Decomposition algorithms for deterministic and uncertain integer programs

B. Detienne

Université de Bordeaux

Institut de Mathématiques de Bordeaux

Thèse d'habilitation à diriger des recherches

présentée par

Boris Detienne

SPÉCIALITÉ : MATHEMATIQUES APPLIQUEES ET CALCUL SCIENTIFIQUE

Decomposition algorithms for deterministic and uncertain integer programs

---

Date de soutenance : 15 décembre 2021

Devant la commission d'examen composée de :

Christian Artigues .	Directeur de recherche, LAAS-CNRS, Toulouse	Rapporteur
Luce Brotcorne ..	Directeur de recherche, Inria, Lille	Examinateur
François Clautiaux .	Professeur, Université de Bordeaux	Examinateur
Safia Kedad-Sidhoum	Professeur, CNAM, Paris	Rapporteur
Ivana Ljubic	Professeur, ESSEC Business School, Paris ..	Examinateur
Gautier Stauffer ..	Professeur, Kedge Business School, Bordeaux	Examinateur
Wolfram Wiesemann	Professeur, Imperial College, London	Rapporteur

– 2021 –

Acknowledgments

First of all I would like to thank Isia, Dimitri and my partner for their love and support.

I would like to thank the reviewers of this HDR, Chrisitan Artigues, Safia Kedad-Sidhoum and Wolfram Wiesemann, as well as the other members of the jury, Luce Brotcorne, François Clautiaux, Ivana Ljubic and Gautier Stauffer for accepting this time-consuming charge.

I am very grateful to Éric Pinson, David Rivreau, Stéphane Dauzère-Pérès, François Clautiaux, François Vanderbeck and Ayşe Nur Arslan for, each in their own way, forging my vision of research.

Finally, I would like to thank my peers and colleagues for their friendship and/or fruitful scientific discussions: Céline Turbillon, Olivier Guyon, Laurent Péridy, Nabil Absi, Claude Yugma, Dominique Feillet, Dominique Quadri, C. Diego Rodrigues, Evgeny Gurevsky, Laurent Facq, Aurélien Froger, Pierre Pesneau, Ruslan Sadykov, Gautier Stauffer, Michaël Poss, Quentin Viaud, Super Guillot Bros., Mohamed Benkirane, Rodolphe Griset, Henri Lefebvre, Xavier Blanchot, Agnès Leroux, Halil Şen, Shunji Tanaka, Pascale Bendotti, Marc Porcheron, and many others.

Contents

List of Acronyms

Chapter 1

Introduction

Contents

This manuscript describes the core of my research activities, which consist of solving mathematical optimization problems based on mathematical programming paradigms, with a focus on the computational performance of the developed algorithms. Decomposition approaches are used to solve hard integer linear programs, that represent real-life economical problems or archetypical idealized problems. The first chapter introduces the types of problems addressed and the different methodologies employed. Chapter 2 concerns deterministic optimization, and in particular two representative studies based on State-Space Relaxation techniques. In Chapter 3, the most significant research about optimization under uncertainty is exposed.

The rest of this chapter is organized as follows. Section 1.1 presents mathematical models and notations encountered throughout the manuscript. Section 1.2 describes methodological tools, recalling their basic principle and mentioning their usage in my research. The main contributions are summarized in Section 1.3.

1.1 Mathematical programming models

This work concerns optimization problems that arise in various application fields of Operations Research (OR), mostly in planning and scheduling. The following

formulation encompasses all problems studied:

$$\inf \quad f(\boldsymbol{x}) \tag{1.1.1}$$
$$\text{s.t.} \quad \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \tag{1.1.2}$$
$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.3}$$

In this manuscript, vectors and matrices are indicated in boldface, whereas a scalar or vector component is non-boldface.

This class of models aims at determining the value of n real decision variables $\boldsymbol{x}$ that minimizes the objective function $f : \mathbb{R}^n \to \mathbb{R}$. The feasible set is composed of vectors of the decision variables that satisfy constraints (1.1.2), where $\boldsymbol{g} = (g_i)_{i\in\{1,\dots,m\}}$ is a vector of functions $g_i : \mathbb{R}^n \to \mathbb{R}$. A large variety of OR applications involve discrete decisions. Constraints (1.1.3) embed variable domain constraints: the typical form of $\mathcal{S}$ is $\mathcal{S} = \mathbb{R}^p_+ \times \mathbb{N}^{n-p}$ with $p \in \mathbb{N}$ the number of decision variables restricted to take integer values.

Unless stated otherwise, we consider finite-dimensional (n and m assume finite value) and single-objective optimization models.

1.1.1 Deterministic mathematical programs

This section introduces two important types of mathematical programs. The class of deterministic Mixed Integer Linear Programming (MILP) models is at the heart of the contributions exposed in this manuscript. Chapter 2 is dedicated to algorithms designed to solve such problems, and the contributions in Chapter 3 are based on deterministic reformulations of uncertain problems.

Mixed integer linear programs This special case is defined by a linear objective function, and affine constraints. The topic is covered by [Schrijver 1986, Wolsey & Nemhauser 1999, Wolsey 2020]. MILP models are then often represented using matrix notations:

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} \tag{1.1.4}$$
$$\text{s.t.} \quad \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{1.1.5}$$
$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.6}$$

Although most real situations involve non-linear systems and uncertain data, there are many reason to consider this simplified setting. The first reason is its high expressive power when it comes to describing OR applications. In a wide range of practical contexts, it is acceptable to neglect the uncertainty in the input data, in the sense that the solution obtained disregarding it will remain practically implementable despite possible variation. Likewise, some non-linearities in the system may be described accurately enough with linear expressions, possibly at the price of introducing additional variables and constraints. In these situations, the cost estimated on the basis of a deterministic input data set and linear model can also be

considered a sufficiently close approximation to the actual cost of the implemented solution.

The second reason is that effective algorithms to solve this kind of mathematical programs have now been developed for decades. Nowadays, off-the-shelf libraries with high-level Application Programming Interface (API) are available to researchers and practitioners and considerably facilitate the development of MILP-based methods.

This leads to the third reason: some uncertain mathematical programs can be recast – sometimes approximately – as deterministic MILP models, or can be tackled with help of procedures that iteratively solve such models. The degree of efficiency attained by MILP solvers also make them tools of choice to develop competitive algorithms for more complex problems.

Bi-level programs Bi-level programming typically models situations where a pair of actors, with potentially conflicting interests, have to make decisions in a system where their choices interact with each other. The terminology *leader/follower* is often used in the literature to refer to the sequential decision making, with the leader making a decision first and the follower second. In this paradigm, the leader has to choose his optimal decision, knowing that the follower will react to these decisions by optimizing their own objective function (see for example [Dempe 2002]).

$$(Bi-level): \min_{\boldsymbol{\alpha}\in\Lambda,\boldsymbol{\beta}} F(\boldsymbol{\alpha},\boldsymbol{\beta}) \tag{1.1.7}$$

$$s.t. \quad G(\boldsymbol{\alpha},\boldsymbol{\beta}) \leq 0 \tag{1.1.8}$$

$$\boldsymbol{\beta} \in \arg\min_{\boldsymbol{\beta}'\in\Pi}\{f(\boldsymbol{\alpha},\boldsymbol{\beta}') : g(\boldsymbol{\alpha},\boldsymbol{\beta}') \leq 0\} \tag{1.1.9}$$

Here, $\boldsymbol{\alpha} \in \Lambda \subset \mathbb{R}^{n_1}$ (resp. $\boldsymbol{\beta} \in \Pi \subset \mathbb{R}^{n_2}$) is the set of leader's (resp. follower's) decision variables. Functions F and f are the objective functions of the leader and the follower, respectively, while functions G and g are used to define the actors' interacting constraints. In our case of interest, F, G, f and g are linear functions, and Λ and Π can include integrality restrictions. Bi-level linear programs are NP-hard in general even when no integrality restriction is imposed [Bard 1991].

1.1.2 Uncertain mathematical programs

In practice, most applications involve uncertain data. This is especially true in the fields of planning and scheduling, which by nature are about taking decisions that will be implemented in the future. This is also true for any problem related to a physical system, where the precision of measurements is limited. In some situations, ignoring the fluctuation between the value of the data at the time of decision-making and their actual value leads to significantly suboptimal or infeasible solutions (see for example the instructive numerical study on `NETLIB` problems in [Ben-Tal & Nemirovski 2000]). A variety of paradigms has been proposed since [Dantzig 1955]

to deal with the uncertainty at the modeling stage of the problem solution process. Several ways of integrating uncertainty into the model can be naturally distinguished using the mathematical tool of random variables and their basic characteristics.

Notations In this document, we adopt notations close to those employed in [Birge & Louveaux 2011] or [Shapiro *et al.* 2014]. Unless stated otherwise, we denote by $\tilde{\boldsymbol{\xi}}$ the random vector that reflect the impact of the random events on our data, Ξ its support, and $\boldsymbol{\xi} \in \Xi$ a realization of this random vector. In this context, we redefine function f to incorporate this piece of information: $f : \mathbb{R}^n \times \boldsymbol{\Xi} \to \mathbb{R}$ and $f(\boldsymbol{x}, \boldsymbol{\xi})$ is the value of solution $\boldsymbol{x}$ under uncertainty realization $\boldsymbol{\xi}$. Functions g_i, $i = 1, \ldots, m$ are modified likewise. Moreover, in more explicit models such as (1.1.4)-(1.1.6), we make the dependence of data on random events explicit using a functional notation. For example, $\boldsymbol{c}(\boldsymbol{\xi})$ denotes the cost vector under random realization $\boldsymbol{\xi}$.

We now very briefly introduce the modeling paradigms used in this manuscript for uncertain problems through key modeling components. The topic of uncertain mathematical programs is covered in [Ben-Tal *et al.* 2009, Birge & Louveaux 2011, Shapiro *et al.* 2014, Shapiro 2021].

1.1.2.1 Handling optimality and feasibility

Relying on random variables to express the problem basically extends both objective function and constraints functions (respectively f and $\boldsymbol{g}$ in (1.1.1)-(1.1.3)) to random variable valued-functions. Then, one has to specify the notion of feasibility of a solution (a random vector in this context) as well as of its optimality. We now introduce a few approaches which have been developed for this purpose.

Risk-neutral stochastic programming In this setting, we consider that the constraints must be satisfied with probability one, while the expected value of the objective function must be optimized:

$$\inf \quad \mathbb{E}_{\tilde{\boldsymbol{\xi}}}[f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}})] \tag{1.1.10}$$

$$\text{s.t.} \quad \boldsymbol{g}(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) \leq \boldsymbol{0} \qquad a.s. \tag{1.1.11}$$

$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.12}$$

Optimizing the average value of the objective function makes most sense when the decision prescribed by the model will be used many times once implemented, so that the empirical, observed, cost tends towards its theoretically computed expected value thanks to the Law of Large Numbers (provided the random data incorporated into the decision model reflects the true distribution of data). This approach is risk-neutral in the sense that it mitigates extremely unfavorable and favorable outcomes. It does not distinguish between two solutions providing the same expected objective function, the first one giving only average quality solutions, and the second one giving only very good or very bad decisions depending on the random scenario. This is relevant in applications where a bad realization of uncertainty, or even several in

a row, cannot harm the decision-maker. This is for example the case with very large companies, which are not likely to go bankrupt if the returns of one of their branches is a few percents less than expected for some year.

Regarding feasibility, the solution is required to be feasible for all subsets of realizations of uncertainty except those whose probability measure equals zero. In this manuscript, only discrete scenario-based approaches are developed. In this context, constraints must be satisfied in each non-zero probability scenario.

Risk-averse stochastic programming These approaches are able to capture the variability of the objective function depending on uncertainty realizations. The information about the random variable representing the objective function is summarized into a single real value using a *risk measure* $\mathbb{M}$:

$$\inf \quad \mathbb{M}_{\tilde{\boldsymbol{\xi}}}[f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}})] \tag{1.1.13}$$

$$\text{s.t.} \quad \boldsymbol{g}(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) \leq \mathbf{0} \qquad a.s. \tag{1.1.14}$$

$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.15}$$

Risk measure $\mathbb{M}$ takes various forms in the literature, each with a specific meaning, and theoretical as well as numerical advantages and shortcomings. In the finance literature, the concept of *coherent risk measures* [Artzner *et al.* 1999] is developed, to characterize their desired properties in practice. We refer to [Shapiro *et al.* 2014] for a thorough discussion of those properties, frequently encountered measures and their relations with other ways of coping with uncertainty in decision problems.

In this document, we only present from this rich theoretical background the coherent and easily linearized measure Conditional Value-at-Risk **CVaR**. It is most conveniently described using the non-coherent, non-linear, risk measure Value-at-Risk. Given a probability threshold $\alpha \in [0, 1]$, the value at risk α of a random variable with continuous probability distribution $\tilde{X}$ is $\mathbf{VaR}_\alpha(\tilde{X})$ is:

$$\mathbf{VaR}_\alpha(\tilde{X}) = \inf \left\{ t : \mathbb{P}(\tilde{X} \leq t) \geq 1 - \alpha \right\}$$

The value at risk 1% associated with the cost of a solution $\boldsymbol{x}$, $\mathbf{VaR}_{0.01}(f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}))$, can be understood as the worst possible cost of solution $\boldsymbol{x}$ in the 99% most favorable scenarios. This risk measure is by nature based on a probabilistic constraint, which are in general very hard to handle when solving optimization problems. Moreover, it is not a coherent measure as defined by [Artzner *et al.* 1999], which may render its use in practice irrelevant. The conditional value at risk α of a random variable with continuous probability distribution $\tilde{X}$ is defined as:

$$\mathbf{CVaR}_\alpha(\tilde{X}) = \mathbb{E}\left[\tilde{X} | \tilde{X} \geq \mathbf{VaR}_\alpha(\tilde{X}) \right]$$

The conditional value at risk 1% associated with the cost of a solution $\boldsymbol{x}$, $\mathbf{CVaR}_{0.01}(f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}))$, can be understood as the average cost of solution $\boldsymbol{x}$ in the 1%

least favorable scenarios. It has been proven in [Pflug 2000] that **CVaR** is a coherent risk measure, and [Rockafellar & Uryasev 2000] show that it can be alternatively expressed as:

$$\mathbf{CVaR}_\alpha(\tilde{X}) = \inf_t \left\{ t + \frac{1}{1-\alpha}\mathbb{E}\left[(\tilde{X} - t)^+\right] \right\}$$

Computing this value can then be done with help of a linear program, merely by linearizing the expression $(\tilde{X} - t)^+$.

Robust optimization Introduced by [Soyster 1973], robust optimization is initially oriented towards feasibility: it aims at finding solutions that are immune to infeasibility when some parameters of the model deviate from their nominal values. It ignores probability distributions and requires only the definition of the so-called *uncertainty set*, which contains all the plausible values of the uncertain parameters and can be assimilated to Ξ, the support of the random variables. The constraints must be satisfied for all elements of Ξ. The idea is extended to the objective function, by modeling it as $\min t$ subject to the additional constraints $t \geq f(\boldsymbol{x}, \boldsymbol{\xi})\ \forall \boldsymbol{\xi} \in \Xi$. This can be seen as a special case of stochastic programming where the risk measure is the worst-case value: $\mathbb{M}_{\tilde{X}} = \sup\{X : X \in \text{support}(\tilde{X})\}$. As such, in applications where the effects of uncertainty can be catastrophic, robust optimization presents itself as a viable modeling approach. Robust mathematical programming models take the form:

$$\inf \quad \sup_{\boldsymbol{\xi} \in \Xi} f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) \tag{1.1.16}$$

$$\text{s.t.} \quad \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}) \leq \mathbf{0} \qquad \boldsymbol{\xi} \in \Xi \tag{1.1.17}$$

$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.18}$$

This paradigm is, in certain aspects, more attractive than stochastic programming. Since only the support of the random variable is required, it can be used without knowledge of probability distributions or of relevant sets of scenarios. Further, robust optimization models with polyhedral or convex uncertainty sets lead to deterministic equivalent formulations that are often in the same complexity class as their deterministic counterparts (see *e.g.* [Ben-Tal *et al.* 2009] for an in-depth discussion). For these reasons, robust optimization has enjoyed and continues to enjoy a growing attention from the research community. Advances in static robust optimization are presented in [Bertsimas *et al.* 2011] and [Gabrel *et al.* 2014].

Other approaches Assuming that we know the distribution of the random variables, the question of feasibility can be addressed through the concept of probabilistic constraints (or **chance constraints**). [Charnes & Cooper 1959, Ahmed & Shapiro 2008, Birge & Louveaux 2011] It basically consists in replacing constraints $\boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0}$ with $\mathbb{P}(\boldsymbol{g}(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) \leq \mathbf{0}) \geq \alpha$, with $\alpha \in [0, 1]$. This intuitive modeling tool unfortunately leads to very hard mathematical programs in general: the chance-constrained counterpart of linear programs is already $\mathcal{NP}$-hard.

1.1.2.2 Decision stages

Orthogonal to the way optimality and feasibility are handled in uncertain decision problems, the articulation of the decisions and of the revelation of the value of uncertain data is an important feature which one has to determine when translating a real decision process into a formal mathematical description. This is commonly addressed using the concept of decision stages. Once again, the interested reader may refer to [Ben-Tal *et al.* 2009, Birge & Louveaux 2011, Kall & Wallace 1994, Shapiro *et al.* 2014].

Static models In static (or *single-stage*) models (Figure 1.1.1), all decisions must be taken before the actual value of the uncertain parameters is revealed.

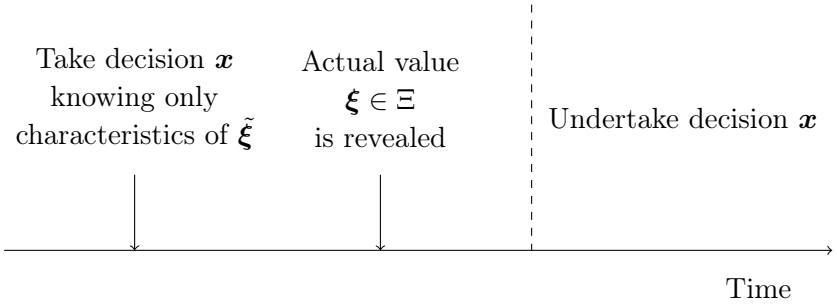


Figure 1.1.1: In static models, the decision-maker must take her decisions before knowing the realization of the uncertain parameters.

Hence, one must choose the vector of decision variables $\boldsymbol{x}$, that must a priori comply with the feasibility restrictions since no action is allowed afterward. Formally, static mixed integer linear stochastic programs take the following form:

$$\min \quad \mathbb{M}_{\tilde{\boldsymbol{\xi}}}\left[\boldsymbol{c}(\tilde{\boldsymbol{\xi}})^\top \boldsymbol{x}\right] \tag{1.1.19}$$

$$\text{s.t.} \quad \mathbf{A}(\tilde{\boldsymbol{\xi}})\boldsymbol{x} \leq \boldsymbol{b}(\tilde{\boldsymbol{\xi}}) \qquad a.s. \tag{1.1.20}$$

$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.21}$$

Because robust optimization assumes no probability distribution, the constraints of static robust mixed integer linear programming models take a slightly different form:

$$\min \quad \max_{\boldsymbol{\xi} \in \Xi} \boldsymbol{c}(\boldsymbol{\xi})^\top \boldsymbol{x} \tag{1.1.22}$$

$$\text{s.t.} \quad \mathbf{A}(\boldsymbol{\xi})\boldsymbol{x} \leq \boldsymbol{b}(\boldsymbol{\xi}) \qquad \forall \xi \in \Xi \tag{1.1.23}$$

$$\boldsymbol{x} \in \mathcal{S} \tag{1.1.24}$$

Notice that in the case of stochastic programs where uncertainty is encoded as a discrete and finite set of scenarios, null-probability scenarios are inherently excluded from Ξ, by definition of the support of random variables. Hence, constraints (1.1.20) can be replaced by (1.1.23).

The fact that $\boldsymbol{x}$ must be completely determined from the start has an interesting consequence: in static models, the uncertainty can be considered independently in each constraint (this is discussed in [Ben-Tal & Nemirovski 1999] for example). This leads to the convenient result that constraints (1.1.23) can be reformulated as:

$$\begin{aligned} & \mathbf{A}_i(\boldsymbol{\xi})\boldsymbol{x} \leq b_i(\boldsymbol{\xi}) && \forall i, \xi \in \Xi \\ \Leftrightarrow & \max_{\boldsymbol{\xi} \in \Xi} \{\mathbf{A}_i(\boldsymbol{\xi})\boldsymbol{x} - b_i(\boldsymbol{\xi})\} \leq 0 && \forall i \end{aligned} \tag{1.1.23'}$$

Then, it is possible to obtain an equivalent model by independently rewriting each constraint with help of various techniques, such as the ones introduced in [Ben-Tal & Nemirovski 1999] for convex uncertainty sets, of which polyhedral uncertainty set [Bertsimas & Sim 2004] is a special case.

Unfortunately, this also formally shows that static models are not suitable for many real situations. There are many applications where such models artificially multiply the variability of the parameters, leading to excessively conservative solutions or infeasible programs. To remedy this issue, one might turn to distributionally robust models [Goh & Sim 2010]. When the underlying application permits it, one might also consider introducing recourse (adjustability/adaptability) in a second decision stage that occurs after the realization of uncertainty.

Two-stage models In two-stage models, decisions are partitioned into first-stage variables, whose value must be fixed before knowing the realization of the uncertainty, and second-stage (or *recourse*) decisions that complete the solution based on full knowledge of the data in order to determine all required decisions, recover feasibility and/or evaluate the quality of the solution after random events occur.

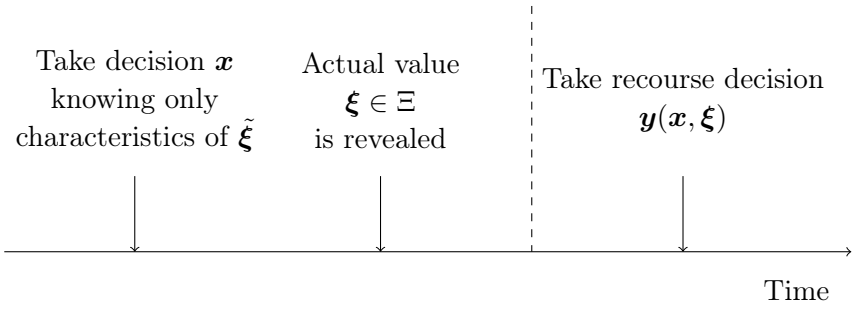


Figure 1.1.2: In two-stage models, the decision-maker first takes planning decisions. Then, the value of uncertain parameters is revealed. Finally, recourse decisions can be taken to complete the solution.

The case of two-stage stochastic linear programs was identified as especially important in practice, as soon as a few years after the development of efficient methods for linear programming. In [Beale 1955, Dantzig 1955], one can find early occurrences of formulations for two-stage and multi-stage programs very close to the ones considered as standard nowadays [Birge & Louveaux 2011, Shapiro *et al.* 2014]:

$$\min \quad \mathbb{M}_{\tilde{\boldsymbol{\xi}}}\left[\boldsymbol{c}^\top \boldsymbol{x} + \min \boldsymbol{q}(\tilde{\boldsymbol{\xi}})\boldsymbol{y}(\tilde{\boldsymbol{\xi}})\right] \tag{1.1.25}$$

$$s.t. \quad \mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1.26}$$

$$\mathbf{T}(\tilde{\boldsymbol{\xi}})\boldsymbol{x} + \mathbf{W}(\tilde{\boldsymbol{\xi}})\boldsymbol{y}(\tilde{\boldsymbol{\xi}}) = \boldsymbol{h}(\tilde{\boldsymbol{\xi}}) \qquad a.s. \tag{1.1.27}$$

$$\boldsymbol{x} \in \mathcal{S}_{\boldsymbol{x}}, \boldsymbol{y}(\tilde{\boldsymbol{\xi}}) \in \mathcal{S}_{\boldsymbol{y}} \tag{1.1.28}$$

This model makes the dependence of second-stage variables to the uncertain parameters explicit using the functional notation: $\boldsymbol{y}(\boldsymbol{\xi})$ is the value of the recourse decision vector under revealed parameters $\boldsymbol{\xi}$. Section 1.2 presents classical techniques to handle a large number of scenarios in two-stage stochastic MILP settings.

The same kind of models can represent two-stage robust problems, also called *adjustable* [Ben-Tal *et al.* 2004], *adaptable* [Bertsimas & Caramanis 2010, Hanasusanto *et al.* 2015], *adaptive* [Bertsimas *et al.* 2013b], *with recourse* [Thiele *et al.* 2009] or *recoverable* [Liebchen *et al.* 2009]:

$$\min \quad t \tag{1.1.29}$$

$$s.t. \quad t \geq \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{q}(\boldsymbol{\xi})\boldsymbol{y}(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi \tag{1.1.30}$$

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{1.1.31}$$

$$\mathbf{T}(\boldsymbol{\xi})\boldsymbol{x} + \mathbf{W}(\boldsymbol{\xi})\boldsymbol{y}(\boldsymbol{\xi}) = \boldsymbol{h}(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi \tag{1.1.32}$$

$$\boldsymbol{x} \in \mathcal{S}_{\boldsymbol{x}} \tag{1.1.33}$$

$$\boldsymbol{y}(\boldsymbol{\xi}) \in \mathcal{S}_{\boldsymbol{y}} \qquad \forall \boldsymbol{\xi} \in \Xi \tag{1.1.34}$$

Introducing additional notations for the feasible sets $\mathcal{X} = \{\boldsymbol{x} \in \mathcal{S}_{\boldsymbol{x}} : \mathbf{A}\boldsymbol{x} = \boldsymbol{b}\}$ and $\mathcal{Y}(\boldsymbol{x}, \boldsymbol{\xi}) = \{\boldsymbol{y} \in \mathcal{S}_{\boldsymbol{y}} : \mathbf{W}(\boldsymbol{\xi})\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\boldsymbol{x}\}$, these problems can also be cast more compactly:

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x},\boldsymbol{\xi})} \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{q}(\boldsymbol{\xi})\boldsymbol{y} \tag{1.1.35}$$

The difficulty of these problems has long been established in the literature even in the simple case of two-stage adjustable robust optimization with linear programming problems in both stages, and a polyhedral uncertainty set (see [Ben-Tal *et al.* 2004]).

1.2 Relaxation, reformulation and decomposition

The research presented in this manuscript is mostly about hard combinatorial optimization problems. This section first motivates our use of relaxation, decomposition and reformulation approaches to deal with them. Then it recalls classical techniques in MILP. We also present a high-level description of various algorithms used in the remainder of the manuscript to obtain (near-)optimal feasible solutions from the iterative solution of relaxations.

1.2.1 Motivation

The different approaches described in this document to tackle $\mathcal{NP}$-hard problems are based on *bounding problems*, which are approximations that can be solved efficiently. This can be done by imposing new restrictions, leading to heuristic algorithms providing *primal bounds*. For example, imposing the sequence of jobs in machine scheduling problems yields (often polynomial time-) list algorithms. Another way to approximate hard problems is to remove some constraints, leading to *relaxations*, providing *dual bounds*. These bounds are iteratively refined using various schemes until they coincide (or the duality gap is sufficiently small), proving the (near-)optimality (see Section 1.2.4) of the best known primal bound.

Let us state the precise definition of the term relaxation, which has entered the folklore of combinatorial optimization, that is used in this manuscript. It is adapted from [Wolsey 2020], with our notations.

Definition 1.2.1 (Relaxation of a combinatorial optimization problem)**.** *A problem* $(RP) : \min_{\boldsymbol{x}\in\mathcal{X}_R} f_R(\boldsymbol{x})$ *is a relaxation of problem* $(P) : \min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$ *if :*

(i) $\mathcal{X} \subseteq \mathcal{X}_R$*, and*

(ii) $f_R(\boldsymbol{x}) \leq f(\boldsymbol{x})$ *for all* $\boldsymbol{x} \in \mathcal{X}$*.*

The optimal value of (RP) *is a dual bound for problem* (P)*:* $Opt(RP) \leq Opt(P)$*.*

The linear relaxation The most well-known and probably the most used relaxation when solving MILP models is the linear relaxation, which is obtained by merely dropping all integrality restrictions of the decision variables. The resulting approximation can then be solved using variations of simplex or interior point algorithms (see *e.g.* [Vanderbei 2020] for both an overview and details on this topic).

Definition 1.2.2 (Linear relaxation of a mixed integer set)**.** *Let us consider the mixed integer set* $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^n : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}\} \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})$*. The linear relaxation of* $\mathcal{X}$*, denoted by* $\overline{\mathcal{X}}$ *in this document, is defined as* $\overline{\mathcal{X}} = \{\boldsymbol{x} \in \mathbb{R}^n : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}\}$*.*

Definition 1.2.3 (Linear relaxation of a mixed integer linear program)**.** *Let us consider the MILP model* $(P) : \min\{\boldsymbol{c}^\top\boldsymbol{x} : \boldsymbol{x} \in \mathcal{X}\}$*, with* $\mathcal{X} = \{\boldsymbol{x} : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}$*. The* linear relaxation *of* (P)*, which is denoted by* $\overline{(P)}$*, is defined as* $\overline{(P)} : \min\{\boldsymbol{c}^\top\boldsymbol{x} : \boldsymbol{x} \in \overline{\mathcal{X}}\}$*.*

Definition 1.2.4 (Strength of a relaxation)**.** *Let* (RP_1) *and* (RP_2) *be two relaxations of problem* (P)*. Relaxation* (RP_1) *is said to be* stronger *than* (RP_2) *if and only if* $Opt(RP_1) > Opt(RP_2)$*.*

Remark 1.2.1 (Optimality of a relaxed solution)**.** *When a relaxation* (RP) *has the same objective function as the original optimization problem* (P) *(i.e.* $f_R = f$*), any optimal solution solution of* (RP) *that is feasible for* (P) *is also optimal for* (P)*. This does not hold in general when* $f_R \neq f$*.*

Practical difficulty of MILP When attempting to solve one of the types of problems described in Section 1.1 with classical integer linear programming approaches, one usually encounters at least one of the following issues:

Issue 1 Poor relaxation: assuming an integer linear formulation is available for the problem, the global solution method suffers from poor dual bounds.

Issue 2 Severely non-linear and non-convex formulations (e.g. bi-level or two-stage robust programs).

Issue 3 Large-scale mathematical programs: this typically arises in the context of stochastic programming with a large number of realizations, robust optimization with infinitely many constraints and/or variables, large-scale deterministic problems that are common in practical applications.

1.2.1.1 Why reformulate?

In this document, reformulation techniques are used to achieve two objectives. First, we design deterministic MILP equivalent formulations of two-stage uncertain problems (see Section 3.1 or Section 3.2), hence we benefit from the abundant MILP toolbox to propose solution approaches. Second, the linear relaxation is used as a basis of many branch-and-X type algorithms (generic branch-and-bound, branch-and-cut, dedicated or generic branch-and-price...). Very efficient commercial codes that implement LP-based branch-and-cut procedures are now available and easy to use, even for practitioners that are not optimization experts. Having a model with strong linear relaxation on hand can drastically change the practical solution of a combinatorial optimization problem through these means. This is one of the main motivations for reformulation techniques: in order to address **Issue 1**, we seek for stronger formulations based on the following ideas.

Ideal formulation of an MILP problem Let us consider the MILP problem (1.1.4)-(1.1.6), that we recall here:

$$(P) : \min \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X} \right\}, \text{ with } \mathcal{X} = \left\{ \boldsymbol{x} : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \right\}.$$

Since the objective function of (P) is convex, at least one of its optimal solutions is an extreme point of the convex hull of $\mathcal{X}$. Hence, this problem is equivalent to $\min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \text{conv } \mathcal{X}\}$. Assuming that $\mathbf{A}$ and $\boldsymbol{b}$ are rational, conv $\mathcal{X}$ is a polyhedron [Meyer 1974]. Thus, (P) is equivalent to the Linear Programming (LP) model:

$$(LP_P^*) : \min \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \mathbf{A}^* \boldsymbol{x} \leq \boldsymbol{b}^*, \boldsymbol{x} \in \mathbb{R}^n \right\}$$

with $\mathbf{A}^*$ and $\boldsymbol{b}^*$ chosen such that $\{\boldsymbol{x} : \mathbf{A}^*\boldsymbol{x} \leq \boldsymbol{b}^*, \boldsymbol{x} \in \mathbb{R}^n\} = \text{conv } \mathcal{X}$.

So, any rational MILP problem can virtually be solved through an equivalent LP problem. Unfortunately explicitly determining the value of $\mathbf{A}^*$ and $\boldsymbol{b}^*$ is $\mathcal{NP}$-hard when (P) is $\mathcal{NP}$-hard (as a consequence of the equivalence of separation and optimization [Grötschel *et al.* 1981]).

Exploiting ideal formulations of subsystems Even if trying to obtain an ideal formulation of the whole problem might not be helpful in practice, we can sometimes identify remarkable subsystems for which we are able to *easily* (not necessarily in polynomial time) write the ideal LP formulation, leading to a new, stronger formulation. Formally, consider a mixed-integer set with the "natural" formulation $\mathcal{Y} = \overline{\mathcal{Y}} \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'})$ with $\overline{\mathcal{Y}} = \{\boldsymbol{y} \in \mathbb{R}^{n'} : \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}\}$, and the ideal formulation $\mathcal{Y}^* = \overline{\mathcal{Y}^*} \cap (\mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'})$ with $\overline{\mathcal{Y}^*} = \{\boldsymbol{y} \in \mathbb{R}^{n'} : \mathbf{D}^*\boldsymbol{y} \leq \boldsymbol{d}^*\}$. Clearly, we have $\mathcal{Y} = \mathcal{Y}^*$, conv $\mathcal{Y}$ = conv $\mathcal{Y}^* = \overline{\mathcal{Y}^*}$, and $\overline{\mathcal{Y}^*} \subseteq \overline{\mathcal{Y}}$ with, in general, $\overline{\mathcal{Y}^*} \neq \overline{\mathcal{Y}}$.

Now let us consider the following combinatorial optimization problem:

$$(CO) : \min \left\{ \boldsymbol{c}_{\boldsymbol{x}}^\top \boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^\top \boldsymbol{y} : \mathbf{B}_{\boldsymbol{x}} \boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}} \boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y} \right\} \tag{1.2.1}$$

Using formulation $\mathcal{Y}$ yields a linear relaxation whose feasible set is:

$$\left\{ (\boldsymbol{x}, \boldsymbol{y}) : \mathbf{B}_{\boldsymbol{x}} \boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}} \boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{x} \in \overline{\mathcal{X}}, \boldsymbol{y} \in \overline{\mathcal{Y}} \right\}.$$

Using formulation $\mathcal{Y}^*$ it becomes:

$$\left\{ (\boldsymbol{x}, \boldsymbol{y}) : \mathbf{B}_{\boldsymbol{x}} \boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}} \boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{x} \in \overline{\mathcal{X}}, \boldsymbol{y} \in \text{conv } \mathcal{Y} \right\}, \tag{1.2.2}$$

which makes the linear relaxation strictly stronger in most cases. Section 1.2.2 shows how to systematically obtain the ideal formulation of some subsystems using extended formulations. Although this seems to worsen **Issue 3** at first sight, the trade-off between size and strength of the formulations is sometimes beneficial to large models. Moreover, some of these techniques come with the possibility of dynamically generating the resulting formulations through column and row generation algorithms, mitigating the effect of increasing the size of the formulation.

1.2.1.2 Why decompose?

We see decomposition techniques as algorithmic strategies that exploit the independence of some subsystems, or the loose connection between them. A natural decomposition emerges between the two levels in bi-level problems, the two stages in the stochastic or robust settings and between the min and max problems in static robust context. In many practical contexts, reasonable size MILP models are appropriate for the first stage alone, and for each of the scenarios of the second stage. However, when considering them together as a whole, they often exhibit **Issue 2** or **Issue 3**. Decomposition approaches permit treating them separately, thus benefiting from their simple structure, the understanding of it and the tools exploiting it.

