déjà effectué dans la littérature [75, 76]. Un classifieur est alors entraîné pour imiter les choix oracles sur un ensemble d'états explorés en apprentissage. La distribution de ces états d'apprentissage diffèrera alors, selon qu'ils soient générés par ladite stratégie oracle (*Behavioral Cloning* [4]) ou bien par une politique évoluant en même temps que le classifieur (*Dataset Aggregation* [7]).
L'inconvénient partagé par ces approches est la non prise en compte du coût associé à la prise d'une décision non oracle. Le fait que ce coût soit non constant au sein d'un même arbre de B&B nous pousse alors à étudier l'approche par renforcement pour l'apprentissage de la stratégie de sélection de nœud. En effet, la généralisation de la méthodologie présentée au chapitre précédent possède l'avantage de naturellement prendre en compte ce coût en prédisant la taille du sous-arbre au nœud courant.
Malgré les mêmes garanties théoriques sur la capacité de l'approche par renforcement à trouver une stratégie oracle, le besoin pratique d'une meilleure efficacité d'échantillonnage nous pousse à vouloir utiliser les informations fournies par la stratégie oracle connue pour guider l'apprentissage. Nous proposons alors de modifier la fonction de perte guidant l'apprentissage par renforcement en utilisant la stratégie oracle connue pour augmenter la quantité d'information utilisée (Équation (5.11)).

Expérimentations – Nous montrons empiriquement qu'utiliser les informations fournies par l'oracle améliore les performances de l'approche par renforcement. Pour autant, les approches supervisées

permettent en moyenne d'obtenir de meilleurs résultats. Les stratégies découvertes, que ce soit en supervisé ou par renforcement, sont en général meilleures que celles utilisées par `CPLEX`.

Chapitre 6 : Apprentissage par renforcement pour la stratégie de branchement et de sélection de nœud

Nous étudions diverses approches pour combiner l'apprentissage de la stratégie de branchement et de la sélection de nœud. Tout d'abord, le cadre de RL utilisé jusqu'à présent peut aisément être élargi pour effectuer à la fois le branchement et la sélection de nœud. Il est également possible de combiner cette approche avec les informations tirées de la stratégie oracle précédemment évoquée en sélection de nœud. Enfin, nous proposons d'utiliser deux agents distincts, l'un entraîné par renforcement pour apprendre la stratégie de branchement et l'autre par imitation pour apprendre la stratégie de sélection de nœud.
Cette dernière méthodologie fournit les meilleurs résultats. Cependant, ils restent inférieurs à ceux obtenus en apprenant uniquement la stratégie de branchement, ce qui traduit un problème de coordination entre les deux agents apprenants.

Partie 3 : Exploitation de la structure des problèmes considérés

Cette partie regroupe trois approches indépendantes exploitant la structure des problèmes étudiés. Ici, nous ne cherchons pas à garantir la découverte de stratégies oracles.

Chapitre 7 : Une heuristique de branchement basée sur une représentation par graphe

Dans ce chapitre, nous proposons une heuristique de branchement, basée sur une représentation par graphe de l'instance considérée. Le but de cette représentation par graphe est de modéliser l'*influence* du branchement sur chaque variable binaire sur les variables restantes. Une fois ce graphe d'influence construit, l'heuristique consistera à brancher sur les variables ayant les plus fortes influences.

Graphe d'influence – Nous définissons un graphe d'influence pour un MILP comme étant un type particulier de graphe primal, c'est-à-dire où les nœuds représentent les variables et où une arête existe entre deux nœuds si les deux variables correspondantes sont présentes dans une même contrainte. Nous construisons cette classe de graphes de manière relativement large, et nous proposons alors différentes instanciations, où l'influence (le poids des arêtes) est définie en prenant en compte les coefficients de la matrice de contraintes et potentiellement une solution du problème dual relâché. Intuitivement, les définitions d'influence sont censées refléter la capacité de chaque variable à fixer les autres à leurs bornes lors d'un branchement.
L'intérêt d'une telle représentation par graphe est notamment son invariance par permutation et sa capacité à modéliser des liens complexes entre les différentes variables. Il est cependant à noter que les définitions d'influence proposées sont sensibles à l'échelle des données, ce qui incite à effectuer une standardisation de ces dernières.

Heuristique de branchement – Etant donné un graphe d'influence associé à un MILP, il semblerait naturel de chercher à brancher sur la variable ayant l'influence maximale. Le problème de maximisation d'influence étant en général NP-difficile, nous optons pour une méthode approchée basée sur une partition des nœuds du graphe d'influence par *Spectral Clustering*, visant à sélectionner plusieurs variables à forte influence. L'heuristique que nous proposons est alors définie comme suit. Soit $K \geqslant 1$ le nombre de clusters désiré, une K-partition des variables binaires est générée au nœud racine, chaque cluster représentant un ensemble de variables étroitement liées. Au sein de chaque cluster, la variable de plus forte influence est sélectionnée. Les K variables ainsi définies sont ordonnées en fonction de leurs influences respectives, et la i-ième variable est alors utilisée pour le branchement en chaque nœud de l'arbre de B&B de profondeur $i - 1$ (cf. Figure 7.2).

Expérimentations – Les résultats observés sont relativement hétérogènes. En particulier, aucune hiérarchie n'est constatée entre les différentes définitions d'influence testées. Cependant, nous notons que l'heuristique proposée est généralement efficace pour les instances que `CPLEX` peine à résoudre, ce qui justifie l'intérêt porté dans la littérature à la représentation par graphe pour l'étude de problèmes combinatoires. Par ailleurs, il est intéressant de constater que les meilleurs résultats sont obtenus pour de grandes valeurs de K, ce qui laisse penser que la stratégie de clustering est pertinente.

Chapitre 8 : Une approche par décomposition-coordination

Contrairement au chapitre précédent, nous nous attachons ici à utiliser une structure spécifique des problèmes qui rencontrés chez EDF, à savoir une structure bloc-diagonale.

Stratégie de découplage – Du fait de cette structure bloc-diagonale, les problèmes peuvent être perçus comme une juxtaposition de sous-problèmes indépendants liés par un ensemble de variables couplantes. Dès lors, si ces variables couplantes étaient fixées (par exemple du fait de décisions de branchement), le problème correspondant serait découplé et pourrait être résolu par la résolution parallèle des sous-problèmes indépendants. Etant donné que les variables couplantes sont rarement toutes fixées, nous nous intéressons à une résolution heuristique, sans garantie d'optimalité. ous nous basons notamment sur une procédure de Relax and Fix (RF, Algorithme 9), qui consiste en la résolution successive de sous-problèmes, partiellement relâchés, en fixant progressivement les variables aux valeurs précédemment obtenues. Lorsque le problème est découplé, une stratégie de RF permet alors de trouver une solution optimale.

Decouple, Relax and Fix – Nous rapprochons la procédure de RF de la décomposition lagrangienne et proposons de la généraliser afin d'obtenir de meilleures solutions. Cela est effectué en intégrant la procédure de RF au sein d'une recherche sur un sous-ensemble faisable de variables couplantes. Une version exacte, mais généralement non tractable, de l'algorithme proposé peut s'écrire, pour un problème de maximisation,

$$\max_{x_c \in X_{\mathcal{C}}} \{RF\left(p(x_c), \mathcal{G}\right)\}$$

avec $p(x_c)$ le problème p augmenté des contraintes $x_{\mathcal{C}} = x_c$, $X_{\mathcal{C}}$ l'ensemble faisable pour les variables couplantes $\mathcal{C}$, et $RF(p, \mathcal{G})$ la borne obtenue par RF en utilisant la décomposition ordonnée $\mathcal{G}$.
Par la suite, nous incorporons la maximisation sur $X_{\mathcal{C}}$ dans un algorithme de B&B et sélectionnons un sous-ensemble d'intérêt lorsqu'une recherche exhaustive n'est pas envisageable. Nous nous restreignons à l'étude des cas où les variables couplantes sont uniquement binaires ou uniquement continues.