Many deterministic problems exhibit a decomposable structure too: an appropriate partition of the set of variables (some possibly artificially introduced) reveals a set of *easy* constraints and a set of *complicating* constraints. Easy constraints form a well-structured subsystem in the sense that, when complicating constraints are ignored, effective algorithms exist for its solution. Decomposition techniques

allow the computation of dual bounds by iteratively solving these easy subproblems (Lagrange relaxation), or by dynamically generating their ideal formulation (Dantzig-Wolfe decomposition). Constraint generation methods, such as Benders decomposition, replace their natural set of variables and constraints with a (usually very large) set of constraints that are iteratively determined and added by solving these subproblems.

1.2.2 Classical mixed integer linear programming reformulations

This section is a short reminder of some reformulation techniques commonly used in mixed integer linear programming. Along with Dantzig-Wolfe (DW) and Benders reformulations, we describe the less well-identified modeling tool that consists in rewriting a sequential decision process as a linear program. Both an overall view and details about these and other decomposition approaches can be found in the comprehensive paper [Vanderbeck & Wolsey 2010].

1.2.2.1 Reformulation of sequential decision processes

Some problems are more easily described using the concept of state of the system, actions applicable from a state, and the state resulting by performing a specific action from a specific state. Dynamic Programming (DP) [Bellman 1954] is widely used to tackle such deterministic decision processes, particularly in the field of production planning and scheduling. This name identifies both the paradigm used to model problems through recurrence equations, and the algorithms and algorithmic strategies employed to solve them (memoization, label setting, label correcting, iterative state space algorithms (see 1.2.4.4))... Such routines can be used in a straightforward way to solve relatively stylized problems (see a few examples in scheduling [Brucker 2004b, Brucker 2004a, Pinedo 2012], production planing [Pochet & Wolsey 2006], packing [Dowsland & Dowsland 1992]...). However, when the sequential decision process is part of a larger system, they cannot be directly used.

Some dynamic programs have an LP formulation, whose extreme point solutions satisfy possible integrality requirements. These systems can be integrated as subsystems of MILP models encompassing the whole problem. This helps solve **Issue 1** by providing an ideal formulation for those subsystems (see Section 1.2.1.1). On the other hand, doing so typically worsens **Issue 3**. Solution methods can integrate algorithmic strategies, such as iterative state space methods, to dynamically manage the size of the model [Ibaraki 1987]. These approaches become more and more popular with the revival of *decision diagrams* [Bryant 1986, Hooker 2013]. Note that the resurgence of decision diagrams being mostly posterior to our use of such techniques, we do not really relate them to our work, although it is clear that it could have benefited from the recent results in this area. In this document, we merely stick to our initial view, based on more general DP. The relation between DP and decision diagrams is discussed in [Hooker 2013]. The interested reader can find more formalism and details about DP in [Karp & Held 1967] (that connects it

to automatons and grammars).

A class of simple dynamic programs In order to stay concise and only introduce what is necessary for the rest of this document, we focus on a special class of DP models. Note however that what we expose below can be generalized to some extent using among other things hypergraphs instead of graphs (see [Martin *et al.* 1990] for theoretic results, and *e.g.* [Clautiaux *et al.* 2018] for a real-life application of a 2D packing problem). Introducing stochasticity leads to considering Markov Decision Processes, for which [Guillot & Stauffer 2020], among other things, generalize some ideas exposed below.

The dynamic programs we consider can be defined by a tuple $\mathcal{M} = (Q_\perp, \Sigma, \delta, q_0, q_\bullet, c)$, where $Q_\perp = Q \cup \{\perp\}$ is a finite set of *states* (also referred to as the state space), Σ a finite set of actions or *transitions*, δ a transition function defined from $Q_\perp \times \Sigma$ to $Q_\perp$, $q_0 \in Q$ an initial state, and $q_\bullet \in Q$ a final state. Symbol $\perp \in Q_\perp$ denotes a non-final state that gathers infeasible sequences. Specifically, we have $\delta(\perp, a) = \perp$ for every transition $a \in \Sigma$ (i.e.,$\perp$ is an absorbing state) and $\delta(q, a) = \perp$ if transition a is not applicable to $q \in Q$. The cost function c is defined from $Q_\perp \times \Sigma$ to $\mathbb{R} \cup \{+\infty\}$. Without loss of generality, we assume that $c(q, a) = +\infty$ for every $q \in Q_\perp$ and every $a \in \Sigma$ such that $\delta(q, a) = \perp$. Let $\overrightarrow{\Phi} : Q_\perp \to \mathbb{R} \cup \{+\infty\}$ be the function that associates to each state $q \in Q_\perp$ the best cost to switch from state q_0 to q. The DP problem (Λ) is to compute the value of $\overrightarrow{\Phi}(q_\bullet)$, the functional equations of Λ being written as:

$$\overrightarrow{\Phi}(q_0) = 0 \tag{1.2.3}$$

$$\overrightarrow{\Phi}(q) = \inf \left\{ \overrightarrow{\Phi}(q') + c(q', a) |\ q' \in Q, a \in \Sigma, \delta(q', a) = q \right\} \quad \forall q \in Q - \{q_0\} \tag{1.2.4}$$

Formulation as a shortest path problem First, we introduce the transition graph $G_\mathcal{M} = (V_\mathcal{M}, A_\mathcal{M})$ associated with $\mathcal{M}$. Graph $G_\mathcal{M}$ is a weighted directed acyclic multigraph. Each node in $V_\mathcal{M}$ represents a state of the DP. Let us denote $q(v) \in Q$ the state associated with node $v \in V_\mathcal{M}$ and $v(q)$ the node associated with a state $q \in Q$. Each arc of the graph corresponds to applying an action from a particular state. Specifically, for each state $q \in Q$ and each transition $a \in \Sigma$ such that $\delta(q, a) \neq \perp$, the graph contains an arc in $A_\mathcal{M}$ labeled with a weight $c(q, a)$. We have $A_\mathcal{M} = \{(v(q), v(\delta(q, a)), c(q, a)) : q \in Q \wedge a \in \Sigma \wedge \delta(q, a) \neq \perp\}$. For each arc $\alpha \in A_\mathcal{M}$, we denote $t(\alpha)$, $h(\alpha)$ and $c(\alpha)$ its tail, head, and cost, respectively. Solving (Λ) is equivalent to computing a shortest path in $G_\mathcal{M}$ from q_0 to $q_\bullet$.

Formulation as a linear program This shortest path problem admits a network flow LP model. Associating decision variables φ_α to the arcs in multidigraph $G_\mathcal{M}$

yields the following arc-flow MILP formulation of (Λ), denoted (F_Λ):

$$(F_\Lambda): \min \sum_{\alpha \in A_{\mathcal{M}}} c(\alpha)\varphi_\alpha \tag{1.2.5}$$

$$\text{s.t.} \sum_{\alpha \in A_{\mathcal{M}}: h(\alpha)=v} \varphi_\alpha - \sum_{\alpha \in A_{\mathcal{M}}: t(\alpha)=v} \varphi_\alpha = \begin{cases} 1 & \text{if } q(v) = q_0 \\ 0 & \text{if } q(v) \in Q \setminus \{q_0, q_\bullet\} \\ -1 & \text{if } q(v) = q_\bullet \end{cases} \quad v \in V_{\mathcal{M}} \tag{1.2.6}$$

$$\varphi_\alpha \in [0,1] \quad \alpha \in A_{\mathcal{M}}. \tag{1.2.7}$$

The objective function (1.2.5) minimizes the total cost to reach state $q_\bullet$ from q_0. Constraints (1.2.6) are the flow conservation constraints. Finally, constraints (1.2.7) define the domain of the decision variables. It is crucial to remark that (1.2.6), (1.2.7) being network flow constraints with integer right-hand-side, extreme point solutions of (F_Λ) are integer.

Reformulation of a subsystem Let us consider a combinatorial optimization problem where a subset of variables $\boldsymbol{y}$ must take their value so that they follow the sequential decision process defined by DP (Λ). More precisely, for all $a \in \Sigma$, variable y_a is equal to the number of times action a is selected in the sequence of decisions, and $\mathcal{Y}$ is the set of feasible vectors $\boldsymbol{y}$ according to (Λ). For the sake of conciseness, let us rewrite (F_Λ) using the following matrix notation: $(F_\Lambda) : \min\{\boldsymbol{c}_{\boldsymbol{\varphi}}^\top \boldsymbol{\varphi} : \boldsymbol{\Phi}\boldsymbol{\varphi} = \boldsymbol{b}_\Lambda, \boldsymbol{\varphi} \geq \boldsymbol{0}\}$. Moreover, let us denote by $\mathbf{A}^\Lambda$ the matrix such that $\mathbf{A}^\Lambda_{i,\alpha} = 1$ if arc $\alpha \in \mathcal{A}$ corresponds to applying action $i \in \Sigma$, and 0 otherwise. Then we have the following ideal formulation for $\mathcal{Y}$:

$$\boldsymbol{y} \in \operatorname{conv} \mathcal{Y} \quad \Leftrightarrow \quad \boldsymbol{y} \in \left\{\mathbf{A}^\Lambda \boldsymbol{\varphi} : \boldsymbol{\Phi}\boldsymbol{\varphi} = \boldsymbol{b}_\Lambda, \boldsymbol{\varphi} \geq \boldsymbol{0}\right\}.$$

This expression can be plugged into the model (1.2.1) by either replacing $\boldsymbol{y}$ (1.2.8) or by replacing the constraints $\boldsymbol{y} \in \mathcal{Y}$ (1.2.9), or a mix of the two approaches.

$$(CO_{DP1}) : \min \Big\{ \boldsymbol{c}_{\boldsymbol{x}}^\top \boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^\top \mathbf{A}^\Lambda \boldsymbol{\varphi} : \mathbf{B}_{\boldsymbol{x}} \boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}} \mathbf{A}^\Lambda \boldsymbol{\varphi} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{X}, \\ \boldsymbol{\Phi}\boldsymbol{\varphi} = \boldsymbol{b}_\Lambda, \boldsymbol{\varphi} \geq \boldsymbol{0}, \mathbf{A}^\Lambda \boldsymbol{\varphi} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'} \Big\} \tag{1.2.8}$$

$$(CO_{DP_2}) : \min \Big\{ \boldsymbol{c}_{\boldsymbol{x}}^\top \boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^\top \boldsymbol{y} : \mathbf{B}_{\boldsymbol{x}} \boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}} \boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{X}, \\ \boldsymbol{y} = \mathbf{A}^\Lambda \boldsymbol{\varphi}, \boldsymbol{\Phi}\boldsymbol{\varphi} = \boldsymbol{b}_\Lambda, \boldsymbol{\varphi} \geq \boldsymbol{0}, \boldsymbol{y} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'} \Big\} \tag{1.2.9}$$

Directly including $\boldsymbol{\varphi}$ in the objective function as in (1.2.8) is helpful when the cost of using an action depends on the state it is applied from. Keeping variables $\boldsymbol{y}$ in the model allows branching on them in a branch-and-X algorithm. The drawback of this approach is that the number of $\boldsymbol{\varphi}$-variables is equal to the number of transitions in Λ, while the number of flow conservation constraints is approximately equal to its number of states.

Work with DP-based (re)formulations Some of the work presented in this manuscript rely on a Dynamic Programming formulation, in particular those based on State-Space Relaxation (SSR) techniques (see Sections 1.2.3.3 and 1.2.4.4), where the models at the heart of the solution approach are shortest path reformulations directly derived from a DP. [Detienne *et al.* 2012, Tanaka *et al.* 2015, Detienne *et al.* 2016] describe applications of SSR for single machine, jobshop and flowshop scheduling (Section 2.1), respectively, and [Clautiaux *et al.* 2021] describes an application for a generalization of the knapsack problem (Section 2.2). The primary motivation is to obtain strong dual bounds (**Issue 1**).

In [Griset *et al.* 2021], the possible schedules of power plants are modeled through a state/transition graph embedding various complex constraints. With the purpose of coping with **Issue 1** and **Issue 2**, we use this graph to build a MILP model on top of which we design our methods (Section 3.1).

Facing **Issue 2** in [Arslan & Detienne 2021], we derive deterministic MILP formulations for two-stage robust integer programs, which rely on the convexification of the second stage feasible set. When the application possesses the appropriate structure, the DP-based reformulation is a possible way of performing this convexification (Section 3.2).

1.2.2.2 Dantzig-Wolfe reformulation/decomposition

The classical *Dantzig-Wolfe reformulation* [Dantzig & Wolfe 1960, Vanderbeck 2000] helps solve **Issue 1** by replacing a specific subsystem with its ideal formulation (see Section 1.2.1.1). Two close but different approaches are described in the literature: [Wolsey & Nemhauser 1999, Vanderbeck 2000, Lübbecke & Desrosiers 2005] present the *discretization* approach, for integer subsystems and the *convexification* approach, that applies also for mixed integer sets. The difference lays in the way the integrality requirements on the subsystem solutions are handled.

We first focus on the convexification approach, which it is based on the Minkowski-Weyl theorem for polyhedra. Let us once again consider a mixed-integer set with the natural formulation $\mathcal{Y} = \{\boldsymbol{y} : \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}, \boldsymbol{y} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'}\}$. As recalled in Section 1.2.1.1, when $\mathbf{D}$ and $\boldsymbol{d}$ are rational, conv $\mathcal{Y}$ is a polyhedron. Then according to Minkowsky-Weyl theorem, it can be expressed as a convex combination of its N extreme points and conic combination of its N' extreme rays:

$$\text{conv } \mathcal{Y} = \left\{ \mathbf{Y}^v \boldsymbol{\lambda} + \mathbf{Y}^r \boldsymbol{\mu} : \mathbf{1}^\top \boldsymbol{\lambda} = 1, \boldsymbol{\lambda} \in \mathbb{R}^N_+, \boldsymbol{\mu} \geq \mathbb{R}^{N'}_+ \right\}$$

Here, each column of matrix $\mathbf{Y}^v$ (resp. of matrix $\mathbf{Y}^r$) is the vector describing one extreme point (resp. one extreme ray) of $\mathcal{Y}$.

Reformulation of a subsytem Let us instantiate problem (1.2.1) as a MILP:

$$\begin{aligned}(OC_{MILP}): \quad & \min \ \boldsymbol{c}_{\boldsymbol{x}}^{\top}\boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^{\top}\boldsymbol{y} \\ s.t. \quad & \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}}\boldsymbol{y} \leq \boldsymbol{b} \\ & \mathbf{A}\boldsymbol{x} \leq \boldsymbol{a}, \boldsymbol{x} \in \mathbb{Z}^{p} \times \mathbb{R}^{n-p} \\ & \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}, \boldsymbol{y} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'}\end{aligned}$$

The DW reformulation by convexification of this model is obtained by replacing $\boldsymbol{y}$ with its Minkowski-Weyl expression:

$$\begin{aligned}(OC_{DW}): \quad & \min \ \boldsymbol{c}_{\boldsymbol{x}}^{\top}\boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^{\top}\left(\mathbf{Y}^{v}\boldsymbol{\lambda} + \mathbf{Y}^{r}\boldsymbol{\mu}\right) & \\ s.t. \quad & \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}}\left(\mathbf{Y}^{v}\boldsymbol{\lambda} + \mathbf{Y}^{r}\boldsymbol{\mu}\right) \leq \boldsymbol{b} & (1.2.10) \\ & \mathbf{A}\boldsymbol{x} \leq \boldsymbol{a}, \boldsymbol{x} \in \mathbb{Z}^{p} \times \mathbb{R}^{n-p} & (1.2.11) \\ & \mathbf{1}^{\top}\boldsymbol{\lambda} = 1 & (1.2.12) \\ & \boldsymbol{\lambda} \in \mathbb{R}_{+}^{N}, \boldsymbol{\mu} \geq \mathbb{R}_{+}^{N'} & (1.2.13) \\ & \left(\mathbf{Y}^{v}\boldsymbol{\lambda} + \mathbf{Y}^{r}\boldsymbol{\mu}\right) \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'} & (1.2.14)\end{aligned}$$

In case of multiple independent subsystems, the same reformulation approach can be used for each of them separately. However, when there are identical subsystems, enforcing constraint (1.2.14) in a branch-and-X method turns out to be non-trivial. The discretization approach is not appropriate when $\mathcal{Y}$ is not an integer set, but it simplifies the management of the integrality restrictions in practice. The difference is essentially to replace constraint (1.2.14) with $\boldsymbol{\lambda} \in \{0,1\}^{N}, \boldsymbol{\mu} \in \mathbb{N}^{N'}$.

Solution via a decomposition algorithm Obviously, the size of model (OC_{DW}) is usually very large, but finite. The introduction of an exponential number of variables in the formulation would worsen **Issue 3** if algorithmic strategies were not developed to deal with them. The standard way to solve (OC_{DW}) is through a branch-and-price procedure, which solves its linear relaxation and uses branching to enforce the integrality requirements. Branch-and-price algorithms take advantage of the known implicit definition of $\mathbf{Y}^{v}$ and $\mathbf{Y}^{r}$ (*e.g.* the MILP formulation of $\mathcal{Y}$, or *the set of all pseudo-schedules on machine 2...*) to solve $\overline{(OC_{DW})}$ through column generation.

The column generation procedure [Dantzig & Wolfe 1960] is comprised of two elements. The *restricted master program* (RMP) simply is a truncated version of $\overline{(OC_{DW})}$ where only a subset of $\boldsymbol{\lambda}-$ and $\boldsymbol{\mu}-$variables are included. Model (RMP) is iteratively solved and missing variables are integrated into it until one can prove the optimality of the current solution for $\overline{(OC_{DW})}$ without adding more variables. The global procedure can be seen as an implementation of the primal simplex algorithm [Dantzig *et al.* 1955] where finding the next column entering the basis is done in two ways. Given current simplex multipliers, it is first done by inspecting the reduced cost of the variables already in (RMP). This is done until all variables in (RMP) have non-negative reduced costs (that is, (RMP) is solved using the simplex

algorithm). Then the *oracle*, second ingredient of the procedure, is used to look for a negative reduced cost variable in $\overline{(OC_{DW})}$ (the oracle solves the so-called *pricing problem*). According to the definition of $\mathbf{Y}^v$ and $\mathbf{Y}^r$, each $\boldsymbol{\lambda}-$ and $\boldsymbol{\mu}-$variable corresponds to a feasible solution or an extreme ray of $\mathcal{Y}$. Thus, given simplex multipliers $(\boldsymbol{\pi}, \eta)$ associated with constraints (1.2.10) and (1.2.12), respectively, the pricing problem reads:

$$(Pricing(\boldsymbol{\pi}, \eta)): \quad \inf \left\{ \left(\boldsymbol{c_y} - \mathbf{B}_{\boldsymbol{y}}^{\top} \boldsymbol{\pi} \right)^{\top} \boldsymbol{y} - \eta : \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}, \boldsymbol{y} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'} \right\}$$

This decomposition algorithm makes DW reformulation an interesting technique even in some cases when it does not improve the strength of the relaxation (*i.e.* when we already have $\overline{\mathcal{Y}} = \text{conv } \mathcal{Y}$, like in Section 3.1), because it contributes to the practical management of huge models.

The case of independent subsystems The last remark is especially true when $\mathcal{Y}$ is composed of a set K of independent subsystems. In this case, $\mathbf{D}$ is block diagonal, and $\mathcal{Y} = \bigtimes_{k \in K} \mathcal{Y}^k$, with $\mathcal{Y}^k = \{\boldsymbol{y} : \mathbf{D}^k \boldsymbol{y} \leq \boldsymbol{d}^k, \boldsymbol{y} \in \mathbb{Z}^{p'} \times \mathbb{R}^{n'-p'}\}$. This leads to

$$\text{conv } \mathcal{Y} = \bigtimes_{k \in K} \left\{ \mathbf{Y}^{vk} \boldsymbol{\lambda}^k + \mathbf{Y}^{rk} \boldsymbol{\mu}^k : \mathbf{1}^{\top} \boldsymbol{\lambda}^k = 1, \boldsymbol{\lambda}^k \in \mathbb{R}_+^{N_k}, \boldsymbol{\mu}^k \geq \mathbb{R}_+^{N'_k} \right\}$$

The formulation (OC_{DW}) is adapted in consequence. In practice, the pricing problem can be decomposed into one subproblem per subsystem, and various policies can be defined concerning which subproblem(s) should be solved at each iteration. This makes DW decomposition particularly interesting to deal with **problems with a constraint matrix exhibiting a block diagonal structure with a set of linking rows**.

In the vein of the diverse improvements of the algorithmic solution of MILP problems, DW decomposition has seen many developments [Pessoa *et al.* 2018a, Sadykov *et al.* 2019, Bergner *et al.* 2015, Elhallaoui *et al.* 2005] making the approach very effective for more and more classes of problems [Pessoa *et al.* 2020, Bulhões *et al.* 2020].

Work with DW reformulation We describe an application of Dantzig-Wolfe reformulation and column generation on a large-scale real-life problem in [Griset *et al.* 2021] (Section 3.1). This is done in conjunction with a DP based reformulation, in order to cope with **Issue 3** caused by the latter. The technique is also used in [Arslan & Detienne 2021] (Section 3.2) in the two-stage robust setting as a means for convexifying the second stage feasible set, in order to write a deterministic equivalent formulation (**Issue 2**).

1.2.2.3 Benders reformulation/decomposition or L-shaped method

Benders decomposition has been introduced by [Benders 1962] to solve MILP models by decomposing the problem into an integer programming problem and a linear

programming problem. Later, [Van Slyke & Wets 1969] independently described the same procedure, that was called the *L-shaped* method, with the aim of solving two-stage stochastic linear programs and some optimal control problems. From our point of view, Benders decomposition helps dealing with **Issue 3** in the context of two-stage mixed-integer stochastic programming, with an LP as recourse problem.

Reformulation of a subsytem Let us consider the following instance of 1.2.1, where the vector of variables $\boldsymbol{y}$ has no integrality requirement:

$$\begin{aligned}(OC_{LinearRecourse}): \quad & \min \ \boldsymbol{c}_{\boldsymbol{x}}^{\top}\boldsymbol{x} + \boldsymbol{c}_{\boldsymbol{y}}^{\top}\boldsymbol{y} \\ s.t. \quad & \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x} + \mathbf{B}_{\boldsymbol{y}}\boldsymbol{y} \leq \boldsymbol{b} \\ & \mathbf{A}\boldsymbol{x} \leq \boldsymbol{a}, \boldsymbol{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \\ & \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}, \boldsymbol{y} \in \mathbb{R}_+^{n'}\end{aligned}$$

The approach relies on the fact that $\boldsymbol{x}$-variables are *complicating* variables: when their value is fixed, the resulting problem is a simple LP. Note that in many contexts (such as two-stage optimization), this LP is itself composed of independent subsystems. This makes Benders decomposition particularly interesting to deal with problems with a constraint matrix exhibiting a **block diagonal structure with a set of linking columns**. Several approaches are described in the literature, based on variations of the reasoning exposed below (*e.g.* [Van Slyke & Wets 1969] propose an approach based on the phase I simplex auxiliary program for checking the feasibility, [Fischetti *et al.* 2010] modify the subproblem to deal with feasibility and optimality at the same time...), or on convex analysis (see *e.g.* [Shapiro *et al.* 2014] for a derivation in terms of subgradients of the recourse value function). Let us assume that the vector $\boldsymbol{x}$ is fixed (let $\bar{\boldsymbol{x}}$ be its value). Then the remaining optimization problem reads:

$$SP_{Benders}(\bar{\boldsymbol{x}}) : \min \left\{ \boldsymbol{c}_{\boldsymbol{y}}^{\top}\boldsymbol{y} : \mathbf{B}_{\boldsymbol{y}}\boldsymbol{y} \leq \boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\bar{\boldsymbol{x}}, \mathbf{D}\boldsymbol{y} \leq \boldsymbol{d}, \boldsymbol{y} \in \mathbb{R}_+^{n'} \right\}$$

Its dual problem is:

$$DSP_{Benders}(\bar{\boldsymbol{x}}) : \max \left\{ (\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\bar{\boldsymbol{x}})^{\top} \boldsymbol{\pi} + \boldsymbol{d}^{\top}\boldsymbol{\eta} : \mathbf{B}_{\boldsymbol{y}}^{\top}\boldsymbol{\pi} + \mathbf{D}^{\top}\boldsymbol{\eta} \leq \boldsymbol{c}_{\boldsymbol{y}}, \boldsymbol{\pi} \leq \mathbf{0}, \boldsymbol{\eta} \leq \mathbf{0} \right\}$$

The reformulation can be derived by characterizing the feasibility of $SP_{Benders}(\bar{\boldsymbol{x}})$, and its optimal value, based on the value of $\bar{\boldsymbol{x}}$. We remark that the feasibility of $DSP_{Benders}(\bar{\boldsymbol{x}})$ does not depend on the choice of $\bar{\boldsymbol{x}}$. So, either:

(i) $DSP_{Benders}(\bar{\boldsymbol{x}})$ is infeasible for all $\bar{\boldsymbol{x}}$, which implies that $SP_{Benders}(\bar{\boldsymbol{x}})$ is infeasible or unbounded for all $\bar{\boldsymbol{x}}$, and thus $(OC_{LinearRecourse})$ is also infeasible or unbounded ; or

(ii) $SP_{Benders}(\bar{\boldsymbol{x}})$ is infeasible if and only if $DSP_{Benders}(\bar{\boldsymbol{x}})$ is unbounded. In this case, there necessarily exists an extreme ray $(\bar{\boldsymbol{\pi}}, \bar{\boldsymbol{\eta}})$ of $DSP_{Benders}(\bar{\boldsymbol{x}})$ such that $(\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\bar{\boldsymbol{x}})^{\top} \bar{\boldsymbol{\pi}} + \boldsymbol{d}^{\top}\bar{\boldsymbol{\eta}} > 0$.

So, imposing that $\bar{\boldsymbol{x}}$ can be completed with a decision variable vector $\boldsymbol{y}$ to form a feasible solution can be done with help of the set of constraints $(\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x})^\top \bar{\boldsymbol{\pi}}^r + \boldsymbol{d}^\top \bar{\boldsymbol{\eta}}^r \leq 0$ for all $(\bar{\boldsymbol{\pi}}^r, \bar{\boldsymbol{\eta}}^r) \in Q_{Feas}$, where Q_{Feas} denotes the set of (normalized) extreme rays of $\{(\boldsymbol{\pi}, \boldsymbol{\eta}) : \mathbf{B}_{\boldsymbol{y}}^\top \boldsymbol{\pi} + \mathbf{D}^\top \boldsymbol{\eta} \leq \boldsymbol{c}_{\boldsymbol{y}}, \boldsymbol{\pi} \leq \mathbf{0}, \boldsymbol{\eta} \leq \mathbf{0}\}$.

An alternative expression of the optimal value of $SP_{Benders}(\bar{\boldsymbol{x}})$ is obtained through the discretization of its dual problem:

$$DSP^D_{Benders}(\bar{\boldsymbol{x}}) : \max\{(\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\bar{\boldsymbol{x}})^\top \bar{\boldsymbol{\pi}}^q + \boldsymbol{d}^\top \bar{\boldsymbol{\eta}}^q : (\bar{\boldsymbol{\pi}}^q, \bar{\boldsymbol{\eta}}^q) \in Q_{Opt}\}$$

where Q_{Opt} denotes the set of extreme points of $\{(\boldsymbol{\pi}, \boldsymbol{\eta}) : \mathbf{B}_{\boldsymbol{y}}^\top \boldsymbol{\pi} + \mathbf{D}^\top \boldsymbol{\eta} \leq \boldsymbol{c}_{\boldsymbol{y}}, \boldsymbol{\pi} \leq \mathbf{0}, \boldsymbol{\eta} \leq \mathbf{0}\}$. Assuming that $SP_{Benders}(\bar{\boldsymbol{x}})$ is feasible, its optimal value is equal to the optimal value of $DSP^D_{Benders}(\bar{\boldsymbol{x}})$. Hence, it is equal to the optimal value of the LP model $\min\{\theta : \theta \geq (\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\bar{\boldsymbol{x}})^\top \bar{\boldsymbol{\pi}}^q + \boldsymbol{d}^\top \bar{\boldsymbol{\eta}}^q \quad \forall (\bar{\boldsymbol{\pi}}^q, \bar{\boldsymbol{\eta}}^q) \in Q_{Opt}, \theta \in \mathbb{R}\}$.

Finally, putting things together yields the following Benders reformulation of problem $(OC)_{LinearRecourse}$:

$$(OC_{Benders}): \quad \min \ \boldsymbol{c}_{\boldsymbol{x}}^\top \boldsymbol{x} + \theta \tag{1.2.15}$$

$$s.t. \quad \mathbf{A}\boldsymbol{x} \leq \boldsymbol{a}, \boldsymbol{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \tag{1.2.16}$$

$$(\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x})^\top \bar{\boldsymbol{\pi}}^r + \boldsymbol{d}^\top \bar{\boldsymbol{\eta}}^r \leq 0 \qquad (\bar{\boldsymbol{\pi}}^r, \bar{\boldsymbol{\eta}}^r) \in Q_{Feas} \tag{1.2.17}$$

$$\theta \geq (\boldsymbol{b} - \mathbf{B}_{\boldsymbol{x}}\boldsymbol{x})^\top \bar{\boldsymbol{\pi}}^q + \boldsymbol{d}^\top \bar{\boldsymbol{\eta}}^q \qquad (\bar{\boldsymbol{\pi}}^q, \bar{\boldsymbol{\eta}}^q) \in Q_{Opt} \tag{1.2.18}$$

$$\theta \in \mathbb{R} \tag{1.2.19}$$

Constraints (1.2.17) are called *feasibility cuts*, while constraints (1.2.18) are referred to as *optimality cuts*.

Solution via a decomposition algorithm As for the DW reformulation, introducing an exponential (but finite) number of constraints in the model would worsen **Issue 3** if algorithmic strategies were not developed to handle them. The classical Benders decomposition algorithm dynamically generates feasibility and optimality cuts until it can be proven that the set of already integrated constraints is sufficient. The constraint generation algorithm starts by solving a relaxed master program, obtained from $(OC_{Benders})$ by only retaining a subset of Q_{feas} and Q_{opt}. Then, the procedure checks the feasibility of its solution $(\bar{\boldsymbol{x}}, \bar{\theta})$ for the whole problem $(OC_{Benders})$ by solving the subproblem $DSP_{Benders}(\bar{\boldsymbol{x}})$. If the latter is unbounded, then an extreme ray defining a violated feasibility cut is found, which is added to Q_{feas}, and the routine loops with solving the new relaxed master program. If the subproblem is bounded, then its optimal value is compared to its approximation $\bar{\theta}$. If it is higher than $\bar{\theta}$, then the optimality cut associated with its solution is violated. Hence, it is added to the relaxed master program and another iteration of the loop is performed. Otherwise, $(\bar{\boldsymbol{x}}, \bar{\theta})$ satisfies all the constraints of $(OC_{Benders})$, so that its cost is a primal bound on its optimal value. It is also an optimal solution of a relaxation, so its cost (in the relaxed master program) is a dual bound on the optimal value of $(OC_{Benders})$. Since both programs have the same objective function, the primal and dual bounds are equal and $(\bar{\boldsymbol{x}}, \bar{\theta})$ is optimal for $(OC_{Benders})$.

The algorithm converges in a finite number of iterations because Q_{feas} and Q_{opt} are finite and one cut is added at each iteration (except for the last one).

The reformulation exposed above is called *single-cut* Benders reformulation. When $DSP_{Benders}(\bar{\boldsymbol{x}})$ is itself composed of independent subsystems (for example in the context of two-stage stochastic programming), their optimization can be done separately. Moreover, the sets of cuts can be considered separately for each subsystem, leading to the *multi-cut* reformulation. There is no clear superiority in general between the two approaches [Birge & Louveaux 2011], empirical results showing that the best choice is problem-dependent.

The introduction of *lazy constraint callbacks* in MILP solvers paved the way to another algorithm to solve model ($OC_{Benders}$). Instead of solving the whole relaxed master program at each iteration, the *Branch-and-Benders-cut* algorithm [Fortz & Poss 2009] explores a single branch-and-bound tree, adding cuts only after a potential incumbent solution found during the search is proven to be infeasible.

A recurrent critic against the use of Benders decomposition is its tendency to exhibit slow convergence because of zigzagging behavior or tailing-off effect. A lot of work has been devoted in the past decade to improve the practical convergence speed of various algorithms (see *e.g.* [Rahmaniani *et al.* 2017] for a review of the improvements), leading to competitive methods (*e.g.* [Fischetti *et al.* 2017, Bucarey *et al.* 2022]).

Work with Benders reformulation In [Griset *et al.* 2021] (see Section 3.1), we present an application of Benders decomposition for a real-life two-stage stochastic problem, and its combination with Dantzig-Wolfe reformulation. In [Detienne *et al.* 2009], we use Benders decomposition to decompose an employee timetabling problem into shift assignment on one side, and task assignment on the other. In both contexts, we aim at managing large size models with the help of a decomposition algorithm that results from Benders reformulation (**Issue 3**).

From a methodological perspective, we propose in [Blanchot *et al.* Submitted] acceleration techniques for problems with many independent subsystems, that arise in stochastic programs with many uncertain realizations (**Issue 3**).

1.2.3 Lagrange relaxation

Lagrangian duality is a widely used concept in mathematical optimization and its applications (in macro-and micro-economy for example). It has been used in discrete optimization (notably in the field of integer linear programming) since at least the 1960's [Everett 1963, Nemhauser & Ullmann 1968, Held & Karp 1970, Fisher 1973]. The basic theoretical properties are established in [Geoffrion 1974]. For an in-depth presentation of Lagrangian relaxation, we refer to [Lemaréchal 2001].

Our use of Lagrangian relaxation aims at solving **Issue 1**, by defining relaxations that are stronger than the linear relaxation of a considered MILP. We also use it to treat independent subsystems separately and deal with **Issue 3**. It is also at the

heart of iterative state-space relaxation methods, described in Section 1.2.3.3 and Section 1.2.4.4.

We consider mathematical programs of the following type:

$$(P_{Lag}) \ : \ \min \ f(\boldsymbol{x}) \tag{1.2.20}$$
$$\text{s.t. } \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \tag{1.2.21}$$
$$\boldsymbol{x} \in \mathcal{X} \tag{1.2.22}$$

with $\mathcal{X} \subset \mathbb{R}^n$ and $\boldsymbol{g} = (g_i)_{i=1,\dots,m}$, and $g_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \dots, m$. Constraints $\boldsymbol{x} \in \mathcal{X}$ usually define "simple" subproblems, while $\boldsymbol{g}(\boldsymbol{x}) \leq 0$ are "complicating constraints". In this context, removing the latter yields an optimization problem that is easy to solve in practice.

The Lagrangian relaxation consists in removing the complicating constraints and penalizing their violation using a penalty term in the objective function (*i.e. dualizing* them). This term is a weighted sum of the violations of individual constraints. Given fixed penalties (so-called Lagrangian multipliers), solving the simple subproblems (with modified objective) yields a dual bound for the optimal value of the original problem. In order to find the best such dual bound, the Lagrangian multipliers need to be optimized, resulting in what is called the Lagrangian dual problem.

Formally, the Lagrangian $\mathscr{L} : \mathbb{R}^n \times \mathbb{R}^m_+ \to \mathbb{R}$ is the function defined by:

$$\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}).$$

We define the Lagrangian dual function $L : \mathbb{R}^n \to \mathbb{R}$ as:

$$L(\boldsymbol{\lambda}) = \min_{\boldsymbol{x} \in \mathcal{X}} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}).$$

The problem of computing $L(\boldsymbol{\lambda})$ is often referred to as the *Lagrangian subproblem* $(R_{Lag}(\boldsymbol{\lambda})) : \min_{\boldsymbol{x} \in \mathcal{X}} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda})$.