Variables couplantes binaires – Lorsque les variables couplantes sont binaires, l'exploration de $X_{\mathcal{C}}$ peut aisément s'effectuer par Branch and Bound, ce qui nous permet d'obtenir une garantie d'optimalité. Les décisions de branchement sont alors restreintes aux variables couplantes dans un premier temps

au sein de ce que nous appelons un arbre *découplant*. A chaque feuille faisable et non prouvée sous-optimale, une procédure de RF est initiée afin d'obtenir si possible une nouvelle borne.

Variables couplantes continues – Dans le cas où les variables couplantes sont continues, la procédure de découplage se révèle être légèrement plus complexe.
Tout d'abord, nous étudions le cas d'une décomposition en deux blocs, avec une unique variable couplante continue. L'espace $X_{\mathcal{C}}$ étant dans ce cas un intervalle, nous restreignons la recherche à un sous-ensemble de points au sein de ce dernier. La sélection de ces points est basée sur le fait que la fonction GRF, définie comme

$$GRF : \begin{cases} X_{\mathcal{C}} \to \mathbb{R} \\ u \mapsto RF\left(p(u), \mathcal{G}\right) \end{cases}$$

est une fonction constante par morceaux. Malgré l'existence de cas pathologiques pour lesquels certains plateaux peuvent être de mesure nulle (Proposition 8.2.2), nous restreignons la recherche d'un plateau de valeur optimale à une recherche discrète, basée sur les raisons pour lesquelles la fonction GRF présente des discontinuités. Nous identifions deux types de discontinuités : (i) les discontinuités de premier ordre, induites par l'infaisabilité de la solution entière après translation des contraintes, (ii) la sous-optimalité de cette solution.
Les discontinuités de premier ordre sont relativement aisées à déterminer, puisqu'elles ne nécessitent que de tester la mesure nulle d'un polytope. A l'inverse, les discontinuités de second ordre requièrent la résolution d'un MILP, ce qui justifie notre choix de ne pas exhiber ces dernières. Nous proposons alors d'effectuer une recherche dichotomique sur l'intervalle $X_{\mathcal{C}}$ (potentiellement réduit par *bound tightening*) afin de trouver ces discontinuités de premier ordre.
Une telle approche se généralise difficilement à plus de deux blocs du fait du fléau de la dimension. Partant, nous proposons un algorithme arborescent permettant de restreindre la recherche. Cette restriction repose, tout comme la procédure de RF, sur la définition d'une décomposition ordonnée des variables.

Expérimentations – Pour les expérimentations, nous proposons trois types de décompositions ordonnées. Les décompositions temporelle et spatiale sont basées sur la connaissance *a priori* de la structure des problèmes étudiés. Par ailleurs, nous proposons une décomposition spectrale, visant à créer une décomposition pertinente sans connaissance *a priori*. Cette décomposition est ensuite heuristiquement

exploitée en maximisant les interconnections entre les différents blocs consécutifs.
Les résultats des tests effectués montrent que les méthodes proposées permettent d'obtenir des bornes satisfaisantes en réduisant drastiquement le nombre de nœuds explorés comparativement à `CPLEX`. Par ailleurs, elles permettent de trouver plus souvent une borne par rapport à la procédure classique de RF. Bien évidemment, la décomposition choisie ainsi que l'ordre induit par cette dernière influent sur les performances observées.

Chapitre 9 : Perturbation de la fonction objectif

Dans ce chapitre, nous étudions l'apprentissage de perturbations visant à réduire les symétries inhérentes au problème considéré.