The Lagrangian dual problem is:

$$(D_{Lag}) : \sup_{\boldsymbol{\lambda} \in \mathbb{R}^m_+} L(\boldsymbol{\lambda}).$$

1.2.3.1 Basic properties

In practice, one of the main interests of the Lagrangian dual function is the weak duality property: the value of the Lagrangian dual function at any feasible point $\boldsymbol{\lambda}$ is a lower bound for (P_{Lag}).

Therorem 1.2.1 (Weak duality theorem)**.** *Let $\boldsymbol{x}$ be a feasible solution of the primal problem (P_{Lag}), and let $\boldsymbol{\lambda}$ be a feasible solution of the dual problem (D_{Lag}). Then $f(\boldsymbol{x}) \geq L(\boldsymbol{\lambda})$.*

So, for any value of $\boldsymbol{\lambda} \in \mathbb{R}^m_+$, the problem $(R_{Lag})(\boldsymbol{\lambda})$ is a relaxation of (P_{Lag}). In order to build effective solution methods, one would like to get the best possible

relaxation, by solving problem (D_{Lag}). This turns out to be a convex optimization problem, for which we are able to obtain relevant information (subgradients) about L as a by-product of its computation.

Proposition 1.2.2. *The dual function L is concave. Moreover, its subdifferential at $\boldsymbol{\lambda}$ is $\partial L(\boldsymbol{\lambda}) = \text{conv}\{\boldsymbol{g}(\boldsymbol{x}) : \boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathcal{X}} \mathscr{L}(\boldsymbol{x}', \boldsymbol{\lambda})\}$.*

Note that Remark 1.2.1 does not always apply for $(R_{Lag}(\boldsymbol{\lambda}))$, for an arbitrary value of $\boldsymbol{\lambda} \in \mathbb{R}^n_+$: given $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \in X} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda})$, we may have $\boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}^*) < 0$ and hence $f(\boldsymbol{x}^*) > \mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda})$. In such a case, one cannot guarantee in general that there is no feasible solution $\boldsymbol{x}'$ of (P_{Lag}) with $\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}) \leq f(\boldsymbol{x}') < f(\boldsymbol{x}^*)$. However, Remark 1.2.1 holds when constraints (1.2.21) take the special form $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$. Even if it is not the case, under certain conditions, it is possible to prove that $\boldsymbol{x}^*$ is optimal for (P), thanks to the following result.

Therorem 1.2.3 (Saddle points of $\mathscr{L}$)**.** *A solution $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$, with $\boldsymbol{x}^* \in X$ and $\boldsymbol{\lambda}^* \in \mathbb{R}^m_+$ is a saddle point of the Lagrangian function $\mathscr{L}$ if and only if:*

a) $\boldsymbol{x}^*$ is an optimal solution of $(R_{Lag})(\boldsymbol{\lambda}^*)$: $\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \min_{\boldsymbol{x} \in \mathcal{X}} \{\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}^*)\}$

b) $\boldsymbol{x}^*$ is feasible for $(P)_{Lag}$: $\boldsymbol{g}(\boldsymbol{x}^*) \leq 0$

c) $\boldsymbol{\lambda}^{*\top} \boldsymbol{g}(\boldsymbol{x}^*) = 0$

If $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is a saddle point of $\mathscr{L}$, then $\boldsymbol{x}^*$ is optimal for (P_{Lag}) and $\boldsymbol{\lambda}^*$ is optimal for (D_{Lag}).

Under mild constraint qualification assumptions, convex programs (and linear programs as a special case) have a strong Lagrangian dual (see Theorem 1.2.4). As a consequence, the optimal value of (P_{Lag}) can be obtained by solving (D_{Lag}), which might be easier in practice. Moreover, some methods [Barahona & Anbil 2000] for solving (D_{Lag}) provide a solution of (P_{Lag}) as a by-product.

Therorem 1.2.4 (Strong duality theorem)**.** *Suppose that $\mathcal{X}$, f and $\boldsymbol{g}$ are convex (that is, (P_{Lag}) is a convex program). Further, assume that there exists $\hat{\boldsymbol{x}} \in \mathcal{X}$ with $\boldsymbol{g}(\hat{\boldsymbol{x}}) < 0$. If (P_{Lag}) admits an optimal solution of value γ, then (D_{Lag}) admits an optimal solution of the same value γ.*

Applications with a natural formulation leading to a strong Lagrangian dual are rather rare. However, the case occurs more frequently when reformulation techniques are used – some solution strategies are even based on the construction of such models (see Section 1.2.4.4).

Relation with MILP reformulations Under some assumptions, the Lagrangian dual problem (D_{Lag}) is in fact equivalent to the replacement of $\mathcal{X}$ with its ideal formulation conv $\mathcal{X}$ (introduced in Section 1.2.1.1). For the special case where (P_{Lag}) is an integer linear program, [Geoffrion 1974] gives a proof of this result, relying on *partial LP duality* (see [Geoffrion 1971]). For the same case, another

proof can be derived from the epigraph LP formulation of the dual function and LP duality (see *e.g.* [Vanderbeck & Wolsey 2010]).

We propose an alternative proof that requires different assumptions.

Therorem 1.2.5. *Consider problem* (P_{Lag}) *and assume that* (P_{Lag}) *is feasible and bounded,* $\mathcal{X}$ *is bounded and finite dimensional, and functions* f, g_i, $i = 1, \dots, m$ *are affine. Moreover, there exists* $\hat{\boldsymbol{x}} \in ri(\text{conv } \mathcal{X})$ *such that* $\boldsymbol{g}(\hat{\boldsymbol{x}}) \leq \boldsymbol{0}$. *Then problem* (D_{Lag}) *is equivalent to:*

$$(P_{Lag}^{\text{conv}}) \ : \ \min \ f(\boldsymbol{x}) \tag{1.2.23}$$

$$s.t. \ \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \tag{1.2.24}$$

$$\boldsymbol{x} \in \text{conv } \mathcal{X} \tag{1.2.25}$$

Proof. First remark that, under our assumptions, function $\mathscr{L}(\cdot, \boldsymbol{\lambda})$ is concave for all $\boldsymbol{\lambda} \in \mathbb{R}^m_+$. It follows that we can extend $\mathcal{X}$ to its convex hull in problem (R_{Lag}) and obtain an equivalent problem. Hence, problem (D_{Lag}) is equivalent to $\sup_{\boldsymbol{\lambda} \in \mathbb{R}^m_+} \min_{\boldsymbol{x} \in \text{conv } \mathcal{X}} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda})$. Since f is continuous and (P_{Lag}) is bounded, $f(\hat{\boldsymbol{x}})$ is finite. So $\mathscr{L}(\hat{\boldsymbol{x}}, \cdot)$ is upper bounded (by value $f(\hat{\boldsymbol{x}})$ because $\boldsymbol{g}(\hat{\boldsymbol{x}}) \leq \boldsymbol{0}$ and $\boldsymbol{\lambda} \geq \boldsymbol{0}$). Besides, $\mathscr{L}(\cdot, \boldsymbol{\lambda})$ is convex for all $\boldsymbol{\lambda} \in \mathbb{R}^m_+$, $\mathscr{L}(\boldsymbol{x}, \cdot)$ is concave for all $\boldsymbol{x}$, conv $\mathcal{X}$ is finite dimensional and bounded. Hence, Theorem 1 of [Perchet & Vigeral 2015] applies and the min and sup operators can be swapped to obtain the equivalent formulation of (D_{Lag}): $\min\{\sup_{\boldsymbol{\lambda} \in \mathbb{R}^m_+} f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) : \boldsymbol{x} \in \text{conv}\mathcal{X}\}$. Remark that the sup problem is bounded if and only if $\boldsymbol{g}(\boldsymbol{x}) \leq 0$. It follows that (D_{Lag}) is equivalent to $\min\{f(\boldsymbol{x}) : \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}, \boldsymbol{x} \in \text{conv } \mathcal{X}\}$. □

Remark that the regularity assumption on the existence of $\hat{\boldsymbol{x}}$ is satisfied when $\dim(\mathcal{X})(\leq n)$ feasible points of (P_{Lag}) with linearly independent projections on conv $\mathcal{X}$ exist, which is more restrictive than the proofs mentioned above.

These results show that the dual bound defined by the Lagrangian relaxation of constraints $\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}$ is the same as the one obtained by the dynamic programming-based reformulation of subsystem $\mathcal{X}$, or by its DW reformulation. In the latter case, the Lagrangian subproblem turns out to be identical to the DW pricing problem.

1.2.3.2 Solving the dual problem

In order to optimize the Lagrangian dual function, [Lemaréchal 2001] counts several type of methods: subgradient-based, cutting plane, ellipsoid, Analytic Center Cutting Plane Method (ACCPM) and bundle methods. In our work, we use only subgradient-based procedures (and, in some sense, column generation when DW decomposition is chosen). This choice is motivated first by the good empirical performance of these algorithms: confirmed by the computational study of [Frangioni *et al.* 2017], that shows that well-tuned subgradient procedures are competitive with more complex ones for large-scale problems (although bundle methods seem more effective for moderate size problems). The second reason is their simple and flexible implementation, and their few requirements, that [Lemaréchal 2001] summarizes as:

- *The only information available on* [the dual function] θ *is an oracle, which returns* $\theta(u)$ *and one subgradient* $g(u)$*, for any given* $u \in \mathbb{R}^m$.
- *No control is possible on* $g(u)$ *when* $\partial\theta(u)$ *is not the singleton* $\nabla\theta(u)$.

These requirements on the components of the optimization algorithm match well with the combinatorial algorithms usually used to solve the Lagrangian subproblem, often based on dynamic programming or combinatorial considerations (in the sense that they do not ask for designing and developing an (effective) algorithm able to return several optimal solutions, or dealing with artificially added non-linearities...).

The basic subgradient algorithm computes a sequence of feasible solutions based on the following recursion, inspired by the descent methods in smooth optimization:

$$\boldsymbol{\lambda}^{k+1} = P(\boldsymbol{\lambda}^k + s_k \boldsymbol{d}^k)$$

Here, s_k is the step length, and $\boldsymbol{d}^k \in \mathbb{R}^m$ is the moving direction, chosen in $\partial L(\boldsymbol{\lambda}^k)$. Function $P : \mathbb{R}^m \to \mathbb{R}^m_+$ is a projection operator that ensures the feasibility of all iterates. A number of subgradient-based optimization algorithms exist, according to the different choices made for these elements. Direction deflecting strategies have been proposed in order to stabilize the procedure, by reducing the zigzagging effect observed when a raw subgradient is used (see [Belgacem & Amir 2018]). Various schemes have been proposed to define the step size s_k, such as the Polyak rule [Polyak 1969]. The Volume algorithm [Barahona & Anbil 2000] combines smoothing of the directions and the *ColorTV rule* (*sic*) for the step size.

Similar to DW or Benders decomposition, when $\mathcal{X}$ is composed of independent subsystems, the dual function can be computed by solving one independent subproblem per subsystem. This makes Lagrangian relaxation a suitable choice for problems with a constraint matrix exhibiting **a block diagonal structure with a set of linking rows**.

1.2.3.3 State-space relaxation

Introduced by [Christofides *et al.* 1981], the state-space relaxation consists in projecting the state space of a dynamic program on a smaller state space in order to derive lower bounds. Specifically, let ρ be the projection operator of the original state space $Q_\perp$ to a smaller space. To define a proper relaxation, the dynamic program resulting from the projection $\rho(Q_\perp)$ should verify the following property: if two states $q, q' \in Q_\perp$ are such that q' can be reached from q after applying a single transition, then $\rho(q')$ can be reached from $\rho(q)$. The functional equation of the relaxed DP is adapted from (1.2.4):

$$\overrightarrow{\Phi}_\rho(q) = \inf\left\{\overrightarrow{\Phi}_\rho\left(q'\right) + c(q'', a) |\ q'' \in Q, a \in \Sigma, \delta(q'', a) = q, q' \in \rho(q'')\right\} \quad \forall q \in \rho(Q - \{q_0\})$$

[Christofides *et al.* 1981] also proposed to penalize the violation of the constraints that are discarded in the relaxed DP, in a Lagrangian fashion.

Formally, let us consider combinatorial optimization problems of the form:

$$\begin{aligned}(CO_{SS}) : \min \quad & f(\boldsymbol{x}) \\ s.t. \quad & \boldsymbol{x} \in \mathcal{X} \\ & g_i(\boldsymbol{x}) \leq 0 \ \forall i \in I\end{aligned}$$

Let us assume that (CO_{SS}) is a special case of (P_{Lag}) ((1.2.20)-(1.2.22)) for which we know a dynamic programming algorithm defined by $\mathcal{M} = (Q_\perp, \Sigma, \delta, q_0, q_\bullet, c)$ (see Section 1.2.2.1). Typically, in single machine scheduling, each constraint $g_i(\boldsymbol{x}) \leq 0$ controls that job i is performed exactly once, while $\mathcal{X}$ encodes the set of pseudo-schedules satisfying the *disjunctive constraints* (*i.e.* the machine capacity constraints). The problems we consider are such that, when a subset of constraints $J \subseteq I$ is removed, the resulting problem can be solved by projecting $Q_\perp$ onto a smaller state-space $Q_\perp^{I-J}$. Moreover, the dynamic program is able to handle the more general objective function $\mathscr{L}^J(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x})$, for all $\boldsymbol{\lambda} \in \mathbb{R}_+^{|I|}$ (one can assume without loss of generality (wlog) that $\boldsymbol{\lambda}_i = 0$ for $i \in I - J$). These assumptions hold for many single machine scheduling problems, and single vehicle routing problems that arise in more complex applications. Under those conditions, state-space relaxation allows defining a family of Lagrangian relaxations of (CO_{SS}), that are parameterized by the set J of dualized constraints. Computing the Lagrangian dual function associated with dualized constraints J for multipliers $\boldsymbol{\lambda}$ amounts to solve, with the help of the relaxed dynamic program:

$$\begin{aligned}(CO_{SS})(J, \boldsymbol{\lambda}) : \min \quad & f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) && (1.2.26)\\ s.t. \quad & \boldsymbol{x} \in \mathcal{X} && (1.2.27)\\ & g_i(\boldsymbol{x}) \leq 0 \ \forall i \in I - J && (1.2.28)\end{aligned}$$

The relaxations defined in this way are more or less easy to solve, and more or less strong. Iterative state-space relaxation methods (Section 1.2.4.4) progressively build stronger and stronger relaxations, potentially possessing the strong duality property.

1.2.3.4 Lagrangian-cost variable fixing

Relaxation-based variable fixing is a technique used to fix the value of some decision variables to their optimal value, thus eliminating them from the optimization problem. Its basic principle is simple: given problem $(P) : \min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X}\}$ and a primal bound $\overline{z}$ of (P), define problem $(P(\mathcal{H}))$ by imposing an additional restriction on the solutions: $(P(\mathcal{H})) : \min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X}, \boldsymbol{x} \in \mathcal{H}\}$. Compute a dual bound $\underline{z}(\mathcal{H})$ of $(P(\mathcal{H}))$. If $\underline{z}(\mathcal{H}) > \overline{z}$, then we proved that (P) is equivalent to $\min\{\boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{X} - \mathcal{H}\}$, and the same process can be applied to this reduced problem with other hypotheses. This broad description covers a very wide range

of algorithmic techniques, that encompasses even the basic principle of branch-and-bound. The specificity of relaxation-based variable fixing is to rely on the quick computation of the conditional lower bounds. The concept has emerged in various fields of optimization: for example, in scheduling under the name *shaving* [Carlier & Pinson 1994, Martin & Shmoys 1996, Carlier *et al.* 2004] when used with combinatorial or Lagrangian dual bounds, in Constraint Programming as *constraint programming based Lagrangian relaxation* [Sellmann 2004] (Lagrangian bound), in Convex Optimization and Machine Learning as *screening* [Ghaoui *et al.* 2011, Atamtürk & Gomez 2020] (Fenchel dual bound), etc.

Reduced-cost fixing In the context of MILP, reduced-cost fixing relies on an optimal solution of the LP relaxation and the associated reduced costs. According the notations above, the basic hypothesis $\mathcal{H}$ that is tested here in a binary LP is of the type $\mathcal{H} = \{\boldsymbol{x} : x_i = 1\}$. [Dantzig *et al.* 1954] uses the idea in an ad-hoc procedure to solve for the first time a 49-city TSP instance, and [Crowder *et al.* 1983] describes its integration in a Binary Linear Programming solver. The following result formalizes the idea.

Proposition 1.2.6. *Let us consider the mixed integer linear program* $(P) : \min\{\boldsymbol{c}^\top \boldsymbol{x} : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{N}^p \times \mathbb{R}_+^{n-p}\}$, $\boldsymbol{\lambda}^*$ *an optimal dual solution of* $\overline{(P)}$ *and* $\underline{z}$ *its value. Moreover, let* $\mathcal{H} = \{\boldsymbol{x} \in \mathbb{R}_+^n : x_k \geq \hat{x}_k\}$ *and* $(P(\mathcal{H})) = \min\{\boldsymbol{c}^\top \boldsymbol{x} : \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{H}, \boldsymbol{x} \in \mathbb{N}^p \times \mathbb{R}_+^{n-p}\}$.

Then $\underline{z} + (c_k - \mathbf{A}^{k\top}\boldsymbol{\lambda}^*)^\top \hat{x}_k$ *is a dual bound for* $(P(\mathcal{H}))$.

The proposition can be proven using LP duality arguments, or Lagrangian duality as we show below. If a primal bound for (P) is known, this results implies that in any optimal solution, $x_k \leq \frac{\overline{z}-\underline{z}}{c_k - \mathbf{A}^{k\top}\boldsymbol{\lambda}^*}$ holds for all k such that $c_k - \mathbf{A}^{k\top}\boldsymbol{\lambda}^* > 0$. If we also have that x_k must be integer, the right-hand-side can be rounded down: $x_k \leq \lfloor \frac{\overline{z}-\underline{z}}{c_k - \mathbf{A}^{k\top}\boldsymbol{\lambda}^*} \rfloor$.

Lagrangian-cost variable fixing In order to solve large-scale DP models, [Ibaraki 1987] bases the Successive Sublimation Dynamic Programming (SSDP) method (see Section 1.2.4.4) on a graph representation (like the one presented in Section 1.2.2.1) of a relaxed DP(1.2.3.3). The algorithm includes an *elimination* phase during which irrelevant states are eliminated. The routine is based on the efficient computation of a set of lower bounds. For each arc in the graph, the minimum cost path going through this arc is calculated. The key is to compute all those shortest paths at the same time, in a forward and a backward pass of the Bellman algorithm. The hypothesis $\mathcal{H}$ that is tested in the elimination phase is $\mathcal{H} = \{\boldsymbol{x} : x_e = 1\}$, where $x_e = 1$ if and only if the path representation of the solution uses arc e. [Ibaraki & Nakamura 1994] improve this filtering technique by considering the minimum Lagrangian cost that goes through the arc. The same idea is redeveloped under the name of *graph cleaning* by [Sourd 2009] for the single machine earliness-tardiness problem. This principle is also used in [Focacci *et al.* 1999, Demassey *et al.* 2006] in

the COST-REGULAR constraint in the Constraint Programming (CP) context. Years later, the same idea called *path-reduced cost fixing* is redeveloped starting from the LP reduced-cost fixing in the context of vehicle routing in [Irnich *et al.* 2010].

Based on another state-space relaxation scheme in the single machine scheduling context, [Detienne *et al.* 2010] uses various Lagrangian dual bounds to test problem-specific hypothesis. Another originality of this work is the exploitation of the decomposition of independent subsystems to effectively compute the conditional bounds. The most basic assumption "can job i start at time t?" comes down to LP reduced-cost fixing, while the more sophisticated assumptions "can the optimal schedule start before (resp. end after) time t" and "can job i be before job j" exploit dominance rules and require solving small auxiliary dynamic programs.

As the following proof shows, reduced-cost fixing is clearly a special case of Lagrangian-cost fixing.

Proof of Proposition 1.2.6. Let us consider the Lagrangian relaxation of (P) where constraints $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}$ are dualized. The Lagrangian subproblem reads $(R_{Lag})(\boldsymbol{\lambda})$: $\min\{(\boldsymbol{c}+\mathbf{A}^\top\boldsymbol{\lambda})^\top\boldsymbol{x}-\boldsymbol{\lambda}^\top\boldsymbol{b} : \boldsymbol{x} \in \mathbb{N}^p\times\mathbb{R}_+^{n-p}\}$. Imposing hypothesis $\mathcal{H}$, the subproblem becomes $(R_{Lag})(\boldsymbol{\lambda},\mathcal{H})$: $\min\{(\boldsymbol{c}+\mathbf{A}^\top\boldsymbol{\lambda})^\top\boldsymbol{x}-\boldsymbol{\lambda}^\top\boldsymbol{b} : \boldsymbol{x} \in \mathcal{H}\cap(\mathbb{N}^p\times\mathbb{R}_+^{n-p})\}$. The optimal solution of $(R_{Lag})(-\boldsymbol{\lambda}^*,\mathcal{H})$ is trivially $x_i = 0$ for all $i \neq k$, and $x_k = \hat{x}_k$, whose cost is $(c_k - \mathbf{A}^{k\top}\boldsymbol{\lambda}^*)^\top\hat{x}_k + \boldsymbol{\lambda}^{*\top}\boldsymbol{b}$. □

The hypograph formulation (1.2.29) of the Lagrangian dual function permits a visual insight of Lagrangian-cost variable fixing (Figure 1.2.1).

$$\begin{aligned}(D_{Lag}) : \max_{\boldsymbol{\lambda}\in\mathbb{R}_+^m} \min\Big\{f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top\boldsymbol{g}(\boldsymbol{x}) : \boldsymbol{x} \in \mathcal{X}\Big\}\\ = \max\Big\{\theta : \theta \leq f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top\boldsymbol{g}(\boldsymbol{x})\ \forall\boldsymbol{x} \in \mathcal{X}, \boldsymbol{\lambda} \in \mathbb{R}_+^m\Big\}\end{aligned} \tag{1.2.29}$$

The figure shows, among other things, that as pointed out by [Sellmann 2004], the best multipliers for (D_{Lag}) are not always the most appropriate for fixing variables. This is why, in our work, we embed variable fixing routines in the course of a subgradient algorithm, in order to perform filtering based on diversified Lagrangian multipliers.

1.2.3.5 Work with Lagrangian relaxation and SSR

Lagrangian relaxation has been used in the context of SSR , in conjunction with Lagrangian cost variable fixing, for solving scheduling problems in [Detienne *et al.* 2010, Detienne *et al.* 2012, Detienne *et al.* 2016], and the Temporal Knapsack Problem (TKP) in [Clautiaux *et al.* 2021], with various primal schemes taking advantage of the very strong dual bounds obtained (**Issue 1**).

In [Detienne *et al.* 2009], it permits exploiting the natural decomposability of the employee timetabling problem. In [Tanaka *et al.* 2015], strong dual bounds are obtained at the price of a very large scale formulation which is optimized using Lagrangian decomposition (**Issue 1**, **Issue 3**). Lagrangian multipliers serve as a guide to obtain good feasible solutions for a lot-sizing problem in [Absi *et al.* 2013].

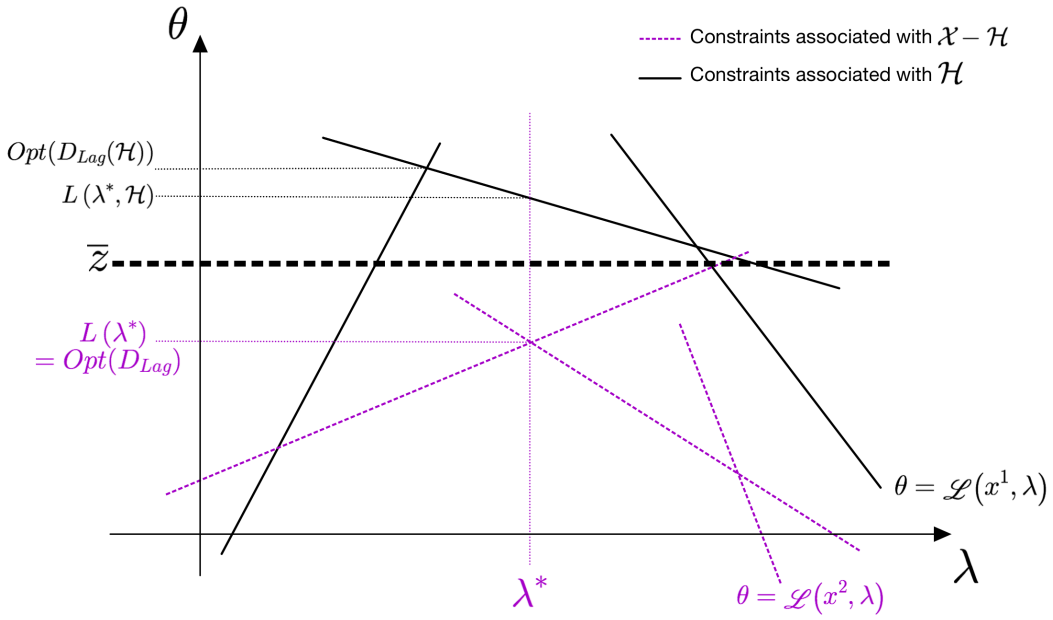


Figure 1.2.1: Illustration of the Lagrangian-cost variable fixing based on the hypograph formulation (1.2.29) of an MILP model. The Lagrangian relaxation is obtained by relaxing only one constraint, whose multiplier is λ. Each solution $\boldsymbol{x} \in \mathcal{X}$ appears as an affine function (linear constraint). Plain black lines are associated with solutions in $\mathcal{X}$ that satisfy hypothesis $\mathcal{H}$, while dashed purple lines are associated with the other solutions of $\mathcal{X}$.
The value of $L(\lambda)$ is achieved at the minimum of all the affine functions at λ. The optimal solution λ^* of (D_{Lag}) is obtained by finding the maximum of the hypograph. The value of the dual function under hypothesis $\mathcal{H}$ at λ^* is obtained by finding the minimum of the affine functions only associated with $\mathcal{H}$ at λ^*. Even though λ^* is not optimal for $(D_{Lag}(\mathcal{H}))$, $L(\lambda^*, \mathcal{H})$ is larger than the known primal bound $\overline{z}$. That proves that no solution in $\mathcal{H}$ can be optimal.
Note that the optimal solution of $(D_{Lag}(\mathcal{H}))$ is not $\boldsymbol{\lambda}^*$. Hence, stronger variable fixing could be achieved using different Lagrangian multipliers.

1.2.4 Relaxation-based solution schemes

Solving a relaxation ensures getting a lower bound on the optimal value of our original problem. However, it does not provide a feasible solution in general. This section summarizes various ways of obtaining optimal solutions based on the successive solution of relaxations: branch-and-bound, constraint generation and iterative state space relaxation are presented under a same generic framework.

1.2.4.1 Obtaining optimal solutions: abstract algorithm

We now give a schematized view of the various algorithms utilized to get an optimal solution to a combinatorial problem based on various relaxations. The common principle to all these methods is to refine a relaxation until a feasible solution is found and its optimality is proven, by the process of excluding infeasible solutions. They differ in the way this exclusion is performed. For the sake of conciseness, our abstract descriptions do not allow the instantiation of all subtleties of the specific procedures. By a slight abuse of notation, in Algorithms 1.2.1 and 1.2.2, notation (P) refers both to the problem (P) and its set of feasible solutions, depending on the context.

Special case: $f_R = f$ We first focus on the special case where Remark 1.2.1 applies, which covers the classical LP-based branch-and-bound, algorithms based on Lagrangian relaxation of equality constraints only or based on combinatorial relaxations. Algorithm 1.2.1 works with a set Q of *open* relaxations, that are defined but not solved yet. At each iteration, an open relaxation (R) is picked up, removed from the queue and solved (`line 3`). If its optimal value is not better than the best known primal bound $\overline{z}$ (`line 4`), then we know that (R) contains no improving solution, and we start again the loop with another open relaxation - (R) is closed. Otherwise, `line 5` checks whether s is feasible for the global problem. If this is the case, we update the best known primal bound (`line 6`), and conclude that (R) contains no better feasible solution. Hence, it is closed.

If s is not feasible for (P), then we have to further explore the set of solutions of (R) for potentially feasible solutions. In this purpose, we define a new relaxation that contains the feasible solutions of $(P) \cap (R)$. Some precautions have to be taken in `line 8` if finite convergence of the algorithm is desired. We propose, as a general rule, to associate a finite measure value $\#(R)$ to each relaxation (R), such as the diameter of the relaxed feasible set or the number of integer points contained in the relaxed feasible set. We also assume that it is defined in such a way that it is bounded from below for any relaxation generated by the algorithm (by zero in our examples). If there exists $\delta_\# > 0$ such that, for each relaxation (R_i) defined in `line 8`, we have that $\#(R_i) + \delta_\# \leq \#(R)$, then Algorithm 1.2.1 converges in a finite number of steps since $\#$ is bounded from below and $\#(R_0)$ is finite. Note that if the set of feasible solutions of (R) is finite, it is sufficient to guarantee that $s \notin \bigcup_{i=1}^{r}(R_i)$ (this typically happens in iterative state-space relaxation algorithms).

Algorithm 1.2.1: Generic overall solution algorithm.
Special case where $f_R = f$ (see Remark 1.2.1)
Input: problem (P), optimality tolerance ε

```
1 Define a first relaxation (R_0) of (P). Q ← {(R_0)}. z̄ ← ∞;
2 while Q ≠ ∅ do
3     Pick up (R) from Q. Solve (R) and let s be an optimal solution;
4     if f_R(s) + ε < z̄ then
5         if s ∈ (P) then
6             Register s as a primal solution, z̄ ← f(s)(= f_R(s)).
7         else
8             Define a family of relaxations (R_i)_{i=1,...,r} such that
                (P) ∩ ⋃_{i=1}^r (R_i) = (P) ∩ (R) and #(R_i) + δ_# ≤ #(R);
9             Add (R_i) to Q for all i = 1, ..., r.
```

This can be modeled by choosing # as the cardinal of the relaxed feasible set, and $\delta_{\#} = 1$. Depending on the algorithm, the new relaxation can take the form of a single refined relaxation, or the disjunction of several "smaller" relaxations (typically by branching). Finally, (R) is closed and the new open relaxations are added to Q.

General case: $f_R(\boldsymbol{x}) \leq f(\boldsymbol{x})$ This case arises, for example, when a Lagrangian relaxation of inequality constraints is performed or when refining an approximation of the cost function from below (case of Benders optimality cuts). In this case, solution s computed at `line 3` may be feasible for (P), but we cannot be sure that no better feasible solution exist in (R) when $f_R(s) < f(s)$.

The generic Algorithm 1.2.2 deals with this situation by either refining the current relaxation by reducing its feasible set like in Algorithm 1.2.1 (`line 8`, case (i)), or by modifying the objective function in the subsequent relaxations (`line 8`, case (ii)). This modification of f_R is such that the objective value of s in relaxation (R_i), $f_{R_i}(s)$, is strictly larger than its value $f_R(s)$ in the current relaxation. Further, condition (ii) ensures that the value of s will never become smaller in all child relaxations of (R_i). Hence, the finite convergence is guaranteed if either (a) the number of solutions of (P) is finite and case (ii) is active only when s is feasible for (P), or (b) each relaxation (R) has a finite number of feasible solutions. Note that this is still not sufficient to ensure the finite convergence of the algorithm (in particular when f is not continuous and the number of feasible solutions of (P) is not finite). Concrete examples where case (i) or case (ii) can be implemented are described below, in Section 1.2.4.2 and Section 1.2.4.4. Once again, when the set of feasible solutions of (R) is finite, replacing conditions (i) and (ii) by assumption $s \notin \bigcup_{i=1}^{r}(R_i)$ ensures the finite convergence.

Algorithm 1.2.2: Generic overall solution algorithm.
Input: problem (P), optimality tolerance ε

1 Define a first relaxation (R_0) of (P). $Q \leftarrow \{(R_0)\}$. $\overline{z} \leftarrow \infty$;
2 **while** $Q \neq \emptyset$ **do**
3 Pick up (R) from Q. Solve (R) and let s be an optimal solution;
4 **if** $f_R(s) + \varepsilon < \overline{z}$ **then**
5 **if** $s \in (P)$ **then**
6 Register s as a primal solution and possibly update the primal bound $\overline{z}$.
7 **if** $f_R(s) + \varepsilon < f(s)$ *(i.e. s cannot be proven $\varepsilon-$optimal for $(P) \cap (R)$)* **then**
8 Define a family of relaxations $(R_i)_{i=1,\dots,r}$ such that $(P) \cap \bigcup_{i=1}^{r}(R_i) = (P) \cap (R)$ and, for each i, either:
(i) $\#(R_i) + \delta_{\#} \leq \#(R)$
(ii) For all ancestors (R^-) of (R) whose optimal solution found at `line 4` was s^-, $f_{R_i}(s^-) \geq f_{R^-}(s^-) + \varepsilon'$
Add (R_i) to Q for all $i = 1, \dots, r$.

1.2.4.2 Branch-and-bound

First described in [Land & Doig 1960] (republished in [Land & Doig 2010]), branch-and-bound is one of the most used procedure in optimization [Wolsey & Nemhauser 1999]. The instantiation of Algorithm 1.2.2 as a branch-and-bound routine is done by associating a node of a search tree to each relaxation. Among the various approaches we list in the following sections, it has the specificity of generating several child relaxations in `line 8`, usually by branching on the value of a variable. Depending on the application, one can also branch on more complex structures like the next job in a schedule, the next arc in a path, a set of jobs that will not be processed next...

Ensuring the finite convergence of the algorithm is usually not difficult when dealing with bounded MILP problems: branching on an integer variable with a fractional value in s strictly decreases the number of such branching possibilities in the resulting relaxation. When using other strategies like spatial branching on continuous variables (like in [Detienne *et al.* Submitted]), the volume of the relaxed set is decreased at each step by a finite amount. However, one must invoke another mechanism to guarantee that this amount is bounded below by a strictly positive value. For example, if the objective function f is Lipschitz-continuous, then all solutions in a box have the same objective value within a tolerance equal to the diameter of the box times the Lipschitz constant of f. Hence, choosing the diameter of the relaxed feasible set for measure value # (and a branching scheme that generates

smaller and smaller boxes) and a non-zero tolerance ε prevents from an infinite loop thanks to the test in `line 4`.

Now let us describe an example of branch-and-bound algorithm where case (i) and case (ii) are relevant. Consider a procedure based on the Lagrangian relaxation of inequality constraints in model (P_{Lag})(1.2.20)-(1.2.22). Assume that the relaxation (R_k) is defined, among other components, by Lagrangian multipliers $\boldsymbol{\lambda}^k$. Its optimal solution $\boldsymbol{x}$ found at `line 3` is feasible for (P), and $\boldsymbol{\lambda}^{k\top}\boldsymbol{g}(\boldsymbol{x}) < 0$, *i.e.* $f_{R_k}(\boldsymbol{x}) < f(\boldsymbol{x})$. Contrary to the context of LP-based branch-and-bound, we are not able to easily branch on some component of $\boldsymbol{x}$ to exclude s from the search (what must be discarded is the whole vector $\boldsymbol{x}$, not a forbidden value of one component). An example of this situation is the problem of finding resource-constrained shortest paths in a Directed Acyclic Graph (DAG) with a single source and sink, resulting from a SSR of a DP. The set of solutions explored is the set of paths, subject to additional constraints, from the source to the sink. Each node k of the search tree is associated with a relaxation (R_k), a DAG node $u(k)$, and an already fixed path from the source node to $u(k)$. Measure $\#(R_k)$ is equal to the maximum length from $u(k)$ to the sink. Relaxation (R_k) consists in finding the minimum Lagrangian cost from the source node to the sink node that goes through node $u(k)$.

- Case (i) occurs when we branch on the successor of the current node $u(k)$ in the DAG and we keep Lagrangian multipliers $\boldsymbol{\lambda}^k$. At the successor node $k+1$ of k, we might very well find $\boldsymbol{x}$ again when solving (R_{k+1}), with exactly the same cost as before. However no infinite loop is possible since we have progressed towards the sink of the DAG.

- Case (ii) occurs when we decide to change the value of $\boldsymbol{\lambda}^k$, for example in the hope to prove optimality of $\boldsymbol{x}$ using Theorem 1.2.3. In this purpose, we choose a relaxed constraint with $\lambda_i^k g_i(\boldsymbol{x}) \leq -\varepsilon'$, and set $\lambda_i^{k+1} \leftarrow 0$ to define a new relaxation (R_{k+1}), also associated with node $u(k+1) = u(k)$ of the DAG. We have $f_{R_{k+1}}(\boldsymbol{x}) = f_{R_k}(\boldsymbol{x}) - \lambda_i^k g_i(\boldsymbol{x}) \geq f_{R_k}(\boldsymbol{x}) + \varepsilon'$. Since $\boldsymbol{x}$ is feasible for (P), we have that $\boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0}$. It follows that not increasing the Lagrangian multipliers in descendants ℓ of k ensures that $f_{R_\ell}(\boldsymbol{x}) > f_{R_\ell}(\boldsymbol{x}) + \varepsilon'$. Another sufficient condition for finite convergence is to restrict Lagrangian multipliers in descendants ℓ of k so that $\lambda_i^\ell = 0$. The dimension of $\boldsymbol{\lambda}$ being finite, we can then use a modified version of measure $\#$ that combines the length from $u(\ell)$ to the sink with the number of components of $\boldsymbol{\lambda}$ fixed to zero to prove the convergence of Algorithm 1.2.2.

Work with branch-and-bound algorithms Such algorithms have been developed in two contexts. For scheduling problems, Lagrangian relaxations are used in [Detienne *et al.* 2016] (Section 2.1) and [Detienne *et al.* 2010]. The algorithm of the first paper relies on SSR, and explores the set of paths in the graph of relaxed Dynamic Programs. The second paper makes intensive use of Lagrangian cost variable fixing in order to reduce the time windows of the jobs. Branching is done

by ordering pairs of jobs (*selection of disjunctions* [Carlier & Pinson 1989]), or by splitting the time window of a job.

The studies in [Arslan & Detienne 2021] (Section 3.2) and [Griset *et al.* 2021] (Section 3.1) make use of DW reformulation. In order to obtain integer feasible solutions, we use branch-and-price algorithms, which are branch-and-bound procedures that solve the linear relaxation of the DW reformulation and branch on the value of variables with integer requirements but fractional optimal value in the relaxation. In the first paper, there is no integrality restrictions on the subproblem variables, so that classical branching on the set of non-dynamically-generated variables is sufficient. In order to cope with cases where DW reformulation leads to a relaxation of the problem, we devise a set of additional cuts that yield an exact reformulation. We propose a branch-and-price-and-cut algorithm to solve the resulting model. In [Griset *et al.* 2021], DW subproblems are shortest path problems in DAG. Hence, it is sufficient to branch on the value of the original arc-variables in the master program to ensure their integrality (cf. [Vanderbeck 2000], Proposition 4). A branch-and-cut-and-price algorithm is proposed to deal with both DW and Benders reformulations.

1.2.4.3 Constraint generation/cutting planes

In this section, we focus on problems defined as MILP models with a bounded feasibility set. In this document, we make the following distinction between *cutting plane* methods and *constraint generation* methods. We define cutting plane algorithms as procedures that iteratively generate valid inequalities in order to strengthen the linear relaxation of a MILP model (see for example [Conforti *et al.* 2010]). We use the terms constraint generation specifically for procedures that are based on a combinatorial relaxation of the problem (*i.e.* where some constraints are just omitted) that generates these constraints when needed. While the cuts of the cutting plane algorithms are optional for the proper definition of the problem, the constraints of the constraint generation methods are mandatory for its full description. In our work, we are interested in branch-and-Benders-cut (Section 1.2.2.3), cut-and-branch, and pure constraint generation algorithms.

Branch-and-Benders-cut procedures [Fortz & Poss 2009] can be regarded either as satisfying Remark 1.2.1 or not, depending on whether we include the optimality cuts as constraints linking variables θ and $\boldsymbol{x}$, or as elements defining the objective function of the relaxation. In both cases, they can be seen as branch-and-bound algorithms. The generation of cuts at a given node of the search tree can be viewed as generating a single new relaxation in `line 8`. When optimality cuts are considered as proper constraints, Algorithm 1.2.1 can be applied and one can define a measure # based on the number of cuts and the number of branching constraints already added. Otherwise, Algorithm 1.2.2 is applied. Case (ii) is performed when optimality cuts are added, while case (i) is active with the same measure # when branching on a variable.

Cut-and-branch algorithms are a variant of branch-and-cut where the valid in-

equalities are added only at the root node of the search tree. Pure constraint generation algorithms such as the classical Benders decomposition method described in Section 1.2.2.3 are similar to branch-and-Benders-cut procedure, without the branching phase. Each relaxation is a mixed integer linear program, which can be seen as a combinatorial relaxation of feasibility and optimality cuts.

Work with constraint generation We decomposed some machine scheduling problems into sequencing and timing problems. An alternative model for the second subproblem was designed, based on a polynomial but large number of knapsack constraints (which exhibit a well-understood polyhedral structure that makes solution with generic integer linear programming approaches efficient). This yielded a pure constraint generation method in [Detienne *et al.* 2011]. The same kind of alternative formulation was used to generate a dedicated variant of *knapsack cover cuts* to reinforce another mixed integer linear program in [Detienne 2014], embedded within a cut-and-branch algorithm.

As already mentioned above, in [Griset *et al.* 2021] (Section 3.1), we combine Benders and Dantzig-Wolfe reformulations to solve a real-life planning problem. This section describes a row-and-column generation algorithm, as well as a heuristic version that we call a *price-and-branch-and-Benders-cut* method.

1.2.4.4 Iterative state space relaxation

While the state-space relaxation (Section 1.2.3.3) was primarily thought as a way to compute dual bounds, dual iterative methods based on the refinement of the projection have been proposed to tackle the original problem (P). Since this type of approach is not as widely used as the previous ones, we propose a brief literature review on the topic. These methods essentially focus on solving the Resource Constrained Elementary Shortest Path Problem (RCESPP), or problems implicitly modeled as such. [Christofides *et al.* 1981] proposed to reinforce the dual bound in their scheme using the so-called *state-space ascent*, whose idea is further developed by [Ibaraki 1987] within the SSDP framework (see also [Ibaraki 1988]) and shortly after by [Abdul-Razaq & Potts 1988], who coined the concept of *state-space modifiers* to improve the lower bounds obtained from a state-space relaxation for single machine scheduling. For the same type of problems, [Ibaraki & Nakamura 1994] integrated state-space modifiers inside SSDP. Long after, when computer memory became significantly larger, [Boland *et al.* 2006] presented a state-space augmenting algorithm where the first projection consists in relaxing the elementary requirement of the RCESPP. A very similar approach was introduced independently by [Righini & Salani 2008], who proposed the (famous in the vehicle routing community) Decremental State-Space Relaxation (DSSR). The main difference lies in the fact that only the constraints violated by the optimal solution of the current relaxation are added at a given iteration. [Desaulniers *et al.* 2008] introduced the partially elementary shortest-path problem with resource constraints. They dynamically introduced elementary requirements in this problem based on the solution of the linear

relaxation to strengthen the latter. At the same time, [Tanaka *et al.* 2009, Tanaka & Fujikuma 2012] revisited the SSDP algorithm to propose an effective method for total cost single machine problems.

SSDP and DSSR are two similar methods. They are both based on SSR, and – at least in their basic version – solve successively refined relaxations. After one relaxation is solved, either the obtained solution is proved to be optimal, or at least one relaxed constraint is added to the state space to define the new relaxation. They differ in the way each relaxation is solved. The known implementations of DSSR do not explicitly build the DP graph, but use labeling algorithms instead. Several labels are kept for each vertex of the graph representation of the initial relaxation. One strength of this method is its ability to effectively deal with elementary constraints in routing problems, making use of strong dominance checks. At each step of the algorithm, SSDP explicitly builds the graph representation of the DP expressed in the current relaxed state-space. This extended graph, which can be exponentially large, is used to eliminate some variables with the help of Lagrangian-cost variable fixing. These two versions of iterative state-space relaxations have different advantages. On the one hand, DSSR allows more dominance checking than SSDP, since advanced label-setting/correcting techniques can be used. On the other hand, filtering techniques based on costs apply only to the initial graph in DSSR, whereas SSDP filters arcs from the extended graphs corresponding to all intermediate relaxations.

We now focus on the use of the SSDP algorithm to solve problems with bounded feasible sets and integer solutions. When the assumption of Remark 1.2.1 holds (*i.e.* only equality constraints are relaxed), Algorithm 1.2.1 takes a rather simple form. Each relaxation is a problem of the form (1.2.26)-(1.2.28). If the solution s found in `line 4` is not feasible for (P), then at least one relaxed constraint in (1.2.28) is added to the set J of constraints considered in the dynamic program to construct the refined relaxation. This ensures the convergence of the method, that never removes constraints integrated before (augmenting the state space related to a violated constraint in a state-space relaxation procedure removes a non-zero number of integer solutions in (R)). The approach is rendered more efficient by the integration of Lagrangian cost-variable fixing within the solution of the Lagrangian subproblem. The filtering of states and transitions in a relaxed DP are propagated in the subsequent relaxations, which – hopefully – prevents the explosion of the size of the relaxed DP, keeping the curse of dimensionality at bay.

When inequality constraints are relaxed, case (ii) of Algorithm 1.2.2 appears to be necessary to its convergence[1]. Assume once again that the relaxation (R_k) is defined, among other components, by Lagrangian multipliers $\boldsymbol{\lambda}^k$. Its optimal solution $\boldsymbol{x}$ found in `line 3` is feasible for (P) and $\boldsymbol{\lambda}^{k\top}\boldsymbol{g}(\boldsymbol{x}) < 0$, *i.e.* $f_{R_k}(\boldsymbol{x}) < f(\boldsymbol{x})$. Since $\boldsymbol{x}$ is feasible for (P), integrating any constraint of (1.2.28) into the state space would not avoid finding it again at the next iteration, with the same

[1]This case does not match the original framework of SSDP defined in [Ibaraki 1987], who proves the convergence of the method based on the so-called *Congener theorem*, whose assumptions include the one of Remark 1.2.1. To our knowledge, all published applications of SSDP satisfy this restrictive assumption, and the adaptation of the algorithm to relax the hypothesis is new.

relaxed objective value. Since no new feasible solution has been found, the duality gap is not reduced. Finally, one could add all constraints into the state space without proving the optimality of s (which can indeed be sub-optimal). In order to guarantee the convergence of SSDP in this case, we choose a relaxed constraint with $\lambda_i^k g_i(\boldsymbol{x}) \leq -\varepsilon'$, and set $\lambda_i^{k+1} \leftarrow 0$ to define a new relaxation (R_{k+1}). We then have $f_{R_{k+1}}(\boldsymbol{x}) \geq f_{R_k}(\boldsymbol{x}) + \varepsilon'$. Only decreasing the Lagrangian multipliers in descendants ℓ of k ensure that $f_{R_\ell}(\boldsymbol{x}) > f_{R_\ell}(\boldsymbol{x}) + \varepsilon'$. Here again, we can also choose to fix $\lambda_i^\ell = 0$ in all subsequent relaxations and prove the convergence of Algorithm 1.2.2 with help of a measure # combining the number of relaxed constraints (1.2.28) and the number of components of $\boldsymbol{\lambda}$ fixed to zero.

Work with Iterative State-Space relaxation We applied the SSDP method for Temporal Knapsack Problem in [Clautiaux *et al.* 2021] (Section 2.2). To our knowledge, this is the first attempt to develop a competitive SSDP algorithm for a problem apart from single machine scheduling. It sheds a different light on how to choose, and how many constraints to incorporate in the state-space, and how to reduce the size of the DP model.

We propose in [Detienne *et al.* 2012] a new solution approach based on state-space relaxation, for a generalization of the minimization of the weighted number of late jobs on a single machine. It consists in successively removing infeasible solutions from the state-space, by directly altering the graph representation of the DP. At each iteration, the algorithm defines a single relaxation (R_1) in `line 8` of Algorithm 1.2.1. The relaxation satisfies the hypothesis of Remark 1.2.1. Here, the set of relaxed solutions is the set of paths in a DAG, which is finite. Hence, the convergence of the algorithm is guaranteed by condition $s \notin (R_1)$.

In [Benkirane *et al.* 2020], a large-scale model is proposed for a real-life railway problem. We use SSR without Lagrangian relaxation to obtain a smaller relaxed LP formulation. We then use reduced-cost variable fixing in order to reduce the size of the original model and solve it using a MILP solver (in order to cope with **Issue 3**).

1.3 Main contributions

References to my work are mentioned all along the previous section according to the lens of the methodological tools involved. Some of them are also described in details in Chapter 2 and Chapter 3, while others are summarized in those same chapters. The current section states my main contributions in the solution of industrial large scale problems, robust optimization and scheduling.

1.3.1 Real-life large scale problems

I co-supervised two PhD thesis funded by CIFRE programs, in collaboration with two companies in the fields of transportation and energy.

During the PhD thesis of Mohamed Benkirane at the French railway company SNCF, which I co-supervised with François Clautiaux, we have studied a rolling stock rotation problem, combined with train selection. In this problem, we are given a set of transportation demands defined on time windows, which must be fulfilled with the help of trains. Trains are composed of rail cars and must be assigned times of departure from each station. Various constraints restrict the (re)composition of trains, their possible timing and route. The problem is to decide the route of each rail car to maximize the fulfillment of the demands while satisfying all technical constraints. The challenge of this problem resides in its size and the variety and number of the constraints. We propose a hypergraph-based MILP model that encompasses all the constraints, but whose size does not allow the direct solution using a generic solver. We develop a three phase approach. The first phase solves a state-space relaxation that basically corresponds to ignoring restrictions on the type of rail cars that can be combined in a same train or be assigned to some routes. In the second phase, we repair the solution of this relaxation, to obtain a feasible solution and the associated primal bound. The latter is used within a reduced-cost fixing procedure that eliminates variables of the original model, which is solved during the third phase. The prototype has been tested and used at SNCF on real data in the context of studies on the transportation offer.

I co-supervised, with François Vanderbeck, the PhD thesis of Rodolphe Griset at the French power company EDF. The main study carried out during this project is described in Section 3.1. In a few words, it is about the long-term planning of outages of nuclear plants. We model the industrial problem as a two-stage stochastic problem, which is solved with the help of combined Dantzig-Wolfe and Benders reformulations. The developed prototype has been tested on real instances and showed that taking into account the stochasticity within a global optimization approach could lead to significant profits compared to the current industrial practice.

1.3.2 Two-stage robust problems

The main contribution on this topic is detailed in Section 3.2, and is about two-stage robust problems with integer recourse. There are only few exact approaches suitable for this class of problems. When uncertainty affects the objective function or the constraint matrix, the column-and-row generation algorithm proposed in [Zhao & Zeng 2012a] is not finitely convergent, and does not perform well in practice. On the heuristic side, binary decision rules [Vayanos *et al.* 2011, Bertsimas & Georghiou 2018] and $K-$adaptability models [Hanasusanto *et al.* 2015, Subramanyam *et al.* 2020] are the most appropriate approaches.

We focus on a subclass of those problems, restricted to deterministic constraints with binary first-stage variables in the constraints linking the first and second stages. Our main contribution is a deterministic equivalent MILP formulation. From the methodological perspective, it allows exactly solving the problem, with the help of a branch-and-price algorithm. Our numerical study reported solution times 2 to 4 orders of magnitude lower than $K-$adaptability methods on the applications

considered. From a theoretical perspective, the formulation allows proving that problems of this class are *only* $\mathcal{NP}$-complete (they are not higher in the polynomial hierarchy unless $\mathcal{P} = \mathcal{NP}$).

1.3.3 Scheduling problems

The most significant contributions in the field of scheduling are about two specific single machine and flowshop problems.

In [Detienne 2014], we study $1|r_i|\sum w_i U_i$, the minimization of the weighted number of tardy jobs on a single machine, and generalizations with machine unavailabitily constraints. For this well-studied problem, we propose a non-trivial MILP model and dedicated valid inequalities. These cuts can be separated with the help of DP algorithms in $\mathcal{O}(n^{10})$- and $\mathcal{O}(n^{10}2^n)$-time complexity (with n the number of jobs), respectively. Despite this prohibitive worst-case complexity, restricting the separation algorithm to cases with a small practical solution time already allows designing an effective MILP-based solution approach, improving the state-of-the-art from 200-job instances to 500-job instances.

In [Detienne *et al.* 2016], we added Lagrangian cost variable fixing to the ideas of [Akkan & Karabati 2004] to solve $F2||\sum C_i$ (and $F2|ST_{SI}|\sum C_i$), the two-machine flowshop total completion time problem (with and without setup times). This allowed us to consider an extended DP formulation, and thus integrate known and new dominance rules to reinforce the Lagrangian relaxation. Using memoù-)ization inside the branch-and-bound algorithm finally improved the state-of-the-art from 45 job-instances (resp. 35 job-instances with setup times) to 140 job-instances (resp. 100 job-instances). This study is presented in Section 2.1.

Chapter 2

State space relaxation algorithms

Contents

This chapter concerns deterministic optimization, and in particular two representative studies based on State-Space Relaxation techniques. In Section 2.1, an application of SSR techniques in the context of machine scheduling. Strong Lagrangian relaxations are derived from a Dynamic Programming formulation of the problem. They are used within an effective branch-and-bound algorithm. Section 2.2 describes the use of SSDP for the Temporal Knapsack Problem, which was developed during the PhD thesis of Gaël Guillot. It shows several problem-specific and generic improvements that need to be made to design a competitive solving method. An overview of other contributions in deterministic optimization is finally reported in Section 2.3.

2.1 Branch-and-bound algorithms: application to the flowshop problem

This section is based on the journal paper [Detienne *et al.* 2016]. We consider the flowshop problem on two machines with sequence-independent setup times to minimize total completion time. Large scale network flow formulations of the problem

are suggested together with strong Lagrangian bounds based on these formulations. To cope with their size, filtering procedures are developed. To solve the problem to optimality, we embed the Lagrangian bounds into two branch-and-bound algorithms. The best algorithm is able to solve all 100-job instances of our testbed with setup times and all 140-job instances without setup times, thus significantly outperforming the best algorithms in the literature. The outline of the section is as follows. We first precisely describe the problem and give an overview of the literature on the topic. Different dominance rules, which are both from the literature and new ones, are described in Section 2.1.1. In Section 2.1.2, we present network flow formulations for the problem, as well as a subgradient algorithm with embedded filtering procedures to obtain Lagrangian dual bounds. Two improved branch-and-bound algorithms for the problem are suggested in Section 2.1.3. Results of computational experiments with these algorithms are given in Section 2.1.4.

Contribution from a State-Space Relaxation perspective. This work is a successful application of SSR in the branch-and-bound setting. We base our study on the network flow formulation from [Akkan & Karabati 2004]. Using Lagrangian-cost variable fixing and integrating dominance rules into the structure of the network, we are able to use a larger network and obtain strong lower bounds. Those are also improved by enforcing the relaxed constraints locally using two- and three-cycle elimination, and altering the graph to remove dominated two-arc paths. The structure of the network allows us to compute an expensive Lagrangian dual bound only once at the root node, and then recompute the bound in linear time at every node of the enumeration tree. Thus, millions of nodes can be checked in a reasonable time.

The original paper [Detienne *et al.* 2016] gives a more extensive literature review. It also proposes a generalization of the mixed-integer linear programming formulation proposed in [Akkan & Karabati 2004] for the problem with setup times $F2|ST_{si}|\sum C_j$, whose linear relaxation is exploited within a preprocessing procedure. For the sake of conciseness, the proofs are omitted in this version of the study. Moreover, the dynamic programming and Lagrangian dual bound are presented in a slightly different way in order to emphasize the SSR methodology.

Problem description. We consider the problem of scheduling a set of jobs $J = \{1, \ldots, n\}$ in a two-machine flowshop with the objective of minimizing the sum of completion times of jobs. The jobs are available at time zero and they should be processed first on machine 1, and then on machine 2. Each machine can process at most one job at a time. Let p_j^m denote the processing time of job j on machine m, where $m = 1, 2$. All processing times are integer. Preemption of the processing of the jobs in not allowed on either machine. Let C_j^m denote the completion time of job j on machine m. According to the scheduling classification, the problem is denoted by $F2||\sum C_j$. It is known to be NP-hard in the strong sense [Garey *et al.* 1976]. It has been shown by Conway et al. [Conway *et al.* 1967] that there exists at least

one optimal solution where both machines have the same sequence of jobs. Thus, we may restrict the search to permutation schedules only.

In addition to this classic two-machine flowshop problem, we also consider its extension in which every job should be set up on each machine before being processed. Let s_j^m denote the setup time of job j on machine m. Setup of a job on machine 2 and processing of the same job on machine 1 can be performed in parallel. Note that setup times do not depend on the job processed just before job j, i.e. the setup times are *sequence independent*. This generalization of the problem has been treated previously in [Gharbi *et al.* 2013]. It can be denoted as $F2|ST_{si}|\sum C_j$ in the scheduling classification. The set of permutation schedules remains dominant for this generalization as indicated in [Allahverdi *et al.* 1999].

Related work. The problem $F2||\sum C_j$ has been studied in the literature for many years. First lower bounds and the branch-and-bound algorithms based on them were proposed by [Ignall & Schrage 1965]. Several Lagrangian relaxation-based lower bounds have been developed for the problem. The most successful one was described by [Akkan & Karabati 2004], who give a positional (assignment) formulation for the problem $F2||\sum C_j$. The authors use the notion of waiting time of a job before its processing starts on the second machine. The formulation has $O(n^2)$ variables and $O(n)$ constraints. In [Hoogeveen *et al.* 2006], it was shown experimentally that the lower bound one can obtain by solving the linear relaxation of the positional formulation is stronger than any other bound proposed previously in the literature. It was also shown that any Lagrangian relaxation does not improve this bound. In [Akkan & Karabati 2004], a network flow formulation for the problem was also suggested. In this network, each node corresponds to a position in the schedule and the waiting time of the job on this position. The network was then reduced by finding bounds on waiting times of jobs on different positions. To find a lower bound and design a branch-and-bound, the Lagrangian relaxation is used, in which job occurrence constraints are relaxed. Here the subproblem is the shortest path problem, and the Lagrangian dual problem is solved using a subgradient method. The branch-and-bound algorithm which uses this Lagrangian relaxation is able to solve instances with up to 60 jobs with small processing times (up to 10) and up to 45 jobs with large processing times (up to 100). Gharbi et al. [Gharbi *et al.* 2013] proposed several dual bounds for the problem $F2|ST_{si}|\sum C_j$. Some of the suggested lower bounding procedures are similar to those used for the problem without setup times. One lower bound is based on solving the linear relaxation of a positional formulation. Another lower bound is based on Lagrangian relaxation similar to one used in [Velde 1990]. Best exact algorithms based on the proposed dual bounds allowed the authors to solve all instances with up to 30 jobs and the majority of instances with 35 jobs with large processing and setup times (up to 100).

2.1.1 DP formulation and dominance rules

This section introduces a DP formulation of the problem, as well as dominance rules that are used in the branch-and-bound procedures to avoid exploring the set of solutions which are proved to be dominated, and in network reduction procedures to shrink the size of the networks.

Lag-based formulation of the objective function. This section generalizes the results of [Akkan & Karabati 2004] for the problem with setup times $F2|ST_{si}|\sum C_j$. Note that the setup time of any job on machine 1 can be integrated into its processing time on machine 1. This follows from the fact that there exists an optimal schedule in which machine 1 process jobs without idle time. So, without loss of generality, for all $j \in J$, we can set $s_j^1 = 0$, and adjust appropriately processing times p_j^1. In the following, $[k]$ denotes the index of the job in position k. Assuming the convention that $C_{[0]}^1 = C_{[0]}^2 = 0$, the completion times $C_{[k]}^m$ of the job in position k, $k \in J$, on machines $m = 1, 2$ can be computed as:

$$\begin{aligned} C_{[k]}^1 &= C_{[k-1]}^1 + p_{[k]}^1. \\ C_{[k]}^2 &= \max\{C_{[k]}^1, C_{[k-1]}^2 + s_{[k]}^2\} + p_{[k]}^2. \end{aligned}$$

In [Akkan & Karabati 2004], the authors introduced the notion of time lag between the processing of the same job on both machines to write a positional model and a network flow model for the problem. This kind of models is also called *waiting time-based* models in [Gharbi *et al.* 2013]. The completion-to-completion lag L_k^c of the job in position k, $k \in J$ is defined as the time elapsed between the completion of the job on machine 1 and its completion on machine 2:

$$\begin{aligned} L_k^c &= C_{[k]}^2 - C_{[k]}^1 \\ &= \max\{0, L_{k-1}^c + s_{[k]}^2 - p_{[k]}^1\} + p_{[k]}^2. \end{aligned}$$

The completion-to-start lag L_k^s of the job in position k, $k \in J$, is defined as the time elapsed between the completion of the job on machine 1 and its start on machine 2:

$$L_k^s = L_k^c - p_{[k]}^2 = \max\{0, L_{k-1}^s + p_{[k-1]}^2 + s_{[k]}^2 - p_{[k]}^1\}.$$

The objective function can finally be rewritten as:

$$\begin{aligned} \sum_k C_{[k]}^2 &= \sum_k (C_{[k]}^1 + L_k^c) \\ &= \sum_k \left((n-k+1)p_{[k]}^1 + L_k^s + p_{[k]}^2\right) \\ &= \sum_k \left((n-k+1)p_{[k]}^1 + L_k^s\right) + \sum_{j \in J} p_j^2. \end{aligned} \tag{2.1.1}$$

Dynamic programming formulation. The problem can also be modeled with help of the following Dynamic Programming equations:

$$\begin{cases} \overrightarrow{\Phi}(0,0,\mathbf{0}) = & 0 \\ \overrightarrow{\Phi}(k,\ell,\boldsymbol{u}) = & \min\left\{\overrightarrow{\Phi}(k-1,\ell',\boldsymbol{u}-\varepsilon_j) + (n-k+1)p_j^1 + \ell : \right. \\ & \qquad\qquad \left. j \in J,\ \ell = \max\{0, \ell' + s_j^2 - p_j^1\}\right\}, \\ & \qquad\qquad k \in \{1,\dots,\eta\}, t \in \{0,\dots,d_k\}, \boldsymbol{u} \in \mathbb{Z}^n \\ \overrightarrow{\Phi}(k,\ell,\boldsymbol{u}) = & \infty \quad \text{otherwise.} \end{cases}$$

In these recurrence equations, $\overrightarrow{\Phi}(k,\ell,\boldsymbol{u}), k \in \{1,\dots,\eta\}, \ell \in \mathbb{N}$, $\boldsymbol{u} \in \mathbb{Z}^n$ denotes the minimum possible cost of a sequence of k jobs, where the completion-to-start lag of the last job is ℓ, and each job j has been processed exactly u_j times. Note that this value anticipates the cost for subsequent jobs that is only due to the jobs already sequenced (more precisely, their completion time on the first machine). Vector ε_j is the j^{th} canonical vector, whose components are all equal to zero except the j^{th}, which equals one. The first relation states initial conditions: If no job is processed at the beginning of the horizon, the cost is null. The second equation keeps the best among one choice per job: Processing job j at position k completing at lag ℓ adds $(n-k+1)p_j^1 + \ell$ to the objective value. The last relation corresponds to infeasible states. The optimal value is obtained by calculating $\min_{\ell\in\mathbb{N}} \overrightarrow{\Phi}(n,\ell,\mathbf{1}) + \sum_{j\in J} p_j^2$, which would lead to straightforward $\mathcal{O}(2^n n \sum_j p_j^2)$-time procedure.

Dominance rules. We now describe dominance rules which allow us to speed up the solution process. As specified in the corresponding sections, they are used in the branch-and-bound procedures to avoid exploring the set of solutions which are proved to be dominated, and in network reduction procedures to shrink the size of the networks. To make them consistent with the network approach, the dominance rules are expressed in terms of waiting time, or lag variables. The next proposition gives a generalization of the dominance rule from [Velde 1990] for the problem $F2||\sum C_j$.

Proposition 2.1.1 ([Allahverdi 2000]). *If jobs i and j satisfy $p_i^1 + s_j^2 \le p_j^1 + s_i^2$, $p_i^2 + s_i^2 \le p_j^2 + s_j^2$, and $p_j^2 \le p_i^2$, then there exists an optimal schedule in which job i precedes job j.*

In our solution methods, Proposition 2.1.1 is used as preprocessing to determine a set of predecessors Γ_i^- and successors Γ_i^+ for each job i. The next proposition is an extension of the dominance rule of [Croce *et al.* 1996] to the case with sequence-independent setup times.

Proposition 2.1.2. *Let σ be a partial schedule, and i and j two jobs not in σ. Let us denote by C_σ^m the completion time of σ on machine $m \in \{1,2\}$. If $p_i^1 \le p_j^1$, $p_i^2 \ge p_j^2$, $s_i^2 \ge s_j^2$, and*

$$\max\{C_\sigma^1 + p_i^1, C_\sigma^2 + s_i^2\} + p_i^2 \le \max\{C_\sigma^1 + p_j^1, C_\sigma^2 + s_j^2\} + p_j^2,$$

then there exists an optimal solution not headed by σj, where σj denotes σ immediately followed by j.

Proposition 2.1.2 can be translated in terms of completion-to-completion lag, by substituting $L^c_\sigma = C^2_\sigma - C^1_\sigma$:

Proposition 2.1.3. *Let σ be a partial schedule, L^c_σ the corresponding completion-to-completion lag, and i and j two jobs not in σ. If $p^1_i \leq p^1_j$, $p^2_i \geq p^2_j$, $s^2_i \geq s^2_j$, and $\max(p^1_i, L^c_\sigma + s^2_i) + p^2_i \leq \max(p^1_j, L^c_\sigma + s^2_j) + p^2_j$, then there exists an optimal solution not headed by σj.*

Let us consider a partial schedule, built up to position $k-1$. For a position k, two jobs a and b and $\ell \in \{k, k+1\}$, let $L^c_l(k, a \to b)$ denote the time lag between the completion times of $[\ell]$ on machines 1 and 2, under the assumption that a is processed at position k and b at position $k+1$ (i.e. $[k] = a$ and $[k+1] = b$):

$$\begin{aligned} L^c_k(k, a \to b) &= \max\{p^1_a, L^c_{k-1} + s^2_a\} + p^2_a - p^1_a \\ L^c_{k+1}(k, a \to b) &= \max\{p^1_b, L^c_k + s^2_b\} + p^2_b - p^1_b \end{aligned}$$

Let us define $f(k, a \to b)$ as the contribution of jobs a and b to the total cost expressed in the objective function (2.1.1), when they are scheduled at positions k and $k+1$ respectively:

$$f(k, a \to b) = (n-k+1)p^1_a + L^c_k(k, a \to b) + (n-k)p^1_b + L^c_{k+1}(k, a \to b)$$

In the same way, one can define $L^c_\ell(k, \sigma)$ and $f(k, \sigma)$, which are the completion-to-completion lag in position ℓ and the partial cost, respectively, when scheduling a sequence of jobs σ starting at position k:

$$L^c_\ell(k, \sigma) = \begin{cases} \max\{p^1_{\sigma(1)}, L^c_{k-1} + s^2_{\sigma(1)}\} + p^2_{\sigma(1)} - p^1_{\sigma(1)} & \text{if } \ell = k \\ \max\{p^1_{\sigma(\ell-k+1)}, L^c_{\ell-1}(k, \sigma) + s^2_{\sigma(\ell-k+1)}\} + p^2_{\sigma(\ell-k+1)} - p^1_{\sigma(\ell-k+1)} & \\ \qquad \text{if } k < \ell < |\sigma| + k \end{cases}$$

$$f(k, \sigma) = \sum_{\ell=k}^{k+|\sigma|-1} (n-\ell+1)p^1_{\sigma(\ell-k+1)} + \sum_{\ell=k}^{k+|\sigma|-1} L^c_\ell(k, \sigma)$$

The next proposition may be useful to improve a bound and/or to reduce the size of networks which will be presented below. It shows that, under some conditions, schedules in which a job j at position k precedes a job i can be discarded.

Proposition 2.1.4 (Dominance on two consecutive jobs)**.** *If $f(k, i \to j) < f(k, j \to i)$ and $L^c_{k+1}(k, i \to j) \leq L^c_{k+1}(k, j \to i)$, then jobs j and i are not processed at positions k and $k+1$, respectively, in any optimal solution.*

The next proposition is used in our branch-and-bound procedures, as well as for removing edges in the networks (see Section 2.1.2). In the latter context, the constraint specifying that each job must be scheduled not more than once is relaxed. Hence, Proposition 2.1.5 is described in such a form that job repetition is allowed in partial sequences.

Proposition 2.1.5 (Dominance on K consecutive jobs)**.** *Let $\sigma = (\sigma_1, \ldots, \sigma_l)$ be a partial sequence of jobs starting at position k, and let σ' be a permutation of σ. If at least one of these conditions hold, then no optimal schedule contains partial sequence σ of jobs starting at position k:*

- *$\sigma_i = \sigma_j$ for some $i \neq j$.*
- *$\sigma_j \in \Gamma^-_{\sigma_i}$ for some $i < j$.*
- *$f(k, \sigma') < f(k, \sigma)$ and $L^c_{k+|\sigma|-1}(k, \sigma') \leq L^c_{k+|\sigma|-1}(k, \sigma)$.*

Consistency of the different rules We should be careful in applying several dominance rules at the same time so that their consistency is ensured. For example, it can happen that one rule eliminates such schedules where job i precedes job j, whereas another rule forbids job j to precede job i. To ensure that at least one optimal schedule is not eliminated, we use a modified version of Proposition 2.1.3 where the condition $\max(p^1_i, L^c_\sigma + s^2_i) + p^2_i \leq \max(p^1_j, L^c_\sigma + s^2_j) + p^2_j$ is replaced by $\max(p^1_i, L^c_\sigma + s^2_i) + p^2_i < \max(p^1_j, L^c_\sigma + s^2_j) + p^2_j$. That way, any deduction made by Propositions 2.1.3, 2.1.4 or 2.1.5 removes only non-optimal schedules. Thus, when applying Proposition 2.1.1, we can safely break ties according to the index of the jobs.

2.1.2 Network flow formulations and lower bounds

In this section, we introduce two minimum cost flow formulations to obtain tight lower bounds. The first one can be obtained from the DP model (2.1.2) by the method exposed in Section 1.2.2.1. In order to get stronger bounds and solve larger instances, we also design an extended network, based on an augmented DP model by adding redundant information into the state-space.

2.1.2.1 Basic network G_1

Our first lower bound is based on a transshipment type network, which is a directed graph $G_1 = (V_1, A_1)$ whose structure is identical to the one proposed in [Akkan & Karabati 2004]:

- Each node $v_{k,\ell} \in V_1$ of the network is associated with a position k in the sequence, and a value ℓ of the completion-to-completion lag of the job in position $k-1$. Node $v_{1,0}$ is the source of the network. A sink node $v_{n+1,0}$ is also added, which represents the end of the schedule.
- For each combination of job j, position k, and completion-to-completion lag ℓ, there is an arc $(v_{k,\ell}, v_{k+1,\ell'}, j) \in A_1$ from node $v_{k,\ell}$ to node $v_{k+1,\ell'}$. Here $\ell' = \max\{0, \ell + s^2_j - p^1_j\} + p^2_j$ if $k < n$, and $\ell' = 0$ if $k = n$. This arc represents the processing of job j in position k, when the completion-to-completion lag of the previous job is equal to ℓ, so that job j ends with a completion-to-completion

lag equal to ℓ'. Note that multiple arcs representing different jobs may connect the same pair of nodes. Following the expression of the objective function given by (2.1.1), the cost $c(v_{k,\ell}, v_{k+1,\ell'}, j)$ of using this arc is $(n-k+1)p_j^1 + \ell'$ when $k < n$, and $c(v_{n,\ell}, v_{n+1,0}, j) = p_j^1 + \max\{0, \ell + s_j^2 - p_j^1\} + p_j^2$.