Paradigme de l'optimisation boîte noire – Soit $f(p, \varepsilon)$ une mesure évaluant l'efficacité d'appliquer la perturbation $\varepsilon \in \Omega \subseteq \mathbb{R}^n$ à la fonction objectif du problème p, avec n le nombre de variables. L'objectif recherché est alors de découvrir en moyenne la perturbation optimale pour un problème donné, c'est-à-dire trouver le minimum de la fonction boîte noire (*black-box*)

$$\mu \ : \ \begin{cases} \Omega \to \mathbb{R} \\ \varepsilon \mapsto \mu(\varepsilon) = \mathbb{E}_{p \sim \mathcal{L}} \left[f(p, \varepsilon) \right] \end{cases}$$

avec $\mathcal{L}$ la distribution inconnue des instances du problème considéré.
L'ajout d'une perturbation à la fonction objectif pouvant modifier la solution trouvée, nous montrons en se restreignant aux problèmes binaires purs qu'il suffit de contrôler la norme de la perturbation pour garantir la non détérioration de la qualité de la solution trouvée.

Réduction de la dimension – L'espace de recherche considéré étant une boule dans $\mathbb{R}^n$, nous proposons de réduire sa dimension avant d'appliquer un algorithme d'optimisation boîte noire. Dans un premier temps, nous justifions empiriquement le choix de ne considérer qu'une sphère et non une boule, la "forme" et la norme d'une perturbation apparaissant comme deux déterminants indépendants de la performance. Par ailleurs, nous proposons l'utilisation d'un auto-encodeur supervisé permettant d'encoder et de décoder les perturbations dans un espace latent de dimension réduite, tout en guidant ce mapping pour assurer que l'espace latent soit structurellement pertinent pour l'évaluation d'une perturbation.

Un auto-encodeur supervisé classique est un système composé de trois fonctions : la fonction d'encodage $\phi : \Omega \to \phi(\Omega)$, la fonction de décodage $\phi : \phi(\Omega) \to \Omega$ et la fonction de prédiction $h : \phi(\Omega) \to \mathbb{R}$, et peut alors être entraîné grâce aux fonctions de perte supervisée l_s et de reconstruction l_r, par exemple :

$$\begin{cases} l_s\left(h, \phi\right) = \mathbb{E}_{\varepsilon \sim \rho}\left[||\left(h \circ \phi\right)\left(\varepsilon\right) - \mu(\varepsilon)||_2^2\right] \\ l_r\left(\phi, \phi^-\right) = \mathbb{E}_{\varepsilon \sim \rho}\left[||\left(\phi \circ \phi^-\right)\left(\varepsilon\right) - \varepsilon||_2^2\right] \end{cases}$$

avec ρ la distribution des perturbations générées.
Dès lors, nous proposons de traiter notre problème d'optimisation boîte noire en générant des perturbations au moyen d'itérations successives de la séquence (i) générer des points dans l'espace latent $\phi(\Omega)$, (ii) décoder ces points pour obtenir des perturbations dans $\Omega \subset \mathbb{R}^n$, (iii) évaluer ces perturbations, (iv) entraîner l'auto-encodeur supervisé en utilisant les nouvelles observations (Algorithme 18). Par ailleurs, nous questionnons la perte de reconstruction précédente. En effet, la distance euclidienne entre une perturbation et sa reconstruction ne semble pas pertinente dans notre cadre d'optimisation boîte noire. A l'inverse, il semble plus pertinent d'être capable de décoder des points de l'espace latent en perturbations ayant la même efficacité. Dès lors, nous proposons d'utiliser la perte de reconstruction $l_r\left(\phi, \phi^-\right) = \mathbb{E}_{\varepsilon \sim \rho}\left[||\left(\phi \circ \phi^- \circ \phi\right)\left(\varepsilon\right) - \phi\left(\varepsilon\right)||_2^2\right]$, en tirant parti du fait que la distance euclidienne dans l'espace latent est plus informationnelle que la distance dans Ω.