Basic network flow formulation The scheduling problem can be seen as the problem of finding a minimum cost flow of value 1 (a path) from the source node to the sink node, going through exactly one arc associated with each job, leading to the following ILP model:

$$\min \sum_{(v,w,j)\in A_1} c(v,w,j) \cdot x_{v,w,j} \tag{2.1.2}$$

$$s.t. \sum_{(v,w,j)\in A_1} x_{v,w,j} = \sum_{(w,v,j)\in A_1} x_{w,v,j} \qquad \forall v \in V_1 - \{v_{1,0}, v_{n+1,0}\} \tag{2.1.3}$$

$$\sum_{(v,w,j)\in A_1} x_{v,w,j} \leq 1 \qquad \forall j = 1,\ldots,n \tag{2.1.4}$$

$$\sum_{(v_{1,0},w,j)\in A_1} x_{v_{1,0},w,j} = 1 \tag{2.1.5}$$

$$x_{v,w,j} \in \{0,1\} \qquad \forall (v,w,j) \in A_1 \tag{2.1.6}$$

Network reduction during its creation In order to reduce the size of the network, we use a procedure similar to the one described in [Akkan & Karabati 2004]:

- An upper bound $\bar{z}$ on the optimum value of the problem is computed. Instead of the meta-heuristic of [Croce *et al.* 1996], we use the iterated enhanced dynasearch heuristic of [Tanaka 2011], with a straightforward adaptation to handle setup times. The time taken by this heuristic is given in Table 2.1.1.

- A modified version of the adapted positional MILP model is used to derive upper bounds on the lag values at each position, in all schedules whose cost is not larger than $\bar{z}$. Bounds on completion-to-completion lags allow one to remove dominated nodes. Bounds on completion-to-start lags and on the sum of those for consecutive pairs of positions allow one to remove dominated arcs. See [Akkan & Karabati 2004] for details. In our implementation, the bounds are improved by removing variables x_{jk} from the model when job j cannot be processed in position k from the dominance rules, that is, $k \leq |\Gamma_j^-|$ or $k \geq n - |\Gamma_j^+|$.

- For each node v, we keep track of the set of jobs $\mathcal{P}(v)$ that exist in all paths from the source node to v, and utilize it when creating arcs out of v as follows:

  - Similar to [Akkan & Karabati 2004], if $j \in \mathcal{P}(v)$, then no arc is created for job j out of v.

  - If $i \in \mathcal{P}(v)$ and $j \in \Gamma_i^-$, then no arc is created for j out of v.

- For each node v, we keep track of the set of jobs $\bar{\mathcal{P}}(v)$ that exist in at least one path from the source to v, and utilize it when creating arcs out of v as follows:
  - If $i \notin \bar{\mathcal{P}}(v)$ and $i \in \Gamma_j^-$, then no arc is created for j out of v.
  - If $i \notin \bar{\mathcal{P}}(v_{k,\ell})$ and j is dominated by i at lag ℓ and position k in the sense of Proposition 2.1.3, then no arc is created for j out of v.

Table 2.1.1: Time (in seconds) required for computing the initial upper bound using heuristic from [Tanaka 2011].

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$ (instances of [Gharbi *et al.* 2013])			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	3.8	13.2	29.1	59.5				
$[1-100]$	5.2	16.6	38.1	80.0	27.5	43.4	65.4	129.4

As a preliminary experiment, we directly apply an MILP solver to formulation (2.1.2)–(2.1.6) based on network G_1 which is reduced as described just above. Table 2.1.2 reports the number of solved instances (without setup times) within the time limit. Table 2.1.5 reports the average number of nodes and arcs in graph G_1.

Durations	n=10	n=30	n=40	n=50	n=60
$[1,10]$	20	20	20	19	18
$[1,100]$	20	15	2	0	0

Table 2.1.2: Number of instances of $F_2||\sum C_i$ solved to optimality within 1000 seconds using formulation (2.1.2)-(2.1.6) based on G_1.

Lagrangian lower bound by state-space relaxation. The authors of [Akkan & Karabati 2004] computed a Lagrangian lower bound based on a similar network. It is used within a branch-and-bound procedure, as well as some dominance rules to solve the problem. Relaxing the constraints (2.1.4) to the objective function leads to the following Lagrangian subproblem:

$$L_1(\pi) = \min \left\{ \sum_{(v,w,j)\in A_1} (c(v,w,j) + \pi_j)\, x_{v,w,j} - \sum_{j=1}^{n} \pi_j \,\middle|\, (2.1.3), (2.1.5), (2.1.6) \right\}$$

For any non-negative vector of Lagrange multipliers π, $L_1(\pi)$ is a Lagrangian lower bound on the optimum of the problem, which can be computed by solving a simple

shortest path problem in G_1 with modified costs $c(v, w, j) + \pi_j$. This problem can be solved in $O(|A_1|)$-time complexity.

To improve bound $L_1(\pi)$, the constraint forbidding immediate repetition of a same job is added to the Lagrangian subproblem (this technique is sometimes called two-cycle elimination in the vehicle routing community [Desrochers *et al.* 1992]). We can take into account this constraint without increasing the complexity of the shortest path algorithm [Abdul-Razaq & Potts 1988, Peridy *et al.* 2003].

Like in [Tanaka *et al.* 2009], we obtain near-optimal Lagrangian multipliers employing the conjugate subgradient algorithm [Sherali & Ulular 1989, Sherali & Lim 2007]. At each iteration k of the procedure, given current Lagrangian multipliers π^k, we first compute $L_1(\pi^k)$. We denote by x^{*k} the corresponding optimal solution of the Lagrangian subproblem. We choose the subgradient vector as $(g_j^k = 1 - \sum_{(v,w,j)\in A_1} x^{*k}_{v,w,j})_{j\in\{1,\dots,n\}}$. The Lagrangian multipliers are updated by

$$d^k = g^k + \xi^k d^{k-1}$$
$$\pi^{k+1} = \pi^k + \gamma^k \frac{UB - L_1(\pi^k)}{||d^k||^2} d^k$$

Following [Sherali & Ulular 1989, Sherali & Lim 2007], we choose $\xi^k = ||g^k||/||d^{k-1}||$. The choice of the step size parameter γ^k obeys these rules:

- The initial value is $\gamma^0 = \gamma^{ini}$.
- It is decreased by $\xi^k = \kappa_S \gamma^{k-1}$ if the best lower bound is not updated for δ_S successive iterations.
- It is increased by $\xi^k = \kappa_E \gamma^{k-1}$ if the best lower bound is improved at iteration k.

The algorithm is terminated if the best lower bound does not increase by $100\varepsilon/(1-\varepsilon)\%$ and the gap between the best lower and upper bounds does not decrease by $100\varepsilon\%$ in δ_T successive iterations, but not before min_{iter} iterations have been completed.

Filtering procedure for G_1 In order to reduce further the size of network G_1, we developed Lagrangian cost variable fixing (see Section 1.2.3.4, [Ibaraki & Nakamura 1994, Tanaka *et al.* 2009]). Given a vector of Lagrange multipliers π,

- let $F^1(v, \pi)$ be the cost of the shortest path from the source to node v in G_1 with modified costs;
- let $B^1(v, \pi)$ be the cost of the shortest path from node v to the sink in G_1 with modified costs.

Given π, values $F^1(v, \pi)$ for all nodes $v \in V_1$ can be found using the forward dynamic programming algorithm, and all values $B^1(v, \pi)$, $v \in V_1$ can be found using backward dynamic programming algorithm. Then an arc $(v, w, j) \in A_1$ can be removed from

G_1 if $F^1(v,\pi) + c(v,w,j) + \pi_j + B^1(w,\pi) \geq \bar{z}$. Furthermore, a node $v \in V_1$ can be removed from G_1 if it does not have any incoming or any outgoing arc anymore. For a fixed vector π, the time complexity of this filtering procedure remains equal to $O(|A_1|)$. The filtering procedure is called at every iteration of the conjugate subgradient algorithm.

The running time of the subgradient algorithm is presented in Table 2.1.3. The relative gaps obtained after the subgradient algorithm are shown in Table 2.1.4. The gap is computed as $\frac{UB-LB}{LB}$, where UB is the upper bound given by the heuristic [Tanaka 2011], and LB is the best Lagrangian bound obtained during the subgradient algorithm. One can see from the results that a very tight lower bound is obtained in small running time, which is 25 seconds for the largest instances without setup times. The computing time is twice larger for instances with setup times. This is explained by the slightly degraded performance of the dynasearch procedure for this class of problems to compute an upper bound (the root gap is twice larger). Hence, the network cannot be reduced as much in this case (as can be seen from Table 2.1.5), so that solving the subproblem at each iteration of the subgradient procedure takes more time.

Table 2.1.3: Average running time (in seconds) for the subgradient procedure on network G_1.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.2	0.4	0.8	1.7				
$[1-100]$	1.1	4.2	10.5	23.5	9.0	16.1	24.7	55.9

Table 2.1.4: Average duality gap produced by the subgradient procedure on network G_1.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.13%	0.11%	0.10%	0.08%				
$[1-100]$	0.16%	0.18%	0.12%	0.10%	0.21 %	0.22 %	0.20 %	0.20 %

From Table 2.1.5 it can be seen that the filtering procedure reduces the size of the graph G_1 significantly: the number of arcs is decreased by a factor of 5 on large instances without setup times, and by more than 2 on instances with setup times. Once again, the degraded results can be attributed to the quality of the upper bound used in the filtering procedure.

We applied an MILP solver to formulation (2.1.2)–(2.1.6) based on filtered network G_1'. Table 2.1.6 reports the number of solved instances (without setup times) within the time limit. Again the improvement of results in comparison with the

Table 2.1.5: Average size of network G_1 before and after filtering.

Number of nodes in G_1 (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.8	1.3	2.0	2.8				
$[1-100]$	8.0	14.6	21.3	30.4	20.3	26.1	31.9	46.3
Number of arcs in G_1(in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	21.9	59.3	119.0	213.9				
$[1-100]$	258.3	731.7	1451.6	2635.2	1026.4	1553.3	2189.3	4030.1
Number of nodes in G_1 after filtering (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.5	0.9	1.4	1.9				
$[1-100]$	4.4	9.1	14.2	21.0	14.9	20.3	24.9	38.6
Number of arcs in G_1 after filtering (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	3.4	11.3	26.2	47.0				
$[1-100]$	30.9	129.8	264.0	520.6	311.6	567.6	828.6	1813.3

network flow formulation based on G_1 is very limited despite a significant reduction of the graph size.

Table 2.1.6: Number of instances of $F2||\sum C_i$ solved to optimality within 1000 seconds using formulation (2.1.2)-(2.1.6) based on G_1'.

Durations	n=10	n=30	n=40	n=50	n=60
$[1,10]$	20	20	20	19	19
$[1,100]$	20	19	9	2	1

2.1.2.2 Expanded network G_2

Although lower bounds obtained using Lagrangian relaxation of the network flow formulation based on G_1 are already very tight, we can improve them by adding another dimension to G_1. It results in a larger network G_2, which allows us to eliminate more dominated subsequences of jobs in the Lagrangian subproblem and thus to improve the lower bound.

Network structure of G_2. The network is a directed graph $G_2 = (V_2, A_2)$ with the following structure.

- Each node $v_{k,\ell,i} \in V_2$ of the network is associated with a position k in the sequence, a job i, and a value ℓ of the completion-to-completion lag of the job in position $k-1$. Node $v_{0,0,0}$ is the source of the network. An additional sink node $v_{n+1,0,0}$ is added, which represents the end of the schedule.
- For each combination of jobs i, j, $i \neq j$, position k, and completion-to-completion lag ℓ, there is an arc $(v_{k,\ell,i}, v_{k+1,\ell',j}) \in A_2$, with:

$$\ell' = \begin{cases} 0 & \text{if } k = 0 \\ \max\{0, \ell + s_i^2 - p_i^1\} + p_i^2 & \text{otherwise} \end{cases}$$

  This arc represents the processing of job i in position k, when the completion-to-completion lag of the job in position $k-1$ is equal to ℓ, the completion-to-completion lag of job i is equal to ℓ', and job j is processed at position $k+1$. When $0 < k < n$, the cost $c(v_{k,\ell,i}, v_{k+1,\ell',j})$ of using this arc is $(n-k+1)p_i^1 + \ell'$. For the first position, $c(v_{0,0,0}, v_{1,0,j}) = 0$. For the last position, $c(v_{n,\ell,i}, v_{n+1,0,0}) = p_i^1 + \max\{0, \ell + s_i^2 - p_i^1\} + p_i^2$.

Network reduction during its creation.

- A node $v_{k,\ell,i}$ in G_2 is created only if an arc $(v_{k,\ell}, v_{k+1,\ell'}, i)$ exists in filtered network G_1'.
- Arc $(v_{k,\ell,j}, v_{k+1,\ell',i})$ is not created if scheduling job j is dominated at lag ℓ by Proposition 2.1.3.
- Arc $(v_{k,\ell,j}, v_{k+1,\ell',i})$ is not created if scheduling of jobs j and i at positions k and $k+1$, respectively, is dominated by Proposition 2.1.4.
- Arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$ is not created if, for all arcs $(v_{k-1,l_0,j_0}, v_{k,l_1,j_1})$, the sequence of jobs (j_0, j_1, j_2) is dominated at lag l_0 and position $k-1$ according to Proposition 2.1.5.

2.1.2.3 Filtering procedure for G_2

We developed a filtering procedure embedded in the subgradient algorithm. The procedure is similar to the one described in Section 2.1.2.1, but has the following differences.

- The Lagrangian lower bound is improved using 3-cycle elimination by forbidding such partial sequences of job as (i, j, i) (adding this constraint does not change the time-complexity of the Lagrangian subproblem [Abdul-Razaq & Potts 1988, Peridy *et al.* 2003]).
- Similar to the filtering procedure in Section 2.1.2.1, Lagrangian cost fixing is performed by computing the following values.

  - $F^2(v,\pi)$ is the cost of the shortest path from the source to node v in G_2 with modified costs;
  - $B^2(v,\pi)$ is the cost of the shortest path from node v to the sink in G_2 with modified costs.

  An arc $(v,w) \in A_2$ can be removed from G_2 if $F^2(v,\pi) + c(v,w) + \pi_j + B^2(w,\pi) \geq \bar{z}$, where j is the index of the job represented by w.

- In addition, Proposition 2.1.5 is applied to perform what we call 3-consecutive-jobs filtering. It removes arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$ if one of these conditions holds:
  - for all arcs $(v_{k-1,l_0,j_0}, v_{k,l_1,j_1})$ in G_2, the sequence of jobs (j_0, j_1, j_2) is dominated at lag l_0 and position $k-1$ according to Proposition 2.1.5 ;
  - for all arcs $(v_{k+1,l_2,j_2}, v_{k+2,l_3,j_3})$ in G_2, the sequence of jobs (j_1, j_2, j_3) is dominated at lag l_1 and position k according to Proposition 2.1.5.

  This 3-consecutive-jobs filtering is costly. Therefore, it is not applied at every iteration of the subgradient procedure, but each time the number of arcs in the graph is reduced by 5% using the Lagrangian cost fixing since the last time 3-consecutive-jobs filtering was used.

- The lower bound is improved using Proposition 2.1.5 by removing dominated sequences of three jobs which are part of the Lagrangian subproblem solution. More precisely, at each iteration of the subgradient procedure, the Lagrangian subproblem solution is inspected. Assume that a subsequence of jobs (j_1, j_2, j_3) starting at lag l_1 and position k is part of the solution and dominated. Let $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2}, v_{k+2,l_3,j_3})$ be the corresponding sequence of nodes. This path is removed from the network using the following procedure (see Figure 2.1.1):
  1. Identify the set $V^*(k,l_1,j_1,l_2,j_2)$ of nodes v' which are successors of v_{k+1,l_2,j_2} and such that sequence $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2}, v')$ is not dominated (by inspection).
  2. Create a new node v'', which is a duplicate for node v_{k+1,l_2,j_2}.
  3. For each node $v' \in V^*(k,l_1,j_1,l_2,j_2)$, create an arc (v'',v'), which is a duplicate for (v_{k+1,l_2,j_2}, v').
  4. Create an arc (v_{k,l_1,j_1}, v'').
  5. Remove arc $(v_{k,l_1,j_1}, v_{k+1,l_2,j_2})$.

  The Lagrangian subproblem is then solved again, on the modified network and with the same Lagrange multipliers. This procedure is repeated until there exist no dominated sequences of three jobs in the Lagrangian subproblem solution. This solution is then returned to the sub-gradient procedure.

- At the end of the subgradient procedure, the following additional filtering procedure is used, as in [Detienne *et al.* 2012]. Given a vector of Lagrange multipliers π,

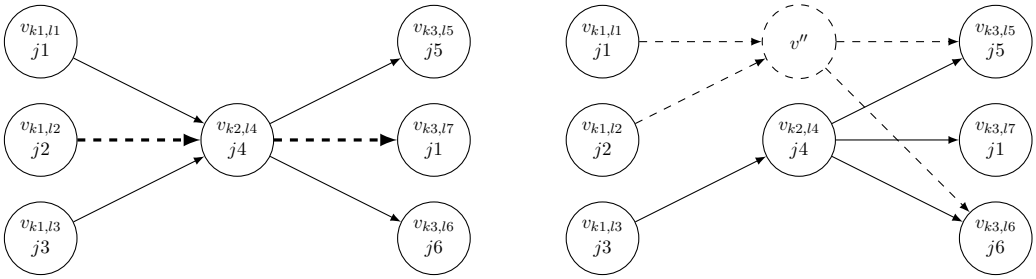


Figure 2.1.1: On the left side, assume that the bold dashed path $(v_{k_1,l_2,j_2}, v_{k_2,l_4,j_4}, v_{k_3,l_7,j_1})$ is part of the optimal solution of the current Lagrangian subproblem, and is dominated in the sense of Proposition 2.1.5. On the right side, this path is removed from the graph by adding a duplicate node v'' for v_{k_2,l_4,j_4} and rerouting non-dominated paths through the new node. Path $(v_{k_1,l_1,j_1}, v_{k_2,l_4,j_4}, v_{k_3,l_7,j_1})$ is also removed since it is not feasible.

- let $F^2_j(v,\pi)$ (resp. $B^2_j(v,\pi)$) denote the cost of the shortest path from the source to node v (resp. from node v to the sink) in G_2 with Lagrangian costs, such that this path goes through exactly one arc representing job j;
- let $F^2_{\neg j}(v,\pi)$ (resp. $B^2_{\neg j}(v,\pi)$) denote the cost of the shortest path from the source to node v (resp. from node v to the sink) in G_2 with Lagrangian costs, such that this path does not contain any arc representing job j.

Given π, these values for all jobs and all nodes can be computed in time $O(n|A_2|)$ by applying several times the forward and backward dynamic programming algorithms. Then for each arc $a = (v_{k,\ell,j}, v_{k+1,\ell',i}) \in A_2$, we compute value $L^2(a,\pi)$ as:

$$L^2(a,\pi) = \max \left\{ \begin{array}{l} F^2_{\neg j}(v_{k,\ell,j},\pi) + c(a) + \pi_j + B^2_{\neg j}(v_{k,\ell',i},\pi), \\ \max\limits_{j' \neq j} \left\{ \min \left\{ \begin{array}{l} F^2_{j'}(v_{k,\ell,j},\pi) + c(a) + \pi_j + B^2_{\neg j'}(v_{k,\ell',i},\pi), \\ F^2_{\neg j'}(v_{k,\ell,j},\pi) + c(a) + \pi_j + B^2_{j'}(v_{k,\ell',i},\pi) \end{array} \right. \right\} \end{array} \right\}.$$

Arc a can be removed from G_2 if $L^2(a,\pi) > \bar{z}$.

The running time of the subgradient algorithm with embedded filtering of graph G_2 is presented in Table 2.1.7. The relative gaps obtained after the subgradient algorithm are shown in Table 2.1.8. One can see from the results that the gap is decreased by 30% in comparison with the subgradient procedure on network G_1. However, the running time is increased by a factor of 5 for instances without setup times, and 16 with setup times.

From Table 2.1.9 it can be seen that, again, the filtering procedure reduces the size of the graph significantly for instances without setup times: the number of arcs is decreased by a factor of 4 on large instances. For instances with setup times, the number of arcs is still very large. In Section 2.1.3, we show that filtering is also efficient for this class of instances when used with a tighter upper bound.

Table 2.1.7: Time in seconds for the subgradient procedure on network G_2.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.2	0.7	2.5	8.9				
$[1-100]$	1.3	9.8	38.1	94.1	68.0	167.7	291.0	913.5

Table 2.1.8: The duality gap produced by the subgradient procedure on network G_2.

	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.07%	0.05%	0.06%	0.06%				
$[1-100]$	0.09%	0.07%	0.08%	0.07%	0.19%	0.20 %	0.19 %	0.19%

2.1.3 Branch-and-bound algorithms

We have implemented two branch-and-bound algorithms.

- Algorithm BB_1 is based on network G_1 and Lagrangian bound $L^1(\pi)$.
- Algorithm BB_2 is based on network G_2 and Lagrangian bound $L^2(\pi)$.

The following parts of the algorithm are the same for both BB_1 and BB_2.

- The initial upper bound is computed by the dynasearch heuristic from [Tanaka 2011].
- Network G_1 or G_2 is constructed and reduced by the corresponding filtering algorithm.
- After the subgradient algorithm, the vector of multipliers π^* which gives the best Lagrangian lower bound $L^1(\pi^*)$ or $L^2(\pi^*)$ is fixed till the branch-and-bound termination.
- The set of possible job sequences is explored, by enumerating the set of feasible (with respect to the job assignment constraint) paths in graph G_1 or G_2. We proceed from the source to the sink in the graph. For each node v, the outgoing arcs (v, w) are sorted in non-decreasing order of $B^1(w, \pi^*)$ or $B^2(w, \pi^*)$. The algorithms use the depth-first-search rule in this order.
- We use Proposition 2.1.5 in a Memory Dominance Rule fashion [Baptiste *et al.* 2004, T'kindt *et al.* 2004, Kao *et al.* 2008]: the set of non-dominated subsequences explored is maintained in a hash map. At each node of the tree, the current subsequence is tested against the subsequences composed of the same set of jobs.

Table 2.1.9: Average size of network G_2 before and after filtering. Filtering is performed with the initial upper bound.

Number of nodes in G_2 (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	2.3	7.8	17.0	35.8				
$[1-100]$	26.5	92.7	212.4	391.3	246.7	426.9	608.4	1 234.1
Number of arcs in G_2 (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	12.9	68.2	217.6	642.7				
$[1-100]$	164.2	937.0	2925.4	6431.4	3818.3	8224.6	13 550.5	35 554.8
Number of nodes in G_2 after filtering (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.4	2.2	6.6	13.0				
$[1-100]$	5.2	35.5	92.5	166.2	163.7	284.8	396.8	766.3
Number of arcs in G_1 after filtering (in thousands)								
	$F2\|\|\sum C_i$				$F2\|ST_{si}\|\sum C_i$			
Duration	n=40	n=60	n=80	n=100	n=60	n=70	n=80	n=100
$[1-10]$	0.8	7.6	38.6	99.2				
$[1-100]$	16.4	170.7	639.0	1465.4	1866.5	4236.0	6931.7	18 544.7

- At each node of the search tree, we maintain incrementally the number of unscheduled predecessors of each job. An arc in G_1 or G_2 corresponding to job j is a candidate for branching only if the number of remaining predecessors of j is zero.

- We branch, i.e. we extend current partial sequence σ with job j, only if the subsequence of the last K jobs ($K = 5$ in our implementation) including j is not dominated, according to Proposition 2.1.5, by any permutation of these K jobs.

- The upper bound may be updated only when a leaf node of the search tree is reached. We do not use any heuristics within the branch-and-bound algorithm.

2.1.3.1 Algorithm BB_1

In algorithm BB_1, at each non-root node of the search tree corresponding to node v in graph G_1, we compute lower bound $L^1(v)$. Let σ be the partial sequence built

so far. Then,

$$L^1(v) = cost(\sigma) + B^1(v, \pi^*) - \sum_{j \notin \sigma} \pi_j^*.$$

Computation of $L^1(v)$ can be done in constant time by incrementally maintaining at each node the current value of $\sum_{j \notin \sigma} \pi_j$.

Note that algorithm BB_1 is similar to the branch-and-bound algorithm presented in [Akkan & Karabati 2004]. The main differences are that graph G_1 is filtered, and we use Proposition 2.1.5 as a memory dominance rule. In [Akkan & Karabati 2004], other dominance rules are used, but preliminary computational experiments indicated that they do not improve the performance of our branch-and-bound procedure.

2.1.3.2 Algorithm BB_2

In algorithm BB_2, at each non-root node of the search tree corresponding to node v in graph G_2, we compute lower bound $L^2(v)$. Again, let σ be the partial sequence built so far. Then,

$$L^2(v) = cost(\sigma) + \max \left\{ \begin{array}{l} B^2(v, \pi^*), \\ \max_{j \notin \sigma} B^2_j(v, \pi^*), \\ \max_{j \in \sigma} B^2_{\neg j}(v, \pi^*) \end{array} \right\} - \sum_{j \notin \sigma} \pi_j^*.$$

Computation of $L^2(v)$ can be done in time $O(n)$.

Tentative upper bound As one can expect from Tables 2.1.8 and 2.1.9, solving large instances of the problem $F2|ST_{SI}|\sum C_i$ by enumerating the set of paths in network G_2 is very time-consuming. The optimal objective values of medium size instances indicate us that the very large size of the network in this case compared with the case without setup times clearly comes from a degraded initial upper bound. That is why, following the idea described in [Tanaka *et al.* 2009], we introduce the use of a tentative upper bound around algorithm BB_2. Contrary to the initial upper bound coming from a feasible solution, a tentative upper bound is fixed a priori without evidence that this value is not smaller than the optimum of the problem. The problem, once restricted by using this bound, can be either equivalent to the original one, or infeasible. Solving the restricted problem yields a posteriori evidence that the tentative upper bound is correct or not. The overall procedure can be summarized like this:

- Build and filter network G_1, to obtain network G'_1.
- If the number of arcs in G'_1 does not exceed a given threshold (fixed empirically to 300 thousands arcs in our experiments), then build network G_2 from G'_1, filter G_2 and run algorithm BB_2 to solve the problem to optimality.
- Otherwise, it is very likely that the upper bound significantly over-estimates the optimal objective value and that it will result in a very large network G_2 at the basis of the branch-and-bound procedure. In this case, perform a major

iteration: use a tentative upper bound UB^{tent} to build and filter network G_2 from G'_1, and run algorithm BB_2. Two outcomes are possible.

- If algorithm BB_2 does not improve the upper bound, then UB^{tent} underestimates the optimal objective value. In this case, the algorithm performs next major iteration with an increased value of UB^{tent}.
- If algorithm BB_2 completes with a new, improved upper bound, then it is optimal and the overall procedure terminates.

Table 2.1.10 shows the usefulness of the tentative upper bound on instances with setup times: the size of network G_2 once it is filtered during the last major iteration (i.e. with the smallest feasible tentative upper bound) is more than seven times smaller than when it is filtered with the initial upper bound for 100-job instances. We do not report the corresponding results for instances without setup times since the impact is very limited for this class of problems.

Number of nodes in G_2 after filtering (in thousands)							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
163.7	284.8	396.8	766.3	63.1	88.4	135.1	237.1
Number of arcs in G_2 after filtering (in thousands)							
Initial upper bound				Best feasible tentative upper bound			
n=60	n=70	n=80	n=100	n=60	n=70	n=80	n=100
1 866.5	4 236.0	6 931.7	18 544.7	344.1	544.5	1013.3	2 237.8

Table 2.1.10: Average size of network G_2 after filtering, for problem $F2|ST_{SI}|\sum C_i$, when filtering is performed with the initial upper bound and the smallest feasible tentative upper bound.

2.1.4 Computational results

All the algorithms were coded in C++ and compiled under Microsoft Visual Studio 2012. All the experiments were conducted on a laptop computer with an Intel i7 2.7 GHz processor and 16GB RAM. The solver used to solve the MILP and LP models is IBM ILOG Cplex v12.6.

The initial value of the tentative upper bound is chosen as $UB^{tent} = \alpha_1(UB - LB) + LB$, where UB and LB are, respectively, the initial upper bound obtained by the dynasearch procedure and the best lower bound at the end of the subgradient procedure applied to filter network G_1. If necessary, UB^{tent} is increased at each major iteration by $\alpha_2(UB - LB)$. In our implementation, $\alpha_1 = 0.4$ and $\alpha_2 = 0.2$, ensuring the convergence of the overall procedure in four major iterations.

We use the same tuning of the different parameters for subgradient procedure for filtering networks G_1 and G_2 for both problem types with and without setup

times: $\gamma^{ini} = 1$, $\kappa_S = 0.95$, $\kappa_E = 1.02$, $\varepsilon = 10^{-4}$, $\delta_S = 2$, $\delta_T = n$, $min_{iter} = 2n$.

2.1.4.1 Instances without setup times

We first tested our branch-and-bound algorithms on instances without setup times generated similarly to the instances in [Haouari & Kharbeche 2013]. The instance generator takes as input the number of jobs n, and a maximum duration of operations p_{max}. The duration of the operations is drawn from the uniform distribution $[1, p_{max}]$. We generated 20 instances for each combination of parameters $n \in \{10, 30, 40, 50, 60, 70, 80, 90, 100, 140\}$ and $p_{max} \in \{10, 100\}$.

In Table 2.1.11 we report the number of instances solved to optimality within the time limit of 1000 seconds (for the whole method) by both algorithms BB_1 and BB_2. Algorithm BB_2 solves all instances of the testbed within the time limit. The hardest 100-job instance is solved in 602 seconds.

Table 2.1.11: $F2||\sum C_i$: Number of instances of our testbed solved to optimality within 1000 seconds. The hardest 100-job instance is solved by BB_2 in 602 seconds.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	20	20	20	20	20	20	20	18	18
BB_1	$[1-100]$	20	20	20	20	19	20	20	19	15
BB_2	$[1-10]$	20	20	20	20	20	20	20	20	20
BB_2	$[1-100]$	20	20	20	20	20	20	20	20	20

In Table 2.1.12 and Table 2.1.13 we give, respectively, the average running time of our branch-and-bound algorithms, and the average number of nodes in the search tree. These average values are computed on the instances solved to optimality by both methods.

Table 2.1.12: $F2||\sum C_i$: Average total running time (in seconds) of the algorithms. Time for initial heuristic, network building and filtering is included. For BB_2, the overall time including all major iterations of the tentative upper bound procedure is reported.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	0.3	2.7	5.7	11.2	19.8	22.7	60.9	62.4	89.6
BB_1	$[1-100]$	0.2	3.4	8.7	16.9	33.3	60.8	113.6	339.1	455.6
BB_2	$[1-10]$	0.3	2.2	4.5	8.9	14.8	23.2	35.1	54.6	95.0
BB_2	$[1-100]$	0.3	3.0	8.2	17.4	34.2	57.3	91.8	153.6	215.8

As it can be seen from the results, algorithm BB_2 performs better than BB_1 in terms of computing time, and explores up to 70 times less nodes for instances with 100 jobs and large durations. The difference in the number of nodes can be explained by the fact that lower bound L^2 is much stronger than L^1 and reduces the

Table 2.1.13: $F2||\sum C_i$: Average number of nodes for the branch-and-bound algorithms (K: thousands, M: millions). For BB_2, the total number of nodes explored in all major iterations of the tentative upper bound procedure is reported.

Alg.	Duration	n=10	n=30	n=40	n=50	n=60	n=70	n=80	n=90	n=100
BB_1	$[1-10]$	0.0 K	0.7 K	5.0 K	38.5 K	124.7 K	777.1 K	7.0 M	20.3 M	15.3 M
BB_1	$[1-100]$	0.0 K	1.7 K	78.0 K	320.6 K	2.6 M	23.3 M	53.0 M	214.0 M	283.5 M
BB_2	$[1-10]$	0.0 K	0.0 K	0.2 K	2.5 K	2.5 K	45.7 K	236.1 K	1.3 M	8.6 M
BB_2	$[1-100]$	0.0 K	0.0 K	0.0 K	4.0 K	12.2 K	303.4 K	348.2 K	3.1 M	3.9 M

size of the search tree by several orders of magnitude for some instances. Moreover, it allows an early detection of infeasible subsequences that cannot be extended to complete sequences with each job processed exactly once. For example, if job j is already scheduled, and no path without job j exists in G_2 from current node to the sink node, then $B_{\neg j}(v) = \infty$ and the branch-and-bound node is pruned by the bound. In less extreme scenarios, such paths exist but are all relatively costly, and $B_{\neg j}(v)$ may be sufficiently large to prune the node. The difference in the solution times is less. One can remark that the distribution of the time consumed by each algorithm is not the same: for BB_1, most of the time is spent in the exploration of the search tree, while for BB_2 the computational effort is balanced between the filtering of G_2 and the exploration of the search tree. The relatively small difference in solution times is also due to the fact that calculating L^2 takes linear time instead of constant time for bound L^1.

In order to test the limits of algorithm BB_2, we generated 40 140-job instances. Our method still performs well for this size of instances: the hardest 140-job instance is solved to optimality in 3006 seconds, and the average computing time is 752 seconds. Within a time limit of 1000 seconds, 18 out 20 (resp. 12 out of 20) instances with small (resp. large) processing times are solved. Furthermore, we tested the proposed algorithms on the instances in [Haouari & Kharbeche 2013] (with up to 70 jobs), and both BB1 and BB2 solved them to optimality within 1000 seconds.

Efficiency of the Memory Dominance Rule The Memory Dominance Rule appears to be critical for the efficiency of algorithm BB_1: when it is disabled, this method does not solve any 100-job instance within 1000 seconds. Moreover, the average (resp. maximum) number of nodes for solving 60-job instances reaches 16 millions (resp. 225 millions) for small processing times, and 1.3 billions (resp. 5.8 billions) for large processing times, while the average computing time is multiplied by a factor 3.

The rule appears to be less critical for algorithm BB_2, probably because of the better quality of the lower bound used. However, it still clearly makes the algorithm more robust in terms of computing time and number of nodes explored. Two 100-job instances are not solved within 1000 seconds when the rule is disabled, but one of them needs more than 7700 seconds to be closed. The average (resp. maximum)

number of nodes for solving 100-job instances reaches 78 millions (resp. 958 millions) for small processing times, and 179 millions (resp. 2.7 billions) for large processing times, while the average computing time is multiplied by a factor 4.

2.1.4.2 Instances with setup times

For the problem $F2|ST_{SI}|\sum C_i$, our solving methods are tested against the testbed of Gharbi et al. [Gharbi *et al.* 2013]. Their generator takes as input a number n of jobs, and a factor K for setup times. The processing time of each operation is drawn from the uniform distribution $[1, 100]$, and one setup time is drawn for each operation from the uniform distribution $[1, 100K]$. As mentioned in Section 2.1.1, we modify the instances by integrating the setup time of the first operation of each job into its processing time. The whole set of instances is composed of 50 instances for each combination of the number of jobs ranging from 10 to 500, and $K \in \{0.25, 0.5, 0.75, 1\}$. We restrict our computational study to the 800 instances with the number of jobs in $n \in \{60, 70, 80, 100\}$ (the test set of [Gharbi *et al.* 2013] does not contain 90-job instances).