Expérimentations – Dans un premier temps, nous montrons que la nouvelle fonction de perte de reconstruction améliore la stabilité de l'auto-encodeur face à un cycle complet de décodage-encodage. Pour autant, les résultats de la procédure d'optimisation boîte noire sont décevants, non seulement en termes d'efficacité relative mais également en comparaison avec une stratégie d'échantillonnage aléatoire, non guidée par l'auto-encodeur.

Conclusion

Le travail mené tout au long de cette thèse a permis de développer une approche d'apprentissage par renforcement visant à découvrir de nouvelles stratégies de B&B, l'accent ayant été mis sur la découverte de stratégies oracles. Par ailleurs, il fut également l'occasion d'étudier d'autres paradigmes, comme l'apprentissage par imitation, le *clustering* et l'optimisation boîte noire.
Concernant l'approche par renforcement, les deux principales pistes identifiées pour améliorer les

performances de la méthodologie proposée sont l'amélioration de la représentation des états considérés ainsi que l'optimisation de la stratégie d'exploration. Ces deux pistes ne sont bien évidemment pas indépendantes, une meilleure représentation des états permettant à l'agent d'associer ou de distinguer plus facilement différentes zones de l'espace d'états, et donc d'améliorer l'échantillonnage de celui-ci, autrement dit l'exploration. Une des pistes envisagées pour améliorer la représentation des états est notamment l'utilisation de *Graph Convolutional Neural Networks*, ainsi qu'une construction de *features* spécifique au problème considéré. Afin de guider l'exploration de manière plus explicite, l'utilisation d'experts pour guider l'apprentissage pouvait être proposé, ou encore l'utilisation de méthodes d'apprentissage par transfert, permettant de généraliser des stratégies découvertes en petite dimension à des problèmes de plus grande dimension.

Ces améliorations participeraient notamment à la possible utilisation de la méthodologie proposée à des problèmes de plus grande dimension que ceux considérés dans ce travail.

Bibliography

[1] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 105–132.

[2] H. Martens and T. Naes, *Multivariate calibration*. John Wiley & Sons, 1992.

[3] C. M. Bishop, "Pattern recognition," *Machine learning*, vol. 128, no. 9, 2006.

[4] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie-Mellon Univ Pittsburgh PA Artificial Intelligence And Psychology, Tech. Rep., 1989.

[5] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, no. 3, pp. 297–325, 2009.

[6] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 661–668.

[7] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[8] H. He, J. Eisner, and H. Daume, "Imitation learning by coaching," *Advances in Neural Information Processing Systems*, vol. 25, pp. 3149–3157, 2012.

[9] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint arXiv:1406.5979*, 2014.

[10] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[13] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[14] 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*, vol. 15, no. 1, pp. 1929–1958, 2014.

[15] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[16] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

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

[18] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The loss surfaces of multilayer networks," in *Artificial intelligence and statistics*. PMLR, 2015, pp. 192–204.

[19] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[22] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[23] D. P. De Farias, "The linear programming approach to approximate dynamic programming: Theory and application," Ph.D. dissertation, stanford university, 2002.

[24] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. Citeseer, 1994, vol. 37.

[25] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[26] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European conference on machine learning*. Springer, 2005, pp. 317–328.

[27] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[29] S. P. Singh and R. C. Yee, "An upper bound on the loss from approximate optimal-value functions," *Machine Learning*, vol. 16, no. 3, pp. 227–233, 1994.

[30] M. Petrik and B. Scherrer, "Biasing approximate dynamic programming with a lower discount factor," *Advances in neural information processing systems*, vol. 21, pp. 1265–1272, 2008.

[31] R. Amit, R. Meir, and K. Ciosek, "Discount factor as a regularizer in reinforcement learning," in *International conference on machine learning*. PMLR, 2020, pp. 269–278.