Table 2.1.14 reports the number of instances solved to optimality by both methods within 1000 seconds. Algorithm BB_2 significantly outperforms BB_1. Moreover, algorithm BB_2 solves all instances of the testbed with up to 100 jobs in less than two hours: only four 100-job instances are not solved within one hour; the hardest instance is solved in 6443 seconds.

The value of parameter K has no significant impact on the performance of algorithm BB_1. However, while BB_2 solves 47 out of the 50 100-job instances within 1000 seconds when $K = 0.25$, it solves only 26 of the instances when $K = 1$ in the same time. This can be explained by the wider range of completion-to-completion lag in the latter case, leading to more nodes in both networks G_1 and G_2. For large instances, the larger size of G_2 implies a consequent computational effort during the first iterations of the subgradient procedure (when the network is only a little shrunk). This hypothesis is confirmed by the fact that the root gap is not significantly impacted by parameter K, while the average time for the subgradient procedure is increased by a factor two for 80-job and 100-job instances when increasing parameter K from $K = 0.25$ to $K = 1$.

Table 2.1.14: $F2|ST_{SI}|\sum C_i$: Number of instances of the testbed of [Gharbi *et al.* 2013] solved to optimality within 1000 seconds. The hardest instance is solved by algorithm BB_2 in 6443 seconds.

Alg.	n=60	n=70	n=80	n=100
BB_1	200	190	125	10
BB_2	200	200	200	145

In Table 2.1.15 and Table 2.1.16 we give, respectively, the average running time of our branch-and-bound algorithms, and the average number of nodes in the search

tree. These results emphasize the importance of having tight lower and upper bounds in our methods: algorithm BB_2 performs two order of magnitude less nodes than algorithm BB_1.

Table 2.1.15: $F2|ST_{SI}|\sum C_i$: Average total running time (in seconds) of the algorithms. Time for initial heuristic, network building and filtering is included. For BB_2, the overall time including all major iterations of the tentative upper bound procedure is reported.

Alg.	n=60	n=70	n=80	n=100
BB_1 - Avg. on solved instances	79.6	280.3	393.9	615.8
BB_2 - Avg. on instances solved by BB_1	99.5	162.6	235.9	478.8
BB_2 - Avg. on all instances	99.5	168.6	287.18	935.2

Table 2.1.16: $F2|ST_{SI}|\sum C_i$: Average number of nodes for the branch-and-bound algorithms (K: thousands, M: millions). For BB_2, the total number of nodes explored in all major iterations of the tentative upper bound procedure is reported.

Alg.	n=60	n=70	n=80	n=100
BB_1 - Avg. on solved instances	30.4 M	142.6 M	216.2 M	301.1 M
BB_2 - Avg. on instances solved by BB_1	125.7 K	310.6 K	626.1 K	822.0 K
BB_2 - Avg. on all instances	125.7 K	369.2 K	2.4 M	42.6 M

2.2 Successive sublimation dynamic programming: application to the temporal knapsack problem

This section is based on the journal paper [Clautiaux *et al.* 2021]. In this section, we address the TKP, a generalization of the classical knapsack problem, where selected items enter and leave the knapsack at fixed dates. We model the TKP with a dynamic program of exponential size, which is solved using SSDP. This method starts by relaxing a set of constraints from the initial problem, and iteratively reintroduces them when needed. We show that a direct application of SSDP to the temporal knapsack problem does not lead to an effective method, and that several improvements are needed to compete with the best results from the literature. The rest of the section is organized as follows. In Section 2.2.1, we formally discuss integer programming and DP formulations for TKP. In Section 2.2.2, we describe an application of SSDP to TKP. Section 2.2.3 outlines the various refinements of the method that are necessary to obtain competitive results. We present our computational experiments in Section 2.2.4.

Contribution from a State-Space Relaxation perspective. The SSDP method has been highly successful for single machine scheduling problems [Tanaka *et al.* 2009, Tanaka & Fujikuma 2012, Tanaka & Araki 2013]. However, to the best of our knowledge, it had not been applied in other contexts, probably because of its demanding implementation and the lack of generic tools. We show that the application of the textbook approach does not make a competitive approach, at least for TKP. Inspired by the impressive work of Dr. Shunji Tanaka (Kyoto University) in scheduling, we integrate many problem-specific properties to help reducing the size of the relaxed DP models. We also exploit an original MILP formulation to obtain good initial Lagrangian multipliers. From a methodological point-of-view, we use partial enumeration techniques to improve the efficiency of variable fixing and take a more general view, investigating different strategies for the choice of the constraints to be reintroduced into the relaxed DP. From a theoretical perspective, we give a formal proof of a conjecture never explicitly stated in the related literature: once an arc is eliminated from the graph at a given iteration, all corresponding arcs can be removed from the graphs built during subsequent iterations.

We choose to keep most of the contents of the original paper [Clautiaux *et al.* 2021] in this document, in order to give a detailed illustration of this rather obscure method.

Problem description. The Temporal Knapsack Problem is a generalization of the well-known knapsack problem, where each item has a time window during which it can be added to the knapsack, and the capacity constraint is considered at each time period. The name Temporal Knapsack was introduced in [Bartlett *et al.* 2005], although the problem had already been studied in [Chen *et al.* 2002] as a bandwidth allocation problem. Formally, the TKP can be stated as follows.

Problem 2.2.1 (Temporal Knapsack Problem)**.** *Let $\mathcal{I} = \{1, \dots, n\}$ be a set of items. Each item $i \in \mathcal{I}$ has a profit $p_i \in \mathbb{R}_+$, a size $w_i \in \mathbb{N}$, and a time interval $[s_i, f_i)$, where $s_i, f_i \in \mathbb{N}$ and $s_i < f_i$. Moreover, let $W \in \mathbb{N}$ be the weight of the knapsack. A feasible solution comprises a subset $\mathcal{J}$ of $\mathcal{I}$ such that for any value of $m \in \mathbb{N}$, the sum of the sizes of the items in $\mathcal{J}$ whose time interval contains m is less than or equal to W. The* Temporal Knapsack Problem *is the problem of finding a feasible subset $\mathcal{J}$ of $\mathcal{I}$ with maximum profit.*

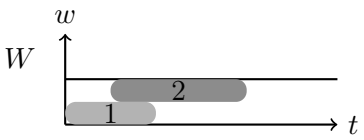
Figure 2.2.1 represents an instance of TKP with three items, and its two inclusion-wise maximal solutions. One can see that no solution can contain both items 2 and 3, since they are both active at time 3, and the sum of their sizes is larger than the capacity of the container. Conversely, 1 and 3 can be selected in the same solution, despite their size, since their time intervals do not overlap.

In its general form, the TKP is NP-hard in the strong sense [Bonsma *et al.* 2014]. The first results proposed for TKP were focused on a theoretical characterization: a polynomially solvable case [Arkin & Silverberg 1987], and approximation results [Chen *et al.* 2002, Calinescu *et al.* 2002]. Two dynamic programs were proposed by [Chen *et al.* 2002] and [Caprara *et al.* 2013], and are described more precisely in the

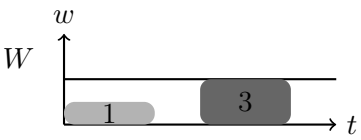
$W = 8$

i	w_i	p_i	$[s_i, f_i)$
1	4	2	$[0, 2)$
2	4	20	$[1, 4)$
3	8	200	$[3, 5)$

(a) A TKP Instance



(b) A first solution of profit 22



(c) A second solution of profit 202

Figure 2.2.1: A TKP instance with three items and its two inclusion-wise maximal solutions

next section. A Dantzig-Wolfe reformulation was proposed by [Caprara *et al.* 2013], where the idea is to partition the time horizon into consecutive time periods (blocks). For each block, the variables related to the items whose time intervals intersect the corresponding time period are duplicated. Each subproblem is a smaller TKP, while the master problem makes sure that the duplicated variables related to the same item have the same value. Based on this reformulation, [Caprara *et al.* 2013] proposed a branch-and-price algorithm. These results were improved in [Gschwind & Irnich 2017] using an innovative stabilization technique. This method relies on so-called dual-optimal inequalities, and uses dominance relations between (pairs of) items to add additional effective dual cuts that are satisfied by at least one optimal dual solution. Finally, [Caprara *et al.* 2016] proposed a method based on the previous Dantzig-Wolfe reformulation, where each subproblem is itself decomposed into a master problem and several smaller TKPs, and solved by branch-and-price.

Here, we propose a new exact algorithm for TKP. It is based on an exponential size dynamic program, where the size of the state-space depends exponentially on the number of items n. SSDP consists in solving a relaxation of the original dynamic program, removing some transitions that cannot belong to an optimal solution, and incrementally reintroducing the relaxed constraints until an optimality proof is reached. An originality of this method is that it does not use a label-setting algorithm, but explicitly builds the graph representation of each relaxed dynamic program. The effectiveness of the method is highly dependent on the capability to reuse information from the previous iterations (primal and dual bounds, variable fixing).

As is the case with many generic methods, obtaining an effective version of SSDP for a new problem is not straightforward. We numerically show that a basic application of this technique to TKP is not competitive compared to state-of-the-art solvers. We then propose several advanced algorithmic techniques which allow a significant improvement on the computational results. We implemented our algorithms and empirically compared them against a commercial MIP solver, using instances proposed in [Caprara *et al.* 2013]. We also report results obtained by [Gschwind & Irnich 2017] on these instances. These experiments show that our algorithm is competitive compared to the state of the art.

2.2.1 Integer programming and dynamic programming models

In this section, we discuss compact MIP formulations and dynamic programs for TKP.

Integer programming formulations We first recall the commonly used integer programming formulation (see *e.g.* [Caprara *et al.* 2013, Caprara *et al.* 2016]) for TKP. In this model, each binary variable x_i is equal to one if item i is selected, and zero otherwise, similarly to the classical knapsack problem. As suggested in [Caprara *et al.* 2013], it is sufficient to check the capacity constraints at starting time s_j of each item j.

$$\max \sum_{i \in \mathcal{I}} p_i x_i \tag{2.2.1}$$

$$\text{s.t.} \sum_{i \in \mathcal{I}: s_i \leq s_j < f_i} w_i x_i \leq W, \quad j \in \mathcal{I} \tag{2.2.2}$$

$$x_i \in \{0,1\}, \quad i \in \mathcal{I} \tag{2.2.3}$$

We now propose an alternative MIP formulation for TKP, which is not meant to be used directly to solve the problem but simplifies the presentation of our dynamic program. In this model, we see the problem as a succession of *events* where decisions have to be taken (adding the item, or removing the item). This means that we split each original variable x_i into two distinct variables that have to take the same value.

Let $(1, \dots, 2n+1)$ be an ordered list of *events*. There are two events in the list for each item, plus an additional dummy event $2n+1$. We distinguish the events related to the beginning of a time window (set $\mathcal{E}^{\text{in}}$) and those related to the end of a time window (set $\mathcal{E}^{\text{out}}$). For each event e in $1, \dots, 2n$, we denote by $i(e) \in \mathcal{I}$ the item related to e, and by $t(e)$ the time period when e occurs, *i.e.*, $t(e) = s_{i(e)}$ if $e \in \mathcal{E}^{\text{in}}$ and $t(e) = f_{i(e)}$ if $e \in \mathcal{E}^{\text{out}}$. Events related to items are ordered from 1 to $2n$ as follows: $e < e'$ if $t(e) < t(e')$ or $(t(e) = t(e') \wedge e \in \mathcal{E}^{\text{out}} \wedge e' \in \mathcal{E}^{\text{in}})$ (ties are broken arbitrarily).

The decisions of the new MIP model are related to these events. For each event e, we define a binary variable y_e that indicates whether the action related to event e is performed or not. If $e \in \mathcal{E}^{\text{in}}$, this decision corresponds to adding $i(e)$ to the current solution. If $e \in \mathcal{E}^{\text{out}}$, the decision corresponds to removing $i(e)$. In a valid solution, an item leaves the knapsack if and only if it enters the knapsack in a previous event. Each variable ϕ_e $(e = 1, \dots, 2n)$ is equal to the total size of the

selected items at the end of event e.

$$\max \sum_{e=1,\dots,2n} \frac{1}{2} p_{i(e)} y_e \tag{2.2.4}$$

$$\phi_1 = w_{i(1)} y_1 \tag{2.2.5}$$

$$\phi_e = \phi_{e-1} + w_{i(e)} y_e \quad e \in \mathcal{E}^{\text{in}} \setminus \{1\} \tag{2.2.6}$$

$$\phi_e = \phi_{e-1} - w_{i(e)} y_e \quad e \in \mathcal{E}^{\text{out}} \tag{2.2.7}$$

$$\phi_e \leq W \quad e = 1, \dots, 2n \tag{2.2.8}$$

$$\phi_{2n} = 0 \tag{2.2.9}$$

$$y_e - y_{e'} = 0 \quad e \in \mathcal{E}^{\text{in}}, e' \in \mathcal{E}^{\text{out}}, i(e) = i(e') \tag{2.2.10}$$

$$y_e \in \{0,1\}, \phi_e \in \mathbb{R}_+ \quad e = 1, \dots, 2n \tag{2.2.11}$$

The objective function is similar to that of model (2.2.1). The only difference is that the profit is split between the two events related to each item. Constraints (2.2.5)–(2.2.7) ensure that the capacity consumption at the end of each event is consistent with the contents of the knapsack. Constraints (2.2.8) and (2.2.9) guarantee that the capacity constraints are satisfied. Note that constraint (2.2.9) is redundant when no other constraint is relaxed. Constraints (2.2.10) state that if an item enters the knapsack, it has to leave it. We call constraints (2.2.10) *consistency constraints*. Using two variables to decide if an item is selected in the solution is redundant. The idea behind this variable splitting is to apply Lagrangian relaxation to the consistency constraints. Our method is based on this relaxation, which is similar to the so-called Lagrangian decomposition technique [Guignard & Kim 1987].

Dynamic programs for TKP. To our knowledge, two DP have been proposed for TKP in the literature. Both DP record in each state the subset of items that belong to a partial solution. In [Chen *et al.* 2002], a state $(i, \mathbf{d})$ is characterized by the current item $i \in \mathcal{I}$ and $\mathbf{d} \in \{0,1\}^n$, the characteristic vector of the set of items currently in the knapsack. Let $\boldsymbol{\varepsilon}_k \in \{0,1\}^n$ be the characteristic vector of set $\{k\}$, for $k \in \mathcal{I}$. The value of a state is computed with the following recursive formulas: $f(n+1, \mathbf{0}) = 0$, and $f(i, \mathbf{d}) = \max\{f(i+1, \mathbf{d}'), \quad p_i + f(i+1, \mathbf{d}' + \boldsymbol{\varepsilon}_i)\}$ where $\mathbf{d}'$ is obtained from $\mathbf{d}$ by removing all items whose departure date is before the starting time of i. The left-hand part of the alternative chooses not to select item i (and thus just removes the items whose intervals do not overlap with item $i+1$), while the right-hand part corresponds to selecting item i (and collects the profit of i). In the latter case, i is recorded in the current vector, and serves to fathom configurations where the total size of the items is larger than the size of the bin. The optimal value is equal to $f(1, \mathbf{0})$. States where $\mathbf{d}$ represents an infeasible subset of items are discarded.

In [Caprara *et al.* 2013], another dynamic program is proposed. The possible subsets of items that can be in the knapsack at the same instant are computed *a priori*, and each of them is related to a state. More precisely, the approach is based on a reformulation of the problem as a maximum profit path problem in

an exponentially large graph. The vertex set of this graph can be partitioned in so-called *layers*, one for each knapsack constraint (2.2.2). In each layer, a node is created for each feasible subset of items that can be in the knapsack. There is an arc between two nodes from two consecutive layers if and only if their contents are consistent. The cost of the arc is equal to the sum of the profits of the items that are added to obtain the new configuration. This method can be used to solve only the smallest instances in the literature, since it cannot be applied when many items can be packed at the same time period, as the number of states in the dynamic program grows exponentially with this quantity.

There are some similarities between the two DP described above. Both state-spaces record the elements in the knapsack at a given step. The first DP builds the possible configurations item by item, and thus may consider non-maximal configurations. On the other hand, it may serve to fathom some configurations using some dominance rules or cost considerations, while in the second DP, one has to build all possible configurations beforehand, which may not be possible when their number is large.

Our dynamic program is based on the concept of *events* used in (2.2.4)–(2.2.11). It is an adaptation of [Chen *et al.* 2002] whereby we add redundant information to the states (the capacity consumption), and split the decision of adding an item into two decisions. Both modifications are necessary to compute our relaxations. The model works similarly to model (2.2.4)–(2.2.11) in the sense that it uses the same decisions, and the current capacity is updated event by event recursively; one has to ensure that the capacity constraint remains satisfied and that decisions related to items are consistent.

We now formally describe our dynamic program using *states* and *transitions*. We define a *state* as a tuple $(e, w, \mathbf{d})$ where $e \in \{1, \dots, 2n+1\}$ is the current event, $w \in \mathbb{Z}_+$ the current consumption of the knapsack capacity, and $\mathbf{d} \in \{0,1\}^n$ the characteristic vector of the set of items currently in the knapsack. Note that w is redundant, since it can be deduced from vector $\mathbf{d}$. We call a *transition* the possibility of passing from one state to another by taking a decision. A transition is defined by a tuple $(\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p)$ where $\Delta_e \in \mathbb{Z}_+$ describes the increase in the current event index, $\Delta_w \in \mathbb{Z}$ is the capacity consumed/released when the decision is taken, $\Delta_{\mathbf{d}} \in \{-1, 0, 1\}^n$ a vector that updates the content of the knapsack, and $p \in \mathbb{R}_+$ the profit obtained when the decision is taken.

The possible decisions that can be taken are defined by ψ, the function that associates each state with a set of feasible transitions. For any feasible state $(e, w, \mathbf{d})$, function $\psi((e, w, \mathbf{d}))$ is computed as follows.

$$\psi((e,w,\mathbf{d})) = \begin{cases} \left\{(1, 0, \mathbf{0}, 0), (1, w_{i(e)}, \boldsymbol{\varepsilon}_{i(e)}, \frac{1}{2}p_{i(e)})\right\} & \text{if } e \in \mathcal{E}^{\text{in}} \wedge w + w_{i(e)} \leq W \\ \{(1, 0, \mathbf{0}, 0)\} & \text{if } e \in \mathcal{E}^{\text{in}} \wedge w + w_{i(e)} > W \\ \left\{(1, -w_{i(e)}, -\boldsymbol{\varepsilon}_{i(e)}, \frac{1}{2}p_{i(e)})\right\} & \text{if } e \in \mathcal{E}^{\text{out}} \wedge \mathbf{d}_{i(e)} = 1 \\ \{(1, 0, \mathbf{0}, 0)\} & \text{if } e \in \mathcal{E}^{\text{out}} \wedge \mathbf{d}_{i(e)} = 0 \end{cases}$$

When an event $e \in \mathcal{E}^{\text{in}}$ is considered, two transitions are possible: one corresponding to selecting $i(e)$, the other to not selecting $i(e)$ (the former exists only if the remaining capacity is large enough). When an event $e \in \mathcal{E}^{\text{out}}$ is considered, only one transition is possible, depending on the value $\mathbf{d}_{i(e)}$. The cost function α from each state $(e, w, \mathbf{d})$ is then expressed in a backward recursive fashion.

$$\alpha((e,w,\mathbf{d})) = \begin{cases} \max\limits_{(\Delta_e,\Delta_w,\Delta_{\mathbf{d}},p)\in\psi((e,w,\mathbf{d}))} \{p + \alpha((e+\Delta_e, w+\Delta_w, \mathbf{d}+\Delta_{\mathbf{d}}))\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } e \in \{1,\dots,2n\} \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } e = 2n+1, w = 0, \mathbf{d} = \mathbf{0} \end{cases} \tag{2.2.12}$$

The optimal value of the TKP is $\alpha((1,0,\mathbf{0}))$.

2.2.2 Specializing Successive Sublimation Dynamic Programming to TKP

In this section, we explain how SSDP can be used to solve TKP. We first describe the generic algorithm, emphasizing the main points to be studied, namely choosing a relaxation, solving the relaxation, and updating the relaxation to obtain a refined model. We then address each point specifically.

2.2.2.1 Graph representation of the dynamic program

We first describe a graph representation of dynamic program (2.2.12), where states are represented by vertices, and transitions by arcs. The graph representation $G = (V, A)$ is obtained by creating a vertex for each possible reachable state of (2.2.12), and an arc for each possible transition. Each arc has the profit p of the corresponding transition. Starting from initial state $(1, 0, \mathbf{0})$, the nodes of the graph are created by computing recursively function $\psi(s)$ and creating the corresponding transitions to obtain the arcs and the vertices.

Figure 2.2.2 illustrates the graph representation of DP (2.2.12) applied to the instance in Figure 2.2.1a. The different paths from $(1, 0, (0, 0, 0)$ to $(7, 0, (0, 0, 0))$ are related to the different solutions for instance 2.2.1a. We call a *layer* the set of vertices related to a given event. Note that in layers related to *out* events, vertices have exactly one outgoing arc, while in layers related to *in* events, there are at most two possible outgoing arcs.

Once the graph representation of the DP is built, the problem is solved by finding the maximum profit path between the vertex associated with $(1, 0, (0, 0, 0))$ and the vertex associated with the final state $(2n + 1, 0, (0, 0, 0))$. Since the graph has no directed cycles, we use Bellman's algorithm.

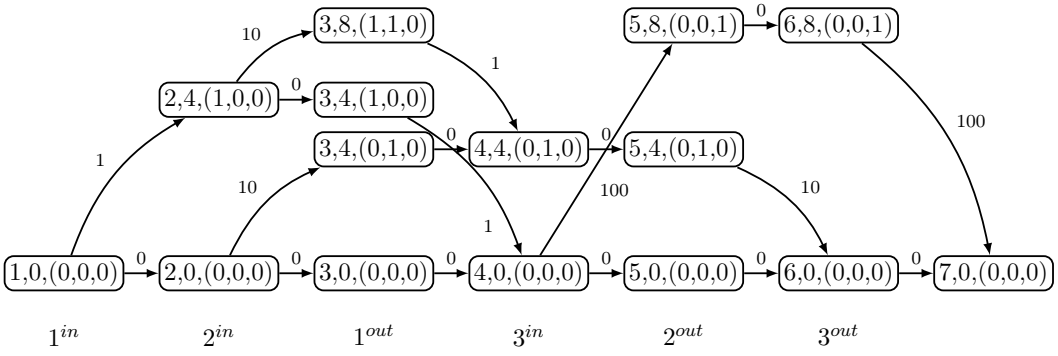


Figure 2.2.2: Graph representation of dynamic program (2.2.12) applied to the instance in Figure 2.2.1a.

2.2.2.2 Iterative state-space relaxation

Building the graph representation of DP (2.2.12) and using Bellman's algorithm does not lead to a practical method. The size of the state-space in (2.2.12) is exponential in the size of $\mathbf{d}$: the size of the binary vector $\mathbf{d}$ is n, so the size of the state-space is in $O(n \times 2^n)$.

For TKP, we decided to use SSDP for several reasons. First, our preliminary experiments had shown that the gap between the dual and primal bounds was good enough to filter a good percentage of arcs on many instances, and thus that the extended graph would not grow too fast. Second, the dominance relations between two labels are weak for TKP, since one has to take into account the residual capacity that is freed by the items leaving the knapsack over time.

2.2.2.3 Presentation of the generic algorithm

SSDP is a dual method that iteratively solves problems obtained by applying state-space relaxation to a dynamic program. An initial relaxation is obtained by relaxing constraints that cause the exponential size of the state-space. A first dual bound is obtained by solving the relaxation. This dual bound is improved by refining the relaxation (*i.e.* reintroducing constraints) until the duality gap to a known primal bound is closed. The bound obtained at each step is possibly reinforced using a Lagrangian relaxation of the constraints. At each step of the algorithm, some unnecessary states and transitions are identified and removed from subsequent relaxations.

An important feature of the algorithm is that at each step it explicitly constructs the graph representation of the current dynamic program. This representation is used to record variable fixing information from one relaxation to the next. The main steps of the method are summarized in Algorithm 2.2.1.

Similar to many generic frameworks, several ad-hoc key ingredients have to be designed for each new problem. The most important are the set of relaxed con-

Algorithm 2.2.1: SSDP

Compute the graph related to the first relaxation.
Build graph G^0, the graph representation of the initial relaxation ;
$\ell \leftarrow 0$;
Solving the relaxation and filtering.
Solve the relaxation corresponding with graph G^ℓ to obtain a solution `sol` ;
if `sol` *is feasible and has a cost equal to the current best dual bound* **then return** `sol`;
Remove non-optimal states and transitions, obtaining graph $\hat{G}^\ell$;
Sublimation.
Construct the new graph $G^{\ell+1}$ from $\hat{G}^\ell$ by reintroducing new constraints ;
$\ell \leftarrow \ell + 1$;
go back to *step 4* ;

straints, and the type of relaxation used. Another major ingredient is the algorithm used to solve each relaxed problem, and its capability to eliminate infeasible/non-optimal partial solutions. Finally, an effective method to update the relaxation at each step is necessary.

2.2.2.4 Relaxation used for TKP

Our relaxation consists in not considering consistency constraints for some items. This is equivalent to considering only a subset of the values in $\mathbf{d}$ in the states, which reduces the size of the state-space. In this case, an item can enter the knapsack and not leave it, or vice-versa. Our algorithm relies on the fact that there is a one-to-one correspondence between the consistency constraints and the dimensions of the DP related to vector $\mathbf{d}$ in (2.2.12).

Observation 2.2.1. *Projecting out vector* $\mathbf{d}$ *in* (2.2.12) *is equivalent to relaxing consistency constraints* (2.2.10) *in* (2.2.4)– (2.2.11).

We use a modified graph representation to compute the relaxation. At a given iteration of the algorithm, the relaxation is based on a set $\mathcal{J}$ of items that have to satisfy constraint (2.2.10). Let $G_\mathcal{J} = (V_\mathcal{J}, A_\mathcal{J})$ be the graph representation of the relaxed DP associated with $\mathcal{J}$. A vertex is now identified by a tuple $(e, w, \mathcal{C})$, where $\mathcal{C} \subseteq \mathcal{J}$ is the subset of items from $\mathcal{J}$ that are in the knapsack. Each vertex v represents a set of states $\mathcal{S}_\mathcal{J}(v) = \mathcal{S}_\mathcal{J}((e, w, \mathcal{C})) = \{(e', w', \mathbf{d}) : e' = e, w' = w, \forall i \in \mathcal{J}, d_i = 1 \leftrightarrow i \in \mathcal{C}\}$. For a given state $s = (e, w, \mathbf{d})$, we denote by $\hat{v}_\mathcal{J}(s)$ the unique vertex v such that $s \in \mathcal{S}_\mathcal{J}(v)$.

Given the modified graph representation, the relaxed dynamic program is obtained by the following recursive functional equation that applies to the vertices of

the graph: $\hat{\alpha}_{\mathcal{J}}((2n+1, 0, \emptyset)) = 0$ and

$$\hat{\alpha}_{\mathcal{J}}((e, w, \mathcal{C})) = \max_{(\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p) \in \cup_{s \in \mathcal{S}_{\mathcal{J}}((e,w,\mathcal{C}))} \psi(s)} \{p + \hat{\alpha}_{\mathcal{J}}(\hat{v}_{\mathcal{J}}(e + \Delta_e, w + \Delta_w, \mathbf{d} + \Delta_{\mathbf{d}}))\} \tag{2.2.13}$$

The optimal solution for the relaxation is obtained by computing $\hat{\alpha}_{\mathcal{J}}((1, 0, \emptyset))$.

For any arc a in $A_{\mathcal{J}}$, we denote by $\mu(a)$ the transition associated with arc a. For each arc $a \in A_{\mathcal{J}}$, let $\tau(a)$ be its tail and $h(a)$ be its head. For a vertex v, let $\Gamma^+(v)$ (resp. $\Gamma^-(v)$) be the set of outgoing arcs (resp. incoming arcs).

Figure 2.2.3 depicts the graph representation of the relaxed version of the dynamic program when $\mathcal{J} = \emptyset$. Note that in this relaxation, the vertices in the *out* layers can have up to two outgoing arcs, instead of one originally. For example, since vertex $(3, 4, \emptyset)$ represents states $(3, 4, (1, 0, 0))$ and $(3, 4, (0, 1, 0))$ of the original dynamic program, it has two outgoing arcs, related to these two original DP states. All feasible paths in the original graph representation remain feasible, but new paths that are not related to feasible solutions are now considered. The optimal solution for this relaxation is to add item 2, remove item 1, add item 3 and remove item 3, for a profit equal to 211.

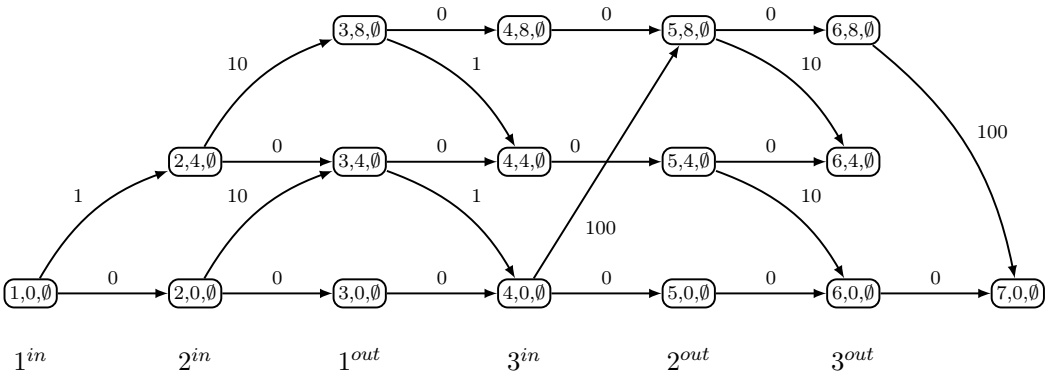


Figure 2.2.3: Graph representation of the relaxed DP with $\mathcal{J} = \emptyset$. In this relaxation, the presence of each item in the knapsack is not recorded. Note that state $(6, 4, \emptyset)$ does not belong to any path from the source to the sink of the graph, and will be deleted.

2.2.2.5 Solving the relaxation and filtering

We use Lagrangian relaxation to produce stronger bounds than those of a combinatorial relaxation, while keeping the problem tractable. This method is used to compute a dual bound, and to remove some arcs that cannot belong to an optimal solution.

Let $\boldsymbol{\pi} \in \mathbb{R}^n$ be the vector of Lagrangian multipliers associated with Constraints (2.2.10) for indices $\mathcal{I} \setminus \mathcal{J}$. To simplify the notation, we assume that $\boldsymbol{\pi}$ and $\mathbf{d}$ are

always of size n. Within this setting, for a given set $\mathcal{J}$, and a given vector of multipliers $\boldsymbol{\pi}$, the Lagrangian dual function can be written as:

$$L_{\mathcal{J}}(\boldsymbol{\pi}) = \max \sum_{e \in \mathcal{E}^{\text{in}}} \left(\frac{1}{2} p_{i(e)} + \boldsymbol{\pi}_{i(e)} \right) y_e + \sum_{e \in \mathcal{E}^{\text{out}}} \left(\frac{1}{2} p_{i(e)} - \boldsymbol{\pi}_{i(e)} \right) y_e \tag{2.2.14}$$

$$(2.2.6) - (2.2.9), (2.2.11) \tag{2.2.15}$$

$$y_e - y_{e'} = 0 \quad e \in \mathcal{E}^{\text{in}}, e' \in \mathcal{E}^{\text{out}}, i(e) = i(e'), i(e) \in \mathcal{J} \tag{2.2.16}$$

In Figure 2.2.4, we report the graph obtained by adding Lagrangian costs to the initial state-space relaxation. There is one multiplier for each consistency constraint. Here, we choose $(0, -3, 0)$ to penalize the possibility of making item 2 enter the knapsack without leaving it. The new optimal solution is the same as in Figure 2.2.3, but its cost is now 208, leading to a better bound. Note that by choosing vector $(0, -10, 0)$ for the Lagrangian multipliers, the relaxation would have allowed the problem to be solved optimally.

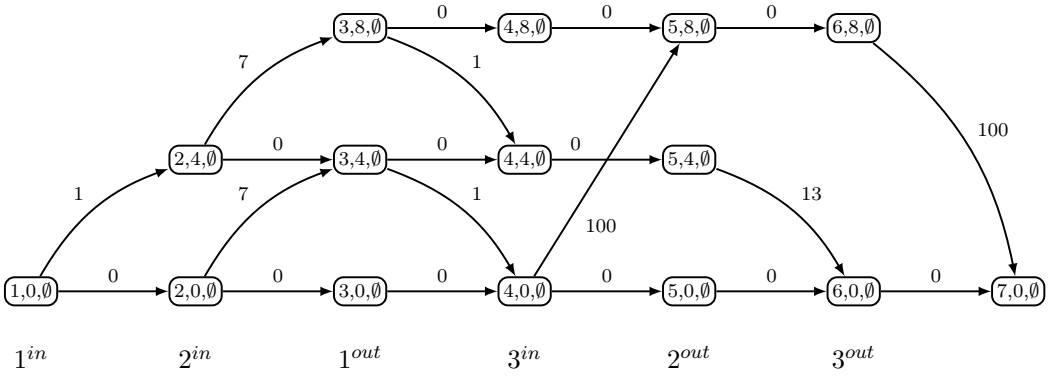


Figure 2.2.4: Graph representation of the relaxed DP for $\mathcal{J} = \emptyset$, with Lagrangian costs $(0, -3, 0)$. Adding item 2 now has a profit of 7 instead of 10, while removing item 2 now has an increased profit of 13 instead of 10.

For fixed $\mathcal{J}$ and any vector $\boldsymbol{\pi}$, $L_{\mathcal{J}}(\boldsymbol{\pi})$ is an upper bound on the optimal value of (2.2.4)-(2.2.11). To compute a good bound using this relaxation, we need to solve approximately the Lagrangian dual problem $\min_{\boldsymbol{\pi} \in \mathbb{R}^n} \{L_{\mathcal{J}}(\boldsymbol{\pi})\}$. In the case of a primal maximization problem, function $L_{\mathcal{J}}(\boldsymbol{\pi})$ is known to be convex, which implies that minimizing this function can be done using a subgradient algorithm, or one of its refinements (see for example [Bertsekas 2015]).

We solve the Lagrangian dual problem using the Volume algorithm proposed in [Barahona & Anbil 2000]. This approximate method builds a sequence of solutions $\boldsymbol{\pi}$ that converges to an optimum. For each value of $\boldsymbol{\pi}$, $L_{\mathcal{J}}(\boldsymbol{\pi})$ is computed by applying Bellman's algorithm on graph $(V_{\mathcal{J}}, A_{\mathcal{J}})$, where the profits of the arcs are modified to take into account the penalization of the relaxed constraints. More

precisely, for each arc a such that $\mu(a) = (\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p)$, the profit of a is now $p+ < \boldsymbol{\pi}, \Delta_{\mathbf{d}} >$. In what follows, we call $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ the graph $G_{\mathcal{J}}$ with the costs modified by the Lagrangian multipliers $\boldsymbol{\pi}$. We denote by $v^0 = (1, 0, \emptyset)$ the vertex representing the initial state, and by $v^{\Omega} = (2n+1, 0, \emptyset)$ the vertex representing the terminal state. We denote by $\alpha^{\boldsymbol{\pi}}_{\mathcal{J}}((e, w, \mathcal{C}))$ the value of Bellman's function for vertex $(e, w, \mathcal{C})$. The value of $L_{\mathcal{J}}(\boldsymbol{\pi})$ is the maximum profit of a path from v^0 to v^{Ω}. Solving the Lagrangian subproblem has complexity $O(|V_{\mathcal{J}}| + |A_{\mathcal{J}}|)$ using Bellman's algorithm.