[32] V. François-Lavet, R. Fonteneau, and D. Ernst, "How to discount deep reinforcement learning: Towards new dynamic strategies," *arXiv preprint arXiv:1512.02011*, 2015.

[33] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

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

[35] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[36] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *31st Conference on Neural Information Processing Systems (NIPS)*, vol. 30, 2017, pp. 1–18.

[37] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[38] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[39] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis *et al.*, "Noisy networks for exploration," in *International Conference on Learning Representations*, 2018.

[40] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," *Advances in neural information processing systems*, vol. 29, pp. 4026–4034, 2016.

[41] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[42] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97. Citeseer, 1997, pp. 12–20.

[43] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[44] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[45] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.

[46] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama, "Active deep q-learning with demonstration," *Machine Learning*, pp. 1–27, 2019.

[47] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz *et al.*, "Miplib 2010," *Mathematical Programming Computation*, vol. 3, no. 2, pp. 103–163, 2011.

[48] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[49] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of combinatorial optimization*. Springer, 2013, pp. 449–481.

[50] M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, "Experiments in mixed-integer linear programming," *Mathematical Programming*, vol. 1, no. 1, pp. 76–94, 1971.

[51] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Finding cuts in the tsp (a preliminary report)," Citeseer, Tech. Rep., 1995.

[52] T. Achterberg, "Constraint integer programming," 2007.

[53] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," *Operations Research Letters*, vol. 33, no. 1, pp. 42–54, 2005.

[54] J. T. Linderoth and M. W. Savelsbergh, "A computational study of search strategies for mixed integer programming," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 173–187, 1999.

[55] T. Achterberg and T. Berthold, "Hybrid branching," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2009, pp. 309–311.

[56] J. Patel and J. W. Chinneck, "Active-constraint variable ordering for faster feasibility of mixed integer linear programs," *Mathematical Programming*, vol. 110, no. 3, pp. 445–474, 2007.

[57] R. Gomory, "An algorithm for the mixed integer problem," Rand Corp. Santa Monica CA, Tech. Rep., 1960.

[58] M. E. Lübbecke, "Column generation," *Wiley encyclopedia of operations research and management science. Wiley, New York*, pp. 1–14, 2010.

[59] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[60] T. Berthold, "Measuring the impact of primal heuristics," *Operations Research Letters*, vol. 41, no. 6, pp. 611–614, 2013.

[61] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.

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

[63] A. M. Alvarez, Q. Louveaux, and L. Wehenkel, "A supervised machine learning approach to variable branching in branch-and-bound," in *In ecml*. Citeseer, 2014.

[64] A. Marcos Alvarez, "Computational and theoretical synergies between linear optimization and supervised machine learning," Ph.D. dissertation, Université de Liège, Liège, Belgique, 2016.

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

[66] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 133–142.

[67] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[68] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *NeurIPS*, 2019.

[69] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *arXiv preprint arXiv:2102.09544*, 2021.

[70] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio, "Parameterizing branch-and-bound search trees to learn branching policies," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 3931–3939.

[71] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Ubbecke, S. Maher, M. Miltenberger, M. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, S. Schl¨osser, C. Schubert, F. Serrano, and M. Lübbecke, "The scip optimization suite 6.0," 07 2018.

[72] A. Marcos Alvarez, L. Wehenkel, and Q. Louveaux, "Online learning for strong branching approximation in branch-and-bound," 2016.

[73] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *International conference on machine learning*. PMLR, 2018, pp. 344–353.

[74] G. Di Liberto, S. Kadioglu, K. Leo, and Y. Malitsky, "Dash: Dynamic approach for switching heuristics," *European Journal of Operational Research*, vol. 248, no. 3, pp. 943–953, 2016.

[75] K. Yilmaz and N. Yorke-Smith, "A study of learning search approximation in mixed integer branch and bound: Node selection in scip," *AI*, vol. 2, no. 2, pp. 150–178, 2021.

[76] H. He, H. Daume III, and J. M. Eisner, "Learning to search in branch and bound algorithms," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3293–3301. [Online]. Available: http://papers.nips.cc/paper/5495-learning-to-search-in-branch-and-bound-algorithms.pdf

[77] R. Baltean-Lugojan, P. Bonami, R. Misener, and A. Tramontani, "Selecting cutting planes for quadratic semidefinite outer-approximation via trained neural networks," Technical Report, CPLEX Optimization, IBM, Tech. Rep., 2018.

[78] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9367–9376.

[79] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[80] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *NIPS*, 2017.

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

[82] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*, 2015.

[83] M. Fischetti, A. Lodi, and G. Zarpellon, "Learning milp resolution outcomes before reaching time-limit," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2019, pp. 275–291.

[84] G. Iommazzo, C. D'Ambrosio, A. Frangioni, and L. Liberti, "A learning-based mathematical programming formulation for the automatic configuration of optimization solvers," in *International Conference on Machine Learning, Optimization, and Data Science*. Springer, 2020, pp. 700–712.

[85] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[86] M. Kruber, M. E. Lübbecke, and A. Parmentier, "Learning when to use a decomposition," in *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems*. Springer, 2017, pp. 202–210.

[87] E. Rachelson, A. B. Abbes, and S. Diemer, "Combining mixed integer programming and supervised learning for fast re-planning," in *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, vol. 2. IEEE, 2010, pp. 63–70.

[88] D. Bertsimas and B. Stellato, "The voice of optimization," *Machine Learning*, vol. 110, no. 2, pp. 249–277, 2021.

[89] N. P. Padhy, "Unit commitment-a bibliographical survey," *IEEE Transactions on power systems*, vol. 19, no. 2, pp. 1196–1205, 2004.

[90] P. Bendotti, P. Fouilhoux, C. Rottner *et al.*, "On the complexity of the unit commitment problem." *Ann. Oper. Res.*, vol. 274, no. 1-2, pp. 119–130, 2019.

[91] F. Kılınç-Karzan, G. Nemhauser, and M. Savelsbergh, "Information-based branching schemes for binary linear mixed integer problems," *Mathematical Programming Computation*, vol. 1, pp. 249–293, 12 2009.

[92] M. Fischetti and M. Monaci, "Branching on nonchimerical fractionalities," *Operations Research Letters*, vol. 40, no. 3, pp. 159–164, 2012.

[93] W. Glankwamdee and J. Linderoth, "Lookahead branching for mixed integer programming," Citeseer, Tech. Rep., 2006.

[94] A. Gilpin and T. Sandholm, "Information-theoretic approaches to branching in search," *Discrete Optimization*, vol. 8, no. 2, pp. 147–159, 2011.

[95] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[96] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[97] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[98] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, "Orbital branching," *Mathematical Programming*, vol. 126, no. 1, pp. 147–178, 2011.

[99] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1852–1872, 2018.

[100] S.-W. Son, H. Jeong, and J. D. Noh, "Random field ising model and community structure in complex networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 50, no. 3, pp. 431–437, 2006.

[101] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[102] D. Ferreira, R. Morabito, and S. Rangel, "Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants," *Computers & Operations Research*, vol. 37, no. 4, pp. 684–691, 2010.

[103] M. Guignard and S. Kim, "Lagrangean decomposition: A model yielding stronger lagrangean bounds," *Mathematical programming*, vol. 39, no. 2, pp. 215–228, 1987.

[104] O. L. Mangasarian and T.-H. Shiau, "Lipschitz continuity of solutions of linear inequalities, programs and complementarity problems," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, pp. 583–595, 1987.

[105] I. Adler and R. D. Monteiro, "A geometric view of parametric linear programming," *Algorithmica*, vol. 8, no. 1, pp. 161–176, 1992.

[106] M. W. Savelsbergh, "Preprocessing and probing techniques for mixed integer programming problems," *ORSA Journal on Computing*, vol. 6, no. 4, pp. 445–454, 1994.