Figure 2.2.5 illustrates a case where a dimension has been added (namely the dimension related to item 2), and Lagrangian costs are used for the other dimensions. Note that all paths where 2 is added and not removed or vice-versa have been excluded.

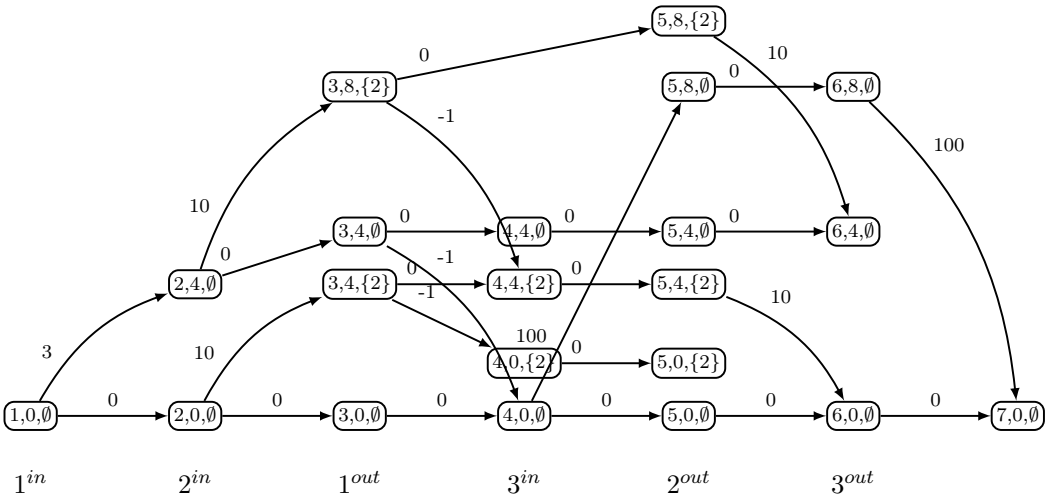


Figure 2.2.5: Graph representation of the relaxed DP for $\mathcal{J} = 2$, with Lagrangian costs $(2, 0, 0)$. Nodes $(4, 0, \{2\})$, $(5, 0, \{2\})$, $(4, 4, \emptyset)$, $(5, 4, \emptyset)$, $(5, 8\{2\})$ and $(6, 4, \emptyset)$ can be deleted since they are not contained in any path from the source to the sink.

Observation 2.2.2. *Problem* (2.2.4)-(2.2.11) *is equivalent to the problem defined by graph* $G^{\boldsymbol{\pi}}_{\mathcal{I}}$*, for all* $\boldsymbol{\pi} \in \mathbb{R}^n$*. Indeed, any path in* $G^{\boldsymbol{\pi}}_{\mathcal{I}}$ *defines a feasible solution of* (2.2.4)-(2.2.11) *with the same cost since the contributions of Lagrangian multipliers are canceled out.*

Now we recall a result used in [Ibaraki 1987,Ibaraki & Nakamura 1994] to remove unnecessary vertices and arcs from $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ (and thus the corresponding states and transitions). For this purpose, let us remark that for any node $(e, w, \mathcal{C}) \in V_{\mathcal{J}}$, Bellman function value $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}((e, w, \mathcal{C}))$ is equal to the maximum cost of a path in $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ from $(e, w, \mathcal{C})$ to v^{Ω}. Likewise, we define $\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}}((e, w, \mathcal{C}))$ as the maximum cost of a path from v^0 to $(e, w, \mathcal{C})$.

Proposition 2.2.1 ([Ibaraki 1987]). *For $\mathcal{J} \subseteq \mathcal{I}$, let $a \in A_{\mathcal{J}}$, such that $\mu(a) = (\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p)$, between two vertices $(e^1, w^1, \mathcal{C}^1)$ and $(e^2, w^2, \mathcal{C}^2)$, and $\boldsymbol{\pi} \in \mathbb{R}^n$. The following value is an upper bound on the cost of any path in $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ that uses arc a:*

$$\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}}((e^1, w^1, \mathcal{C}^1)) + p + < \boldsymbol{\pi}, \Delta_{\mathbf{d}} > + \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}((e^2, w^2, \mathcal{C}^2))$$

This result allows us to remove unnecessary transitions from graph $G_{\mathcal{J}}$: if the upper bound for arc a is lower than a known lower bound for the problem, then arc a and the related transition cannot be in an optimal solution of the relaxation defined by $\mathcal{J}$. We illustrate the filtering process in Figure 2.2.6, whic pictures the graph obtained after applying the filtering phase to the graph in Figure 2.2.4. This eliminates from the graph in all arcs that only belong to paths whose profit is less than a lower bound equal to 202. The filtering also works if a value is smaller than the optimum, although fewer arcs may be removed. We keep the same Lagrangian multipliers for the filtering.

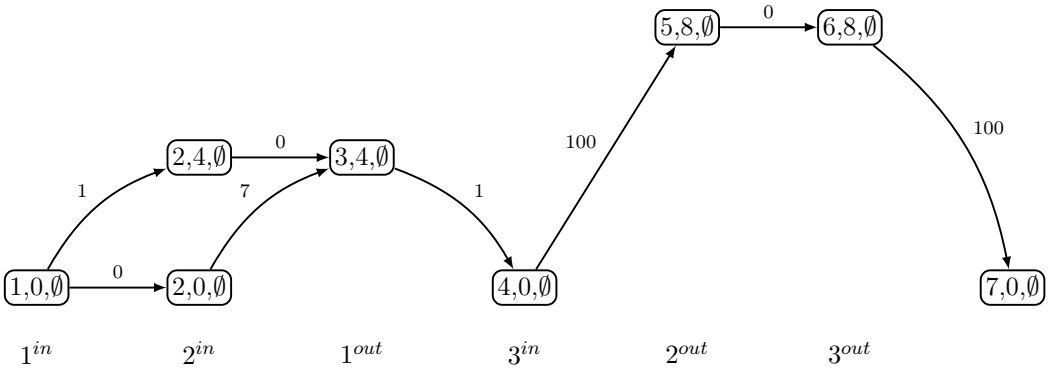


Figure 2.2.6: Graph representation of the relaxed DP for $\mathcal{J} = \emptyset$, with Lagrangian multipliers $(0, -3, 0)$ after filtering arcs whose Lagrangian profit is not large enough (using a lower bound equal to 202).

The efficiency of SSDP lies in the fact that the corresponding arcs in subsequent stronger relaxations can be removed as well. However, to our knowledge, the validity of this permanent removal is only implicitly assumed in the literature. We now formally prove of this property in our specific context. The following lemma shows that the set of arcs going out of each vertex of a refined relaxation related to $\mathcal{J}$' is a subset of the arcs going out of the vertex it comes from, through the sublimation of the relaxation related to $\mathcal{J}$.

Lemma 2.2.2. *Let us consider $e \in \{1, \ldots, 2n+1\}$, $w \in \{0, \ldots, W\}$, $\mathcal{J}$ and $\mathcal{J}$' such that $\mathcal{J} \subseteq \mathcal{J}' \subseteq \mathcal{I}$, $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$ and $v' = (e, w, \mathcal{C}') \in V_{\mathcal{J}'}$ such that $\mathcal{C} \cap \mathcal{J} = \mathcal{C}' \cap \mathcal{J}$ (i.e. vertex v' comes from the sublimation of vertex v). Then $\cup_{s \in \mathcal{S}_{\mathcal{J}}((e,w,\mathcal{C}))} \psi(s) \supseteq \cup_{s \in \mathcal{S}_{\mathcal{J}'}((e,w,\mathcal{C}'))} \psi(s)$.*

Proof. Let $(e, w, \mathbf{d}) \in S_{\mathcal{J}'}(e, w, \mathcal{C}')$. Then by definition of $S_{\mathcal{J}}$, for all $i \in \mathcal{J}'$, $d_i = 1 \leftrightarrow i \in \mathcal{C}'$. Since $\mathcal{J} \subset \mathcal{J}'$, for all $i \in \mathcal{J}$, $d_i = 1 \leftrightarrow i \in \mathcal{C}'$, and so $d_i = 1 \leftrightarrow i \in \mathcal{C}$ because $\mathcal{C} \cap \mathcal{J} = \mathcal{C}' \cap \mathcal{J}$. Thus $(e, w, \mathbf{d}) \in S_{\mathcal{J}'}(e, w, \mathcal{C})$, from which the result follows. □

The next lemma formally shows that for a given vector of multipliers $\boldsymbol{\pi}$ the Lagrangian cost of taking a decision from a specific state cannot increase when the relaxation is refined.

Lemma 2.2.3. *Let us consider $e \in \{1, \ldots, 2n+1\}$, $w \in \{0, \ldots, W\}$, $\mathcal{J}$ and $\mathcal{J}'$ such that $\mathcal{J} \subseteq \mathcal{J}' \subseteq \mathcal{I}$, $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$ and $v' = (e, w, \mathcal{C}') \in V_{\mathcal{J}'}$ such that $\mathcal{C} \cap \mathcal{J} = \mathcal{C}' \cap \mathcal{J}$ (i.e. vertex v' comes from the sublimation of vertex v) and $\boldsymbol{\pi} \in \mathbb{R}^n$ a vector of Lagrangian multipliers. Then $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e, w, \mathcal{C}) \geq \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e, w, \mathcal{C}')$ and $\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}}(e, w, \mathcal{C}) \geq \hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e, w, \mathcal{C}')$.*

Proof. We proceed by induction on e to prove the part of the proposition involving $\hat{\alpha}$. A straightforward adaptation of the proof yields the result for $\hat{\gamma}$. At rank $e = 2n+1$, the property is satisfied since we have $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e, 0, \mathcal{C}) = \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e, 0, \mathcal{C}') = 0$ and $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e, w, \mathcal{C}) = \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e, w, \mathcal{C}') = -\infty$ if $w \neq 0$, $\mathcal{C} \neq \emptyset$ or $\mathcal{C}' \neq \emptyset$.

At rank $e \in \{1, \ldots, 2n\}$, assume that $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e+1, \bar{w}, \bar{\mathcal{C}}) \geq \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e+1, \bar{w}, \bar{\mathcal{C}}')$ for all $(e+1, \bar{w}, \bar{\mathcal{C}}) \in V_{\mathcal{J}}$ and $(e+1, \bar{w}, \bar{\mathcal{C}}') \in V_{\mathcal{J}'}$ such that $\bar{w} \in \{0, \ldots, W\}$ and $\bar{\mathcal{C}} \cap \mathcal{J} = \bar{\mathcal{C}}' \cap \mathcal{J}$. Then for all $(e, w, \mathcal{C}) \in V_{\mathcal{J}}$ and $(e, w, \mathcal{C}') \in V_{\mathcal{J}'}$ such that $w \in \{0, \ldots, W\}$ and $\mathcal{C} \cap \mathcal{J} = \mathcal{C}' \cap \mathcal{J}$, and for all $(\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p) \in \cup_{s \in \mathcal{S}_{\mathcal{J}}((e,w,\mathcal{C}))} \psi(s)$, we have

$$\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e + \Delta_e, w + \Delta_w, \mathcal{C}_+) \geq \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e + \Delta_e, w + \Delta_w, \mathcal{C}'_+)$$

with $\mathcal{C}_+ = \mathcal{C} \cup \{i \in \mathcal{I} : (\Delta_d)_i = 1\} \setminus \{i \in \mathcal{I} : (\Delta_d)_i = -1\}$ and $\mathcal{C}'_+ = \mathcal{C}' \cup \{i \in \mathcal{I} : (\Delta_d)_i = 1\} \setminus \{i \in \mathcal{I} : (\Delta_d)_i = -1\}$. Indeed, $\mathcal{C}_+ \cap \mathcal{J} = \mathcal{C}'_+ \cap \mathcal{J}$. From (2.2.13) and Lemma 2.2.2, we have $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e, w, \mathcal{C}) \geq \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e, w, \mathcal{C}')$. □

The proposition below is crucial for the efficiency of the SSDP algorithm: it shows that once an arc is eliminated from a graph at a given iteration, all corresponding arcs can be removed from the graphs built during subsequent iterations.

Proposition 2.2.4. *Let LB be a valid lower bound for the problem, $\mathcal{J} \subseteq \mathcal{I}$, $a \in A_{\mathcal{J}}$ such that $\tau(a) = (e^1, w^1, \mathcal{C}^1)$, $h(a) = (e^2, w^2, \mathcal{C}^2)$, $\mu(a) = (\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p)$, and $\boldsymbol{\pi} \in \mathbb{R}^n$. If $\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}}(e^1, w^1, \mathcal{C}^1) + < \Delta_{\mathbf{d}}, \boldsymbol{\pi} > + \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}(e^2, w^2, \mathcal{C}^2) < LB$, then the shortest path problem in $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ without arc a is a relaxation of (2.2.4)-(2.2.11). Moreover, for any $\mathcal{J}' \supseteq \mathcal{J}$ and $\boldsymbol{\pi}' \in \mathbb{R}^n$, the shortest path problem $G^{\boldsymbol{\pi}'}_{\mathcal{J}'}$ without any arc related to transition $\mu(a)$ from states $(e^1, w^1, \mathcal{C}^{1'})$ such that $\mathcal{C}^{1'} \cap \mathcal{J} = \mathcal{C}^1 \cap \mathcal{J}$ is a relaxation of (2.2.4)-(2.2.11) as well.*

Proof. First, we prove the validity of the proposition for the same vector $\boldsymbol{\pi}$ and a relaxation refined by enforcing a larger set of consistency constraints $\mathcal{J}'$. Let us consider arc $b \in \mathcal{A}_{\mathcal{J}'}$, such that $\mu(a) = \mu(b)$, $\tau(b) = (e^1, w^1, \mathcal{C}^{1'})$ such that $\mathcal{C}^{1'} \cap \mathcal{J} = \mathcal{C}^1 \cap \mathcal{J}$. Then $h(b) = (e^2, w^2, \mathcal{C}^{2'})$, such that $\mathcal{C}^{2'} \cap \mathcal{J} = \mathcal{C}^2 \cap \mathcal{J}$. Indeed, $\mathcal{C}^{2'} \cap \mathcal{J} = (\mathcal{C}^{1'} \cap \mathcal{J}) \cup (\{i \in \mathcal{I} : (\Delta_d)_i = 1\} \cap \mathcal{J}) \setminus \{i \in \mathcal{I} : (\Delta_d)_i = -1\} = \mathcal{C}^1 \cup (\{i \in$

$\mathcal{I} : (\Delta_d)_i = 1\} \cap \mathcal{J}) \setminus \{i \in \mathcal{I} : (\Delta_d)_i = -1\} = \mathcal{C}^2 \cap \mathcal{J}$. Hence Lemma 2.2.3 implies that $\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e^1, w^1, \mathcal{C}^{1\prime}) + < \boldsymbol{\Delta}_{\mathbf{d}}, \boldsymbol{\pi} > + \hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}'}(e^2, w^2, \mathcal{C}^{2\prime}) < LB$. Arc b being part of a feasible solution of the relaxation defined by $G^{\boldsymbol{\pi}}_{\mathcal{J}'}$ that is optimal for the problem would contradict LB being a lower bound. It follows that b can be removed from $G^{\boldsymbol{\pi}}_{\mathcal{J}'}$ that will still define a relaxation of (2.2.4)-(2.2.11).

Second, we prove the validity of the proposition for a fixed set of consistency constraints $\mathcal{J}$ and a different vector of multipliers $\boldsymbol{\pi}'$. Lemma 2.2.1 shows that arc u cannot be part of a feasible solution of the relaxation associated with $G^{\boldsymbol{\pi}}_{\mathcal{J}}$ that would be optimal (and feasible) for the problem, since that would imply that LB is not a lower bound. Hence, no solution using a in $G^{\boldsymbol{\pi}'}_{\mathcal{J}}$, $\boldsymbol{\pi}' \in \mathbb{R}^n$ can be optimal for the problem, and removing a does not remove optimal solutions of the problem from those relaxations. □

Values $\hat{\alpha}^{\boldsymbol{\pi}}_{\mathcal{J}}((e, w, \mathcal{C}))$ and $\hat{\gamma}^{\boldsymbol{\pi}}_{\mathcal{J}}((e, w, \mathcal{C}))$ can be computed for all nodes $(e, w, \mathcal{C}) \in V_{\mathcal{J}}$ in two passes using Bellman's forward and backward dynamic programming algorithm, respectively.

If an arc is filtered from a graph $G^{\boldsymbol{\pi}}_{\mathcal{J}}$, it is filtered in the graph representation $(V_{\mathcal{J}}, A_{\mathcal{J}})$, and all vertices with no predecessors or no successors are removed from $V_{\mathcal{J}}$. The corresponding states and transitions will not be considered in subsequent iterations.

2.2.2.6 Sublimation and convergence

In SSDP, the sublimation phase consists in strengthening the current relaxation by enforcing some constraints that are violated in the current solution. In our application, the set $\mathcal{J}$ of consistency constraints taken into account in the DP is extended by adding new ones, defining $\mathcal{K} \supset \mathcal{J}$. Let $\rho_{\mathcal{J}}$ be the function that associates with state $s = (e, w, \mathbf{d})$ the set of transitions that were not filtered up to the iteration related to set $\mathcal{J}$.

$$\rho_{\mathcal{J}}((e, w, \mathbf{d})) = \begin{cases} \emptyset & \text{if } \hat{v}_{\mathcal{J}}((e, w, \mathbf{d})) \notin V_{\mathcal{J}} \\ \{\mu(a) : a \in \Gamma^+(\hat{v}_{\mathcal{J}}(e, w, \mathbf{d}))\} & \text{otherwise} \end{cases}$$

The sublimation phase builds a new graph $G_{\mathcal{K}}$ from filtered graph $G_{\mathcal{J}}$ using the following modified transition function $\hat{\psi}_{\mathcal{K}}$:

$$\hat{\psi}_{\mathcal{K}}((e, w, \mathbf{d})) = \psi((e, w, \mathbf{d})) \cap \rho_{\mathcal{J}}((e, w, \mathbf{d}))$$

This function defines the set of transitions going out of a given state $(e, w, \mathbf{d})$. These transitions should not have been discarded by filtering during previous iterations (*i.e.*, they should be in $\rho_{\mathcal{J}}((e, w, \mathbf{d}))$). Also, the latter set might contain infeasible transitions for $(e, w, \mathbf{d})$ specifically since it gathers all transitions from states which are not distinguishable from $(e, w, \mathbf{d})$ in the previous relaxation (*i.e.* from all the states associated with $\hat{v}_{\mathcal{J}}(e, w, \mathbf{d})$ in $V_{\mathcal{J}}$). Thus, only the transitions that are in $\psi((e, w, \mathbf{d}))$ as well are kept. The maximum number of iterations of the algorithm is n, since at least one item index is added to $\mathcal{J}$ at each sublimation step,

and when $\mathcal{J} = \mathcal{I}$, the relaxation obtained is equivalent to (2.2.4)-(2.2.11) (Observation 2.2.2). However, a feasible solution may be found at step 5 when $\mathcal{J} \neq \mathcal{I}$. In the latter case, the cost of this solution in model (2.2.14)-(2.2.16) is equal to its cost in (2.2.4)-(2.2.11), so that it provides dual and primal bounds with the same value and the algorithm terminates with this optimal solution.

In Figure 2.2.7, we report the graph obtained after the sublimation step from $\mathcal{J} = \emptyset$ to $\mathcal{J} = \{2\}$ (that adds dimension 2 to the state space). Note that only arcs that are represented by an arc in the graph in Figure 2.2.6 are created. When $\mathcal{J} = \{2\}$, vertex $(3, 4, \{2\})$ is eliminated, since the only possible arc from vertex $(3, 4, \emptyset)$ (when $\mathcal{J} = \emptyset$) removes item 1, which would lead to an infeasible state (total weight of 0 and item 2 included in the knapsack).

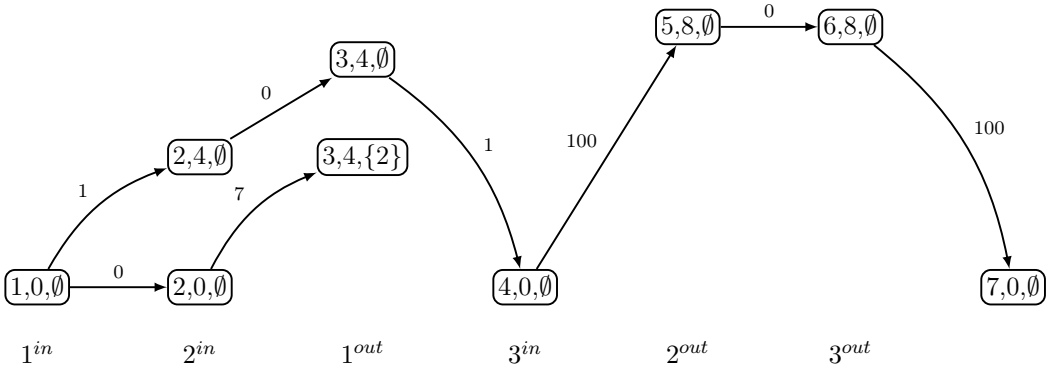


Figure 2.2.7: Graph representation built from the filtered graph in Figure 2.2.6 by adding dimension 2 to the state space.

2.2.3 Refinements of SSDP to solve TKP effectively

Preliminary computational experiments showed that a direct implementation of SSDP for TKP is not able to produce results that can compete with state-of-the-art TKP solvers. This can be explained by several issues: the method requires a long computation time to build the first relaxation, many states that are not useful are generated when the first relaxation is computed, and the gap is not reduced significantly when only one constraint at a time is reintroduced in the sublimation phase. We now propose several techniques to deal with these issues, and improve the performance of SSDP for solving TKP.

2.2.3.1 Attaching additional information to the states

Let $\mathcal{J}$ be the index set of constraints that are currently taken into account in the dynamic program. For a given vertex $v = (e, w, \mathcal{C})$ such that $e \in \mathcal{E}^{\text{out}}$, if $i(e) \notin \mathcal{J}$, two transitions are possible (removing item $i(e)$ or not). In some cases, all states

from the original state-space represented by vertex v contain $i(e)$. In some others, not any one of such states contains $i(e)$. In both cases, only one transition should be created.

To detect these cases, we attach to each vertex v an additional vector $\mathbf{d}^\eta(v) \in \{0,1,\emptyset\}^n$. If $\mathbf{d}^\eta(v)_i = 0$, v represents only states where i is not in the knapsack, while $\mathbf{d}^\eta(v)_i = 1$ means that v represents only states where i is in the knapsack. If $\mathbf{d}^\eta(v)_i = \emptyset$, v represents both types of states.

For $i \notin \mathcal{J}$, we set $\mathbf{d}^\eta(v^0)_i = 0$, and $\mathbf{d}^\eta(v)_i$ is computed recursively for each other vertex v as follows:

$$\mathbf{d}^\eta(v)_i = \begin{cases} 1 & \text{if } \forall a = (e+\Delta_e, w+\Delta_w, \mathbf{d}+\Delta_{\mathbf{d}}) \in \Gamma^-(v), \\ & \qquad \big(\mathbf{d}^\eta(\tau(a))_i = 1 \text{ and } \Delta_{\mathbf{d}} \neq -\boldsymbol{\varepsilon}_i\big) \text{ or } \Delta_{\mathbf{d}} = +\boldsymbol{\varepsilon}_i \\ 0 & \text{if } \forall a = (e+\Delta_e, w+\Delta_w, \mathbf{d}+\Delta_{\mathbf{d}}) \in \Gamma^-(v), \\ & \qquad \big(\mathbf{d}^\eta(\tau(a))_i = 0 \text{ and } \Delta_{\mathbf{d}} \neq +\boldsymbol{\varepsilon}_i\big) \text{ or } \Delta_{\mathbf{d}} = -\boldsymbol{\varepsilon}_i \\ \emptyset & \text{otherwise} \end{cases}$$

To illustrate the computation of this information, take the graph in Figure 2.2.3. Let v be vertex $(2,4,\emptyset)$. This vertex can only be obtained by adding item 1, and the other items have not been considered yet. Therefore, for this vertex, $\mathbf{d}^\eta(v) = (1,0,0)$. On the contrary, let v' be vertex $(3,4,\emptyset)$. This vertex can be obtained by either selecting item 2, or from vertex $(2,4,\emptyset)$, so $\mathbf{d}^\eta(v') = (\emptyset,\emptyset,0)$.

Consequently, vector $\mathbf{d}^\eta(v)$ is computed on the fly when $(V_{\mathcal{J}}, A_{\mathcal{J}})$ is created. We attach another piece of information to each vertex v, which corresponds to redundant constraints. For each vertex $v = (e, w, \mathcal{C})$, we define $q^\eta_{\min}(v)$ (resp. $q^\eta_{\max}(v)$) as a lower (resp. upper) bound on the number of items that can be in the knapsack in the states of $\mathcal{S}_{\mathcal{J}}(v)$. These values can be computed recursively, in a similar way to vector $\mathbf{d}^\eta(v)$. We set $q^\eta_{\min}(v^0) = 0$ and for each other vertex v, $q^\eta_{\min}(v) = \min_{a=(\Delta_e,\Delta_w,\Delta_{\mathbf{d}},p)\in\Gamma^-(v)}\{q^\eta_{\min}(\tau(a)) + < \Delta_d.\mathbf{1} >\}$. Value $q^\eta_{\max}(v)$ can be obtained by replacing min by max in the expression. As an example, consider the graph in Figure 2.2.3. Although we are not able to know the configuration related to vertex $(3,4,\emptyset)$, we know that all configurations represented by the vertex contain exactly one item.

Let $\mathcal{I}(e) = \{i \in \mathcal{I} : s_i \leq t(e) < f_i\}$ be the set of items that may belong to the knapsack when event e occurs. For each event e, let $Q^{\max}(e) = \max\{|S| : S \subseteq \mathcal{I}(e), \sum_{i\in S} w_i \leq W\}$ be the maximum number of items that can belong to the knapsack when this event occurs (this value can be computed in time linear in $|\mathcal{I}(e)|$ for each event e when the elements of this set are sorted by non-decreasing order of size). Obviously, for vertex $v = (e, w, \mathcal{C})$, the number of items in any valid state represented by v is in $[0, Q^{\max}(e)]$.

2.2.3.2 Feasibility tests

In the rest of this section, a *feasible state* is defined as a state that can be generated from s^0 by applying a feasible sequence of transitions following recurrence equations

(2.2.12). We denote by $\mathcal{S}^+$ the set of feasible states. We recall that $V_{\mathcal{J}}$ is the set of vertices considered when solving the relaxation related to $\mathcal{J}$. Note that any vertex in $V_{\mathcal{J}}$ that is not related to a state in $\mathcal{S}^+$ can be removed from the graph without impairing the validity of the algorithm. We now describe several techniques used to detect infeasibilities of such vertices. The following results are stated without proof.

The first feasibility test checks that the number of items in the knapsack is consistent with the list of items that are either present or absent in all states represented by a vertex v.

Proposition 2.2.5. *Let $\mathcal{J} \subseteq \mathcal{I}$ and $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$. If $card(\{i \in \mathcal{I} : \mathbf{d}^\eta(v)_i \neq 0\}) < q^\eta_{\min}(v)$ or $card(\{i \in \mathcal{I} : \mathbf{d}^\eta(v)_i = 1\}) > q^\eta_{\max}(v)$ then $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$.*

For example in Figure 2.2.3, vertex $(4, 8, \emptyset)$ would be eliminated, since the minimum number of items in the configuration represented by the vertex is 2, and for this layer, there cannot be more than one item.

Another feasibility test is based on the set of possible weights of subsets of items that can belong to the knapsack at a given event. For this purpose, we precompute for each event e the following set: $\mathcal{F}(e) = \{\sum_{i \in S} w_i : S \subseteq \mathcal{I}(e), \sum_{i \in S} w_i \leq W\}$, which corresponds to all reachable weights of a subset of items. Each of these sets can be computed in $\mathcal{O}(nW)$-time using a straightforward dynamic programming algorithm. Proposition (2.2.6) follows from the definition of $\mathcal{F}$.

Proposition 2.2.6. *Let $\mathcal{J} \subseteq \mathcal{I}$ and $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$. If $w \notin \mathcal{F}(e)$ then $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$.*

This rule also serves to remove vertex $(4, 8, \emptyset)$ from the graph in Figure 2.2.3, since for this layer, there are no possible item combinations whose size is 8 (the only possible values are 0 and 4).

This rule can be improved by considering additional information gathered from $\mathbf{d}^\eta(s)$. We precompute, for each event e and each item $i \in \mathcal{I}(e)$, $\mathcal{F}^+(e, i)$ and $\mathcal{F}^-(e, i)$, the possible weights that can be reached using item i, and without item i, respectively. We have $\mathcal{F}^+(e, i) = \{\sum_{j \in S \cup \{i\}} w_j : S \subseteq \mathcal{I}(e) \setminus \{i\}, \sum_{j \in S} w_j \leq W - w_i\}$ and $\mathcal{F}^-(e, i) = \{\sum_{j \in S} w_j : S \subseteq \mathcal{I}(e) \setminus \{i\}, \sum_{j \in S} w_j \leq W\}$.

Proposition 2.2.7. *Let $\mathcal{J} \subseteq \mathcal{I}$, $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$ and $i \in \mathcal{I}(e)$. If $\mathbf{d}^\eta(v)_i = 1$ and $w \notin \mathcal{F}^+(e, i)$ then $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$. Likewise, if $\mathbf{d}^\eta(v)_i = 0$ and $w \notin \mathcal{F}^-(e, i)$ $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$.*

The following result allows the detection of nodes related to states that can be generated only by removing an item from the knapsack without adding it first. In such cases, the weight recorded in the state might become lower than the weight of the items whose presence in the knapsack is known for sure.

Proposition 2.2.8. *Let $\mathcal{J} \subseteq \mathcal{I}$ and $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$. If $\sum_{i \in \mathcal{I} : \mathbf{d}^\eta(v)_i = 1} w_i > w$ then $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$.*

For a vertex $(e, w, \mathcal{C})$, the following feasibility test integrates the bounds on the number of items in the knapsack to ensure the consistency of set $\mathcal{C}$ with respect to value w.

Proposition 2.2.9. *Let $\mathcal{J} \subseteq \mathcal{I}$ and $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$. Let $S^1 = \{i \in \mathcal{I}(e) : \mathbf{d}^\eta(v)_i \neq 0\}$ be the set of items potentially in the knapsack, and $S^2 = \{i \in \mathcal{I}(e) : \mathbf{d}^\eta(v)_i = 1\}$ the set of items in the knapsack for sure. If $\max\left\{\sum_{i\in S} w_i : S \subseteq S^1, |S| \leq q^\eta_{\max}(v)\right\} < w$ or $\min\left\{\sum_{i\in S} w_i : S^2 \subseteq S \subseteq S^1, |S| \geq q^\eta_{\min}(v)\right\} > w$ then $\mathcal{S}_{\mathcal{J}}(v) \cap \mathcal{S}^+ = \emptyset$.*

In the classical knapsack problem, an item i is dominated by item j if $p_i \leq p_j$, $w_i \geq w_j$ and one of the two inequalities is strict. In this case, from any feasible solution where item i is chosen but not item j, we can build another solution where j is chosen and whose profit is not smaller than that of the initial solution. Thus, among solutions that include i, only those including j as well need to be considered. For TKP, dominance relations must take into account the temporal aspect as well.

Proposition 2.2.10. *Item i is dominated by item j if $p_i \leq p_j$, $w_i \geq w_j$, $s_i \leq s_j$, $f_i \geq f_j$ and one of the four inequalities is strict.*

In the course of SSDP, we can take advantage from it by modifying the recurrence equations as follows. Let us consider vertex $v = (e, w, \mathcal{C}) \in V_{\mathcal{J}}$ such that $e \in \mathcal{E}^{\text{in}}$, $i(e) = j$ and $\mathbf{d}^\eta(v)_i = 1$ with i an item which is dominated by item j. Then any transition $(\Delta_e, \Delta_w, \Delta_{\mathbf{d}}, p) \in \cup_{s\in\mathcal{S}_{\mathcal{J}}(v)}\psi(s)$ with $(\Delta_{\mathbf{d}})_j = 0$ can be discarded from equation (2.2.13). Indeed, selecting this transition would mean choosing item i but not item j.

2.2.3.3 Partial enumeration of transitions

Combining several consecutive transitions into single compound transitions allows the enforcement of some consistency constraints locally, even if the corresponding dimensions are not included in the current state space.

Our implementation of this idea uses an input parameter k^{enum}, which controls the depth of the enumeration of consecutive transitions. More precisely, from a given state, instead of computing the two possible transitions, we compute the $O(2^{k^{\text{enum}}})$ possible sequences of k^{enum} successive transitions. Sequences that violate consistency constraints are not generated. Figure 2.2.8 illustrates a case where three consecutive events are considered.

In some cases this technique might lead to a larger network. However, when the enumeration of some transitions spans the two events of the same item, the associated consistency constraint is always satisfied, which results in a stronger relaxation. Moreover, the filtering procedure may filter a compound transition because the related sequence of transitions has a poor cost, while in the initial graph, each intermediate arc of this sequence may individually belong to a path with a better cost, preventing the removal of any transition. This contributes to limiting the growth of the network, and improves the quality of the relaxation at the same time.

2.2.3.4 Criteria for selecting constraints to reintroduce

At each sublimation step, one has to select the dimensions related to violated constraints that are integrated into the state-space. To make this selection, we record

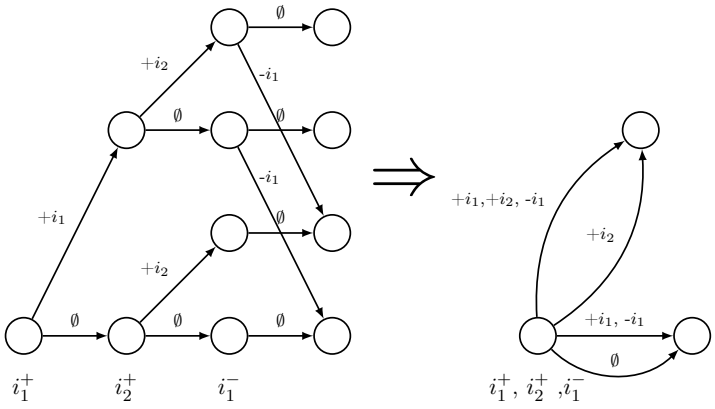


Figure 2.2.8: Partial enumeration of three events. Four sequences out of eight are feasible.

some information that will be used in our method. At the iteration determined by set $\mathcal{J}$, we record a list $(\boldsymbol{\pi}^1, \ldots, \boldsymbol{\pi}^{k^{\text{nbsol}}})$ of Lagrangian multipliers that led to the best upper bounds in the Volume algorithm. This list is sorted by decreasing value of $L_{\mathcal{J}}(\boldsymbol{\pi}^q)$. For each recorded vector $\boldsymbol{\pi}^q$, let $\boldsymbol{y}^q$ be the corresponding solution expressed in terms of the variables of (2.2.14)-(2.2.16).

First, we do not consider all dimensions for inclusion in the state-space. Only those that are related to violated constraints are considered. Let $\mathcal{J}^{\neq} = \left\{ i \in \mathcal{I} \setminus \mathcal{J} : \exists q \in \{1, \ldots, k^{\text{nbsol}}\}, \boldsymbol{y}^q_{e^{in}(i)} \neq \boldsymbol{y}^q_{e^{out}(i)} \right\}$ be the set of items whose consistency constraint is violated in at least one of the solutions of $(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^{k^{\text{nbsol}}})$. The corresponding dimensions are natural candidates to be considered for the sublimation phase.

We now describe three criteria for estimating the computational attractiveness of adding a specific dimension to set $\mathcal{J}$.

The first criterion (**Lagrangian Multipliers**) is to use the best Lagrangian multipliers $\boldsymbol{\pi}^1$ found to determine the attractiveness of each consistency constraint: $\psi^1_i = |\boldsymbol{\pi}^1_i|$. A Lagrangian cost of large magnitude tends to indicate that many solutions violate the corresponding constraint, which is penalized by Volume.