[107] T. Walsh, *Symmetry in Constraint Optimization*, 2011.

[108] F. Margot, "Symmetry in integer linear programming," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 647–686.

[109] C. Rottner, "Combinatorial aspects of the unit commitment problem," Ph.D. dissertation, Sorbonne Université, 2018.

[110] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[111] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[112] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Fitness expectation maximization," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2008, pp. 337–346.

[113] F. Stulp and O. Sigaud, "Policy improvement methods: Between black-box optimization and episodic reinforcement learning," 2012.

[114] L. Le, A. Patterson, and M. White, "Supervised autoencoders: Improving generalization performance with unsupervised regularizers," in *Advances in Neural Information Processing Systems*, 2018, pp. 107–117.

[115] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[116] M. Karamanov and G. Cornuéjols, "Branching on general disjunctions," *Mathematical Programming*, vol. 128, no. 1, pp. 403–436, 2011.

[117] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi, "Ecole: A gym-like library for machine learning in combinatorial optimization solvers," *arXiv preprint arXiv:2011.06069*, 2020.

le cnam

Marc Etheve

Using machine learning to solve repeated optimization problems

le cnam

Résumé : Cette thèse a pour but d'utiliser des techniques d'apprentissage automatique pour la résolution de problèmes linéaires en nombres entiers issus de données stochastiques. Plutôt que de les résoudre indépendamment, nous proposons de tirer profit des similarités entre instances en apprenant différentes stratégies au sein d'un algorithme de Branch and Bound (B&B).
L'axe principal développé est l'utilisation d'apprentissage par renforcement pour découvrir des stratégies minimisant la taille des arbres de B&B. Afin de s'adapter à l'environnement induit par l'algorithme de B&B, nous définissons un nouveau type de transitions au sein de processus de décision markoviens, basées sur la structure d'arbre binaire. Par ailleurs, nous étudions différents modèles de coûts et prouvons l'optimalité du modèle de coût unitaire sous les transitions classiques et binaires, dans l'apprentissage des stratégies de branchement et de sélection de nœud. Pour autant, les expérimentations menées suggèrent qu'il peut être préférable de biaiser le modèle de coût afin d'améliorer la stabilité du processus d'apprentissage. En ce qui concerne la stratégie de sélection de nœud, nous démontrons l'optimalité d'une stratégie explicitement définie, qui peut être apprise plus efficacement de manière supervisée.
Par ailleurs, nous proposons d'exploiter la structure des problèmes étudiés. Nous étudions pour cela une stratégie de décomposition-coordination, une heuristique de branchement basée sur une représentation par graphe d'un nœud de l'arbre de B&B et enfin l'apprentissage de perturbations de la fonction objectif.

Mots clés : Apprentissage, MILP, Problèmes répétés

Abstract : This thesis aims at using machine learning techniques in the context of Mixed Integer Linear Programming instances generated by stochastic data. Rather than solve these instances independently using the Branch and Bound algorithm (B&B), we propose to leverage the similarities between instances by learning inner strategies of this algorithm, such as node selection and branching.
The main approach developed in this work is to use reinforcement learning to discover by trials-and-errors strategies which minimize the B&B tree size. To properly adapt to the B&B environment, we define a new kind of tree-based transitions, and elaborate on different cost models in the corresponding Markov Decision Processes. We prove the optimality of the unitary cost model under both classical and tree-based transitions, either for branching or node selection. However, we experimentally show that it may be beneficial to bias the cost so as to improve the learning stability. Regarding node selection, we formally exhibit an optimal strategy which can be more efficiently learnt directly by supervised learning. In addition, we propose to exploit the structure of the studied problems. To this end, we propose a decomposition-coordination methodology, a branching heuristic based on a graph representation of a B&B node and finally an approach for learning to disrupt the objective function.

Keywords: Machine Learning, MILP, Repeated problems