The second criterion (**Network Size**) aims to control the number of states in the network after the sublimation step. For each constraint, we compute an estimation of the growth of the vertex set if the corresponding constraint – and only that one – is included in the state-space. When adding a single constraint related to item i, only states s such that $\mathbf{d}^{\eta}(s)_i = \emptyset$ can yield two different states after sublimation. Hence, the second criterion we define is the opposite (smaller is better) of an upper bound on the number of additional states due to i: $\psi^2_i = -|v \in V_{\mathcal{J}} : \mathbf{d}^{\eta}(v)_i = \emptyset|$.

The third criterion (**Number of violations**) favors the constraints that are violated in many solutions recorded in the list $(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^{k^{\text{nbsol}}})$. This can be computed as follows: $\psi^3_i = \frac{1}{k^{\text{nbsol}}} \sum_{q=1}^{k^{\text{nbsol}}} \left| \boldsymbol{y}^q_{e^{in}(i)} - \boldsymbol{y}^q_{e^{out}(i)} \right|$.

2.2.3.5 Reintroducing batches of constraints

Adding several violated constraints at once is generally a good strategy for TKP. The sublimation step is a time-consuming procedure, and preliminary experiments have shown that in many cases, adding only one constraint is not sufficient to ensure a significant decrease in the dual bound. However, adding too many constraints dramatically increases the size of the network. Therefore, we have to find a good trade-off between the quality of the bound and the size of the network.

A first issue is to compute a reliable estimation of the size of the network when a new constraint is introduced. The following proposition provides an upper bound on the number of labels in the network given a set of consistency constraints enforced in the state space.

Proposition 2.2.11. *Let $\mathcal{J}$ and $\mathcal{K}$ such that $\mathcal{J} \subseteq \mathcal{K} \subseteq \mathcal{I}$. It holds that*

$$|V_{\mathcal{K}}| \leq \sum_{e=1,\dots,2n+1} \left(2^{|(\mathcal{K}\setminus\mathcal{J})\cap\mathcal{I}(e)|} card(\{(e', w, \mathcal{C}) \in V_{\mathcal{J}} : e' = e\})\right).$$

Proof. The set of vertices created in $V_{\mathcal{K}}$ from a given vertex $(e, w, \mathcal{C}) \in V_{\mathcal{J}}$ is included in the set

$$\begin{aligned}\{(e, w, \mathbf{d}') :& \mathbf{d}'_i = \mathbf{d}_i \ \forall i \in \mathcal{J}, \\ & \mathbf{d}'_i \in \{0, 1\} \ \forall i \in (\mathcal{K} \setminus \mathcal{J}) \cap \mathcal{I}(e), \\ & \mathbf{d}'_i = 0 \ \forall i \in \mathcal{I} \setminus (\mathcal{J} \cup (\mathcal{K} \cap \mathcal{I}(e)))\}\end{aligned}$$

whose cardinality is $2^{|(\mathcal{K}\setminus\mathcal{J})\cap\mathcal{I}(e)|}$. Summing up over all states in $V_{\mathcal{J}}$ yields the result. □

This indicates that adding constraints related to items with pairwise disjoint time windows is particularly attractive: in such cases, for all events $e \in \{1, \dots, 2n+1\}$ we have $\text{card}((\mathcal{K} \setminus \mathcal{J}) \cap \mathcal{I}(e)) \leq 1$. It follows that the network grows by a constant factor only, as stated formally in the next corollary.

Let $G^{\text{int}} = (\mathcal{I}, E^{\text{int}})$ be the interval graph related to intervals $[s_i, f_i), i \in \mathcal{I}$. An arc in this graph represents a pair of dimensions that should not be added together (if one wants to avoid overly rapid growth of the network size). We also define the subgraph of G^{int} induced by $\mathcal{J}^{\neq}$: $G^{\text{int}}_{\neq} = (\mathcal{J}^{\neq}, E^{\text{int}}_{\neq})$, where $E^{\text{int}}_{\neq} = E^{\text{int}} \cap (\mathcal{J}^{\neq} \times \mathcal{J}^{\neq})$.

Corollary 2.2.12. *Let $\mathcal{J}$ and $\mathcal{K}$ be such that $\mathcal{J} \subseteq \mathcal{K} \subseteq \mathcal{I}$, and $\mathcal{K} \setminus \mathcal{J}$ is a stable set in graph G^{int}. Then $|V_{\mathcal{K}}| \leq 2|V_{\mathcal{J}}|$.*

This corollary guided our strategies to select the set of dimensions that are added at each sublimation step. We propose four strategies that aim at finding a good tradeoff between the approximate growth of the network and the quality of the relaxation.

The first strategy, which we call **Weighted stable set**, limits the expected network growth during the sublimation step. To this end, we use the upper bound on the number of additional states in the new DP provided by Corollary 2.2.12: when

set $\mathcal{J} \subseteq \mathcal{K}$ forms a stable set in G^{int}, $\sum_{i \in \mathcal{J} \subseteq \mathcal{K}} \psi_i^2$ is an upper bound on the overall number of the new states. This upper bound allows us to account for the effect of including item i in set $\mathcal{J}$. Note that in [Tanaka & Fujikuma 2012], the authors try to heuristically restrain the growth of the DP by choosing the constraints impacting the smallest number of arcs. Here, we rely on properties of our specific problem, allowing for a stricter control of the number of additional states. Based on criteria ψ_i^1, ψ_i^2 and ψ_i^3, we define a weight ψ_i for each item. We then search for a stable set in G^{int} of maximum weight, such that the estimated growth of the number of states is less than parameter MAXG.

This is done by solving the model below, where a binary variable x_i is created for all $i \in \mathcal{J}^{\neq}$, indicating whether i is selected or not.

$$\max \left\{ \sum_{i \in \mathcal{J}^{\neq}} \psi_i x_i : \sum_{i \in \mathcal{J}^{\neq}} -\psi_i^2 x_i \leq \text{MAXG}, \right.$$
$$\left. x_i + x_j \leq 1 \ \forall (i,j) \in E^{\text{int}}_{\neq}, x_i \in \{0,1\} \ \forall i \in \mathcal{J}^{\neq} \right\}$$

This problem is a knapsack problem with conflicts, which is NP-complete, but can be solved in pseudo-polynomial time through dynamic programming for interval graphs (see [Sadykov & Vanderbeck 2013]). For the instances we considered, the time needed to solve this subproblem is negligible compared to the overall time of the algorithm.

Our second strategy, called **Cardinality constrained stable set**, aims at circumventing a major drawback of the Weighted stable set strategy, which sometimes adds too few new constraints, leading to a slow convergence of the overall algorithm. Our strategy consists first in solving the model above with unit profits to find a stable set of maximum cardinality (which we denote by $C_{\max}$). We then seek a stable set of cardinality larger than $C_{\max} * k^{\text{rstable}}$ where k^{rstable} is a parameter in $(0,1]$, by solving the following cardinality constrained maximum weight stable set problem. Using the same variables as the model above, one obtains the following model.

$$\max \left\{ \sum_{i \in \mathcal{J}^{\neq}} \psi_i x_i : \sum_{i \in \mathcal{J}^{\neq}} x_i \geq C_{\max} * k^{\text{rstable}}, \right.$$
$$\left. x_i + x_j \leq 1 \ \forall (i,j) \in E^{\text{int}}_{\neq}, x_i \in \{0,1\} \ \forall i \in \mathcal{J}^{\neq} \right\}$$

This problem is also NP-complete, but is solved effectively by modern MILP solvers (*i.e.*, the time needed to solve it is also negligible compared to the overall time of the algorithm).

The third strategy, called ***k*-coloring**, considers not only stable sets in $G^{\text{int}}_{\neq}$, but in a larger graph also representing constraints that were added in the previous iterations. To this end, we denote this extended set $\mathcal{J}^+ = \mathcal{J} \cup \mathcal{J}^{\neq}$, we define $E^{\text{int}}_+ = E^{\text{int}} \cap (\mathcal{J}^+ \times \mathcal{J}^+)$, and we consider the subgraph $G^{\text{int}}_+ = (\mathcal{J}^{\neq} \cup \mathcal{J}, E^{\text{int}}_+)$

of G^{int} induced by $\mathcal{J}^+$. Let k^{color} be equal to the number of colors that we allow at the current step. At each iteration, we seek a k^{color}-colorable subgraph of G^{int}_+ of maximum weight. If the solution is different from $\mathcal{J}$, then we add the new constraints selected. If the solution only contains $\mathcal{J}$, then we increase the value of k^{color} by one unit, and solve the model again. Initially, k^{color} is set to one. We repeatedly solve the following model, where z_{ij} are binary variables indicating that item i is assigned to color j.

$$\begin{aligned}
\max \sum_{i \in \mathcal{J}^{\neq}} \sum_{j=1,\dots,k^{\text{color}}} & \psi_i z_{ij} \\
z_{ij} + z_{\ell j} &\leq 1, \forall (i,\ell) \in E^{\text{int}}_+, j \in \{1,\dots,k^{\text{color}}\} \\
\sum_{j=1,\dots,k^{\text{color}}} z_{ij} &\leq 1, \forall i \in \mathcal{J}^{\neq} \\
\sum_{j=1,\dots,k^{\text{color}}} z_{ij} &= 1, \forall i \in \mathcal{J}^{+} \\
z_{ij} &\in \{0,1\}, \forall i \in \mathcal{J}^{\neq} \cup \mathcal{J}^{+}, j \in \{1,\dots,k^{\text{color}}\}
\end{aligned}$$

We also solve this subproblem with a general-purpose MILP solver. Again, the solution time is negligible compared to the time needed to build the new graph at each iteration.

Finally, we consider a strategy called **Hybrid**, which favors a strong improvement in the gap in the first iterations, and then favors a network of manageable size when the number of vertices reaches a threshold. This is based on the following observation: in the first iterations, one would like to close the gap between the primal and dual bounds as fast as possible to allow a better-performing filtering procedure, but when the number of vertices in the network becomes large, the most important criterion becomes its size, since a too large network may lead to intractable Lagrangian subproblems. Therefore, the hybrid strategy uses one of the strategies above in the first iterations, and when the number of nodes is larger than a given threshold, the choice is only based on the expected size of the network.

2.2.3.6 Implementation issues

The effectiveness of the filtering step depends heavily on the fact that a good primal solution is known. In general, during the optimization of the Lagrangian multipliers, it may happen that a primal solution is computed as a side product of the method. However, one cannot rely on this for TKP, since many constraints are often violated in a solution of a relaxation. To produce a lower bound for our problem, we heuristically solve model (2.2.1)–(2.2.3) by giving a small amount of time to a general-purpose integer linear programming solver.

A good dual solution is also useful to warmstart the Volume algorithm, which may take a long time to converge when the DAG are large. To find a good set of multipliers, we solve the LP relaxation of (2.2.4)–(2.2.10), and use the optimal dual

values of constraints (2.2.10). Solving the LP relaxation is fast and provides a good starting point for the Volume algorithm.

We implemented a parallel version of Bellman's algorithm. We first compute the longest path (in terms of number of arcs) from s^0 to all vertices. All vertices at the same distance are stored in a common bucket. The treatment of vertices in the same bucket can be done in parallel.

2.2.4 Computational experiments

In this section, we provide experimental results for our methods. For each refinement proposed, we evaluate its impact on the performance of the general algorithm. Finally, we compare our results to those of [Gschwind & Irnich 2017] and to the results obtained using an all-purpose commercial Integer Linear Programming solver. In this section, we consider an instance as solved if the algorithm finds an optimal solution and proves its optimality.

All our experiments are conducted using 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz with 128Gb RAM. For each instance, our code[1] was run on 6 threads and a 32 Gb RAM limit. All models considered in subroutines are solved with IBM ILOG Cplex 12.7.

We use instances proposed in [Caprara *et al.* 2013], composed of two groups. For instances in the first group (I), the number of items ranges from 2071 to 13025, the size of the knapsack is 100, and each item has a profit and a weight between 1 and 100. Since the method described in [Gschwind & Irnich 2017] and our methods can solve all those instances to optimality in a short time, we do not report the corresponding results. For the 100 instances in the second group (called U), the number of items is 1000 and the size of the knapsack ranges from 500 to 520. Each item has a profit and a weight between 1 and 100.

The goal of our experiments is twofold. We first want to determine the best parameters for our algorithm, and evaluate the impact of the different improvements that we have proposed. Secondly, we want to assess their effectiveness against the state-of-the-art methods from the literature.

2.2.4.1 Parameters of the method

We first evaluate the impact of the improvements proposed. For this purpose, we report the results obtained by the best combination of techniques (called SSDP* below), and methods obtained from SSDP* by deactivating some features. We deactivated the partial enumeration technique (subsection 2.2.3.3) and dominance and feasibility tests (subsection 2.2.3.2). For each method, we report in Figure 2.2.9 the number of instances solved along time, with a limit of three hours.

The number of instances solved optimally within 3 hours increases by about 20% when partial enumeration is used. This means that enumerating sequences of tran-

[1]The source code of our algorithms is available at `gitlab.inria.fr/realopt/TemporalKnapsack`

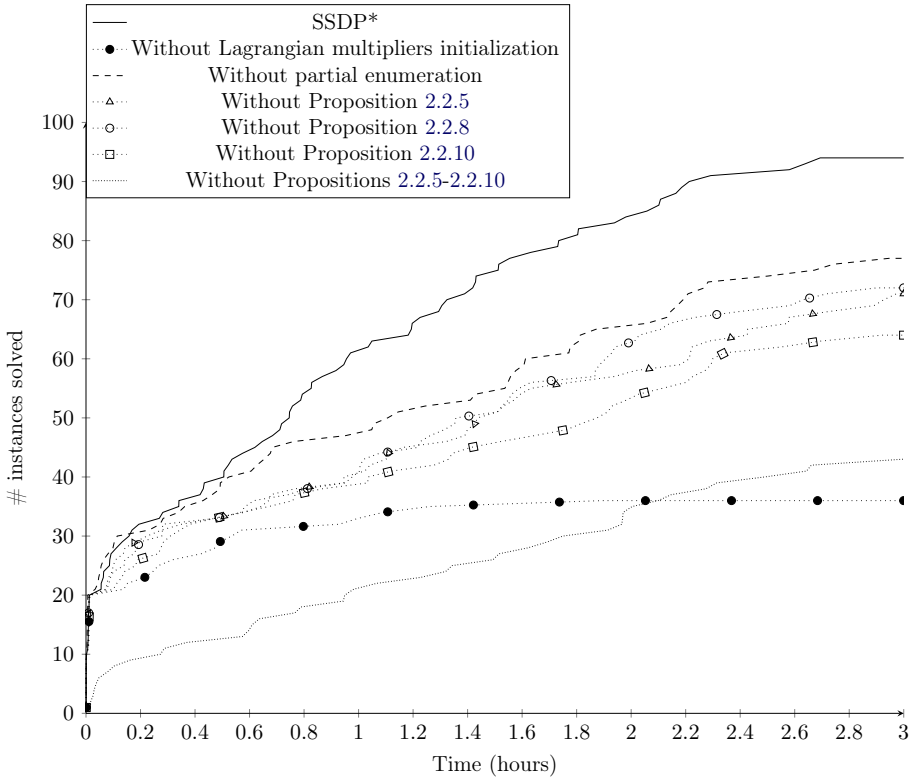


Figure 2.2.9: Number of instances of TKP from data set U [Caprara *et al.* 2016] solved along time, for the best version of our algorithm, and six versions obtained by deactivating some techniques.

sitions allows us to include useful information that is used by the filtering algorithm. To illustrate the effect of partial enumeration, we report in Table 2.2.1 the size of the first network constructed, for different values of k^{enum}. As one might expect, increasing the number of consecutive transitions considered reduces the number of nodes in the network but increases the number of arcs (since combinations of arcs are replaced by single arcs). Although it yields a higher memory consumption and a longer solution time of the relaxations, it can be advantageous to some extent: selecting one arc means deciding more arcs simultaneously and more impact on the Lagrangian cost of the solution. Thus, such long sequences of decisions are more easily discarded by Lagrangian filtering. This also explains, along with the removal of short infeasible sequences, why the network with $k^{\text{enum}} = 2$ has fewer arcs than that with $k^{\text{enum}} = 1$.

The feasibility tests proposed in Propositions 2.2.5 to 2.2.10 have a crucial impact on the performance of our algorithm, since they allow us to solve 40 more instances in a one-hour time limit, and 11 additional instances in a three-hour time limit. These tests remove about 20% of the nodes and 32% of the arcs included in the first network when $k^{\text{enum}} = 4$. One can also remark that each of them improves the overall procedure significantly.

Table 2.2.1: Average size of first network for different value of k^{enum}, after the filtering step. "k" stands for thousands.

k^{enum}	1	2	3	4	5	6
Average nodes	703 k	384 k	264 k	202 k	162 k	135 k
Average arcs	1,392 k	1,344 k	1,639 k	2,269 k	3,155 k	4,658 k

Table 2.2.2: Parameters of the tested methods.

Configuration	Strategy	Criterion ψ_i
SSDP*	Cardinality constrained	$\psi_i^2\left(1-\psi_i^3\right)$
Stable	Weighted stable set	$\frac{-\psi_i^2+\max_{j\in\mathcal{J}^{\neq}}\psi_j^2}{\max_{j\in\mathcal{J}^{\neq}}\psi_j^2}\psi_i^3$
NbLabels	Cardinality constrained	ψ_i^2
Hybrid	Cardinality constrained	First ψ_i^3, then ψ_i^2
LagMult	Cardinality constrained	ψ_i^1
KColor	K-Color	$\frac{-\psi_i^2+\max_{j\in\mathcal{J}^{\neq}}\psi_j^2}{\max_{j\in\mathcal{J}^{\neq}}\psi_j^2}\psi_i^3$

2.2.4.2 Strategies for reintroducing constraints

As stated in most papers working on iterative state-space relaxations, the selection of the constraints to reintroduce is the most critical component in the method. In what follows, we empirically compare our different strategies to determine the most effective. In Table 2.2.2, we report how each configuration of our algorithm is parameterized. Preliminary experiments led us to set parameter $k^{\text{nbsol}} = 20$ for the computation of criterion ψ^3. The performance of the overall method was impaired by values of k^{nbsol} less than 10, but does not seem to be affected by values larger than 20. In the method based on Cardinality constrained strategy, parameter $k^{rstable}$ is empirically fixed to 0.7. For the SSDP* method, the weight associated with each constraint tries to balance the upper bound on the number of additional labels and the frequency of violation of this constraint in the best relaxed solutions considered. When using the Weighted stable set strategy, minimizing the expected number of added labels boils down to selecting no new constraint. That is why we maximize the complement to the maximum number of expected additional labels. This value is weighted by the frequency of violation of the constraint. Configuration Hybrid aims at improving the dual bound as fast as possible by enforcing the most frequently violated constraints. Once the network is too large (we empirically fixed a limit at 4,000k nodes), adding fewer labels is preferred.

Figure 2.2.10 numerically compares our different methods for selecting the constraints to add during the sublimation phase. Similarly to Figure 2.2.9, we report the total number of instances solved along time, with a limit of three hours. We observe that the $k-$coloring strategy is clearly not competitive compared to strategies based on stable sets. The fact that this strategy performs poorly shows that our method to evaluate the size of the network in the stable-set based strategies is

useful, and that one cannot just rely on the interval structure of the constraints. All strategies based on stable sets have similar behaviour. Configuration Stable performs reasonably well within medium time limits. However, it does not seem to be more effective when more time is allocated. This can be explained by the fact that, the larger the network, the less controlled the number of new constraints added. Indeed, we empirically observed that only a few constraints are added in general by this method once a critical size is reached. The configurations based on the Cardinality constrained stable set strategy do not suffer from that drawback. For the group U of instances which are composed of 1000 items, both SSDP* and Stable configurations reintroduce between 5 and 20 constraints in more than 75% of the iterations, while the maximum number of constraints added by the algorithm at each iteration is respectively 29 and 41. The total number of constraints added is lower than or equal to 15 for 20% of the instances, more than 274 (resp. 267) for 20% of the instances for configuration SSDP* (resp. Stable). The maximum number of constraints added for a single instance is 328 (resp. 348).

The performance of Hybrid configuration is disappointing. This might be due to the difficulty of finding a good rule for switching between the two criteria. Overall, an important conclusion is that taking into account the increase in the size of the network appears to be crucial to the method.

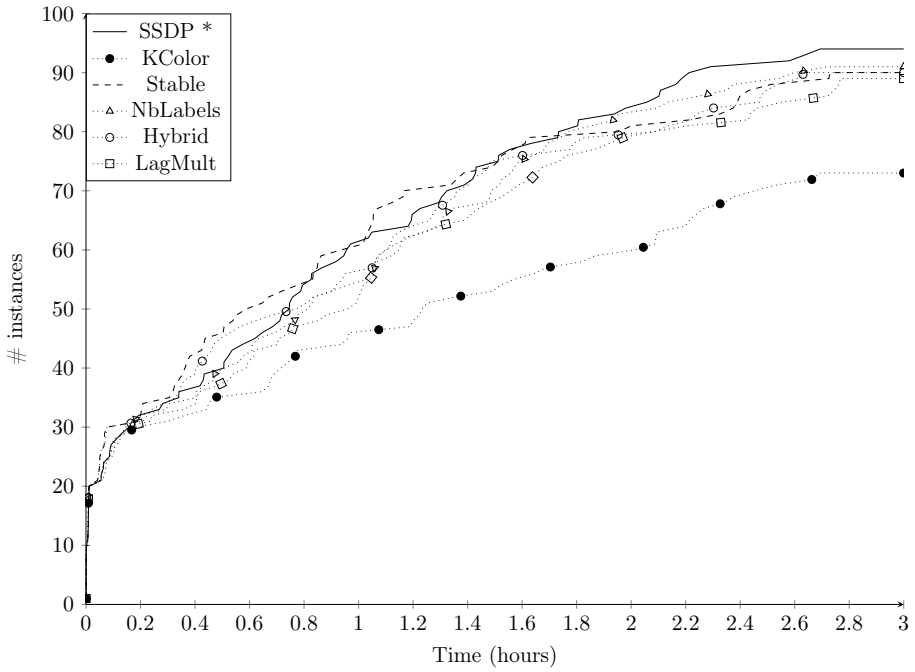


Figure 2.2.10: Number of instances solved along time for different methods used to determine the constraints to add at each iteration.

2.2.4.3 Comparison with the branch-and-price of [Gschwind & Irnich 2017]

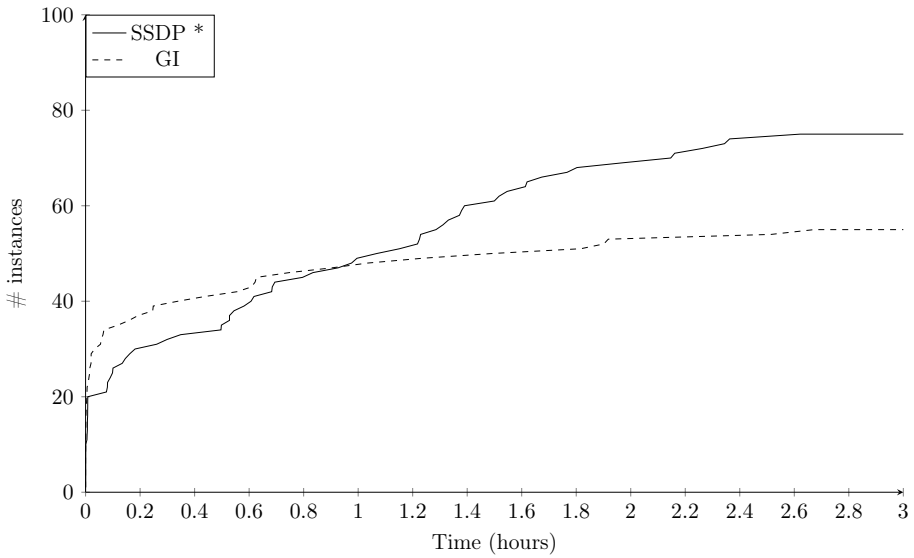


Figure 2.2.11: Number of instances solved along time for our best algorithm (SSDP*) and the algorithm of [Gschwind & Irnich 2017] on Intel(R) Core(TM) i7-2600 at 3.4 GHz with 16.0 GB main memory using a single thread only.

We now compare our method with the algorithm of [Gschwind & Irnich 2017]. The authors kindly provided us with the results obtained with their algorithm within a three-hour time limit. They implemented a pure branch-and-price without any primal heuristics and using best-first as node selection strategy. Their experiments were performed on a standard PC with an Intel(R) Core(TM) i7-2600 at 3.4 GHz with 16.0 GB main memory using a single thread. Figure 2.2.11 reports the performance of our best algorithm (SSDP*) and those obtained by [Gschwind & Irnich 2017](GI) using a similar computer (same processor, same amount of RAM), using a single thread only. The processor speeds in this setting and in the one described at the beginning of Section 2.2.4 are roughly comparable. However, the limited amount of memory on this machine is not always enough for our method (the algorithm ran out of memory for eight instances that are solved on the other machine).

The approach of [Gschwind & Irnich 2017] is more efficient within a short computing time: it solves 41 instances in 30 minutes, whereas the best version of our algorithm, when restricted to a single thread on the same machine, solves only 35 instances. Both algorithms perform similarly within a one-hour time limit (47 instances solved versus 49). However, only a few more instances are solved by the branch-and-price approach within 3 hours of computing time (55 instances in total), while our approach solves 50 percent of the still-unsolved instances between 1 and 3 hours (75 instances in total).

2.2.4.4 Comparison with a general-purpose MILP solver

Figure 2.2.12 reports the performance of our best algorithm (SSDP*), and those obtained by ILP solver IBM Ilog Cplex when solving model (2.2.1)-(2.2.3) (CPLEX) . For the sake of completeness, we also report the results obtained with model (2.2.4)-(2.2.11) (CPLEX-Event-based). We limited all methods to a single thread.

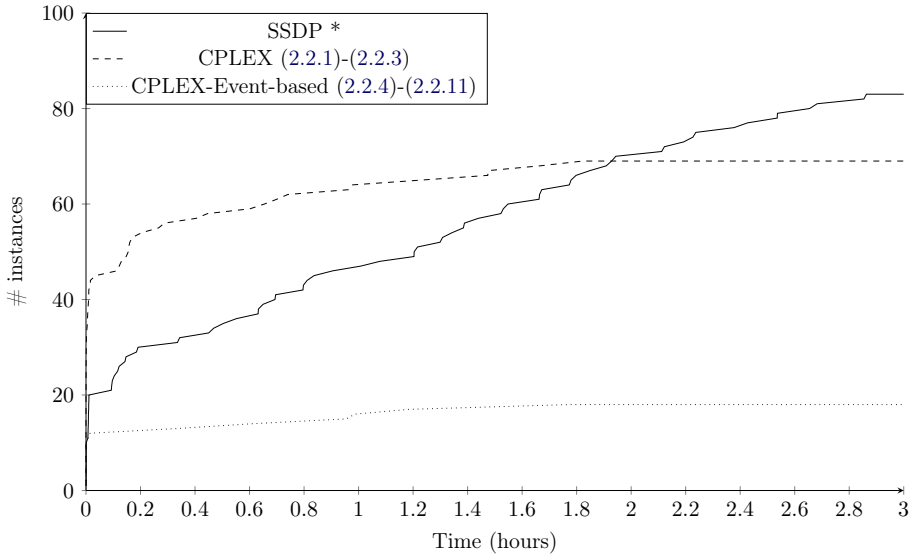


Figure 2.2.12: Number of instances solved along time for our best algorithm (SSDP*) and an ILP solver solving model (2.2.1)-(2.2.3) (CPLEX) and model (2.2.4)-(2.2.11) (CPLEX-Event-based), using a single thread.

First, the straightforward use of the event-based model (2.2.4)-(2.2.11) is not a competitive approach. All instances but five were solved by at least one of the two other methods. In terms of number of instances solved after three hours, SSDP* shows the best performance. After three hours, SSDP* optimally solves 83 instances, which is 14 more than CPLEX. CPLEX is better than SSDP* for several instances, mostly instances numbered from 1 to 55. The solver is able to solve most of them in a few seconds, while SSDP* may need minutes or hours. That can be explained by the powerful procedures embedded in such solvers to deal with knapsack constraints (for example to derive cuts), as well as good generic heuristics. From instance 55 to 99 however, CPLEX is only able to solve 16 instances. This can be explained by the structure of the instances: each batch of ten consecutive instances has a similar structure, most notably the maximum number of items in a clique. This number increases with the index of the instances. It transpires from these experiments that linear programming based methods are highly sensitive to this parameter, which is not the case for SSDP*.

We now report the performances of CPLEX and our method in their multiple thread setting. Figure 2.2.13 reports the results with six threads for SSDP*, CPLEX and CPLEX-Event-based. Using more cores is beneficial for all methods,

although CPLEX-Event-based only solves two more instances within the time limit. After those three hours, SSDP* with six threads optimally solved 94 instances, 12 more than CPLEX. Our method is not six times faster, since only the Lagrangian problem solver is parallelized. The time needed to construct and update the graph representation of the dynamic program represents a large percentage of the total running time, and this part of the algorithm does not benefit from a multi-core architecture. Only two instances are solved by CPLEX and not by SSDP* (U69 and U78). Conversely, SSDP* is able to find 14 solutions when CPLEX does not reach convergence.

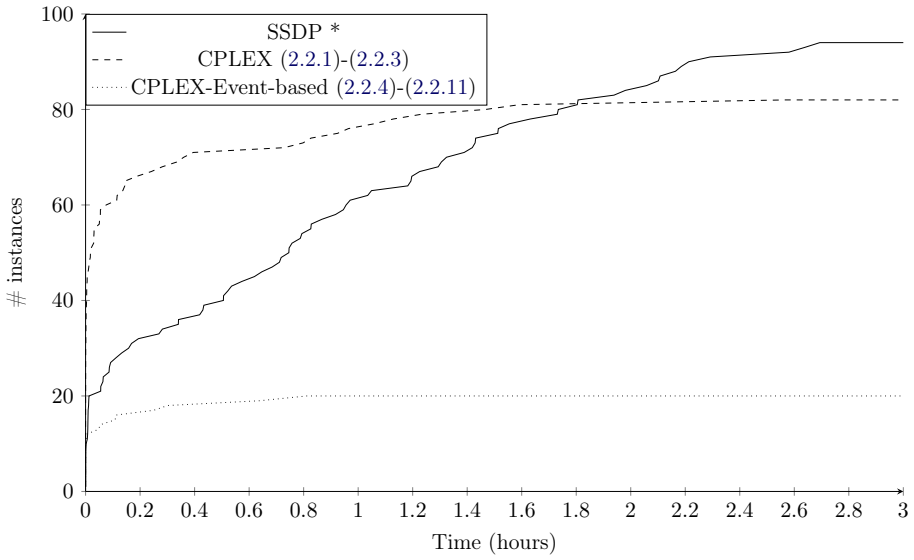


Figure 2.2.13: Number of instances solved along time for our best algorithm (SSDP*) using 6 threads and an ILP solver on model (2.2.1)-(2.2.3) (CPLEX) and model (2.2.4)-(2.2.11) (CPLEX-Event-based), using 6 threads.

2.2.4.5 Sensitivity of our method to the knapsack capacity

The size of our dynamic program depends on the knapsack capacity W. When this value increases, so does the time and memory needed to create the graph representation of the problem. Conversely, MIP solvers are generally less sensitive to the value of this parameter.

We run additional experiments on instances with larger knapsack capacities. We implemented the data generators of [Caprara *et al.* 2013], the same that were used to create the instances we use in the previous experiments. As mentioned above, the authors generated 20 different classes of instances ($I1$ to $I10$ and $U1$ to $U10$). We generated four new instances for each class: two with $W = 1000$ and two with $W = 10000$. Therefore, the instances we generated have the same structure as the instances generated by [Caprara *et al.* 2013], but with a larger knapsack capacity. For these experiments, Proposition 2.2.7 is not used within the SSDP algorithm,

Table 2.2.3: Sensitivity of our method to the knapsack capacity. In these experiments, Proposition 2.2.7 is not used within the SSDP algorithm.

Instance type	Capacity	#Instance status (over 20)			
		CPLEX only	SSDP only	solved by both	unsolved
I	1000	0	12	8	0
U	1000	5	0	7	8
I	10000	0	11	9	0
U	10000	14	0	0	6

because the memory requirements of the additional data is too large.

Our results are reported in Table 2.2.3. For each instance type (I or U), and each knapsack capacity (1000 and 10000), we report the number of instances (over the 20 generated) that are solved by CPLEX only, SSDP only, and both methods, within a three-hour time limit. We also report the number of unsolved instances.

According to these results, SSDP clearly suffers when there is a larger knapsack capacity and from the deactivation of Proposition 2.2.7, as expected. On average, SSDP remains competitive even with larger capacities (47 instances solved against 43), in particular for instances in type I. Although these instances were solved in a handful of seconds by all methods for $W = 100$, CPLEX was only able to solve 8 instances over the 20 generated with $W = 1000$, and 9 instances when $W = 10000$. It is able to find a good solution quickly, but is not able to close the gap after three hours of computation time. SSDP is able to solve all large type I instances, although it comes with a larger computational cost (respectively 284s and 596s on average for $W = 1000$ and $W = 10000$). Conversely, CPLEX is more effective for instances of type U. For $W = 10000$, it is able to solve 14 instances, while SSDP is not able to solve any instance of this set, because the memory needed to store the graph representation of the first dynamic program is too large.

2.3 Other contributions in State-Space Relaxation and deterministic optimization

SSR for scheduling In [Detienne *et al.* 2012], study we a single machine scheduling problem whose objective is to minimize a regular step total cost function. Lower and upper bounds, obtained from linear and Lagrangian relaxations of different Integer Linear Programming formulations, are compared. A dedicated exact approach is presented, based on a Lagrangian relaxation. It consists of finding a Constrained Shortest Path in a specific graph designed to embed a dominance property. Filtering rules are developed for this approach in order to reduce the size of the graph, and the problem is solved by successively removing infeasible paths from the graph. Numerical experiments are conducted to evaluate the lower and upper bounds. Moreover, the exact approach is compared with a standard solver and a naive branch-and-bound algorithm.

MILP approach for scheduling problems In [Detienne 2014], we investigate scheduling problems that occur when the weighted number of late jobs that are subject to deterministic machine availability constraints have to be minimized. These problems can be modeled as a more general job selection problem. Cases with resumable, non-resumable, and semi-resumable jobs as well as cases without availability constraints are investigated. The proposed efficient mixed integer linear programming approach includes possible improvements to the model, notably specialized lifted knapsack cover cuts. The method proves to be competitive compared with existing dedicated methods: numerical experiments on randomly generated instances show that all 350-job instances of the test bed are closed for the well-known problem $1|r_i|\sum w_iU_i$. For all investigated problem types, 98.4% of 500-job instances can be solved to optimality within one hour.

SSR and MILP for a railway problem The subject of [Benkirane *et al.* 2020] is an integrated optimization approach for timetabling and rolling stock rotation planning in the context of passenger railway traffic. Given a set of possible passenger trips, service requirement constraints, and a fleet of multiple heterogeneous self-powered railcars, our method aims at producing a timetable and solving the rolling stock problem in such a way that the use of railcars and the operational costs are minimized. To solve this hard optimization problem, we design a mixed-integer linear programming model based on network-flow in an hypergraph. We use this models to handle effectively constraints related to coupling and decoupling railcars. To reduce the size of the model, we use an aggregation and disaggregation technique combined with reduced-cost filtering. We present computational experiments based on several French regional railway traffic case studies to show that our method scales successfully to real-life problems.

Lagrangian relaxation-based heuristics for lot-sizing In [Absi *et al.* 2013], we deal with the multi-item capacitated lot-sizing problem with setup times and lost sales. Because of lost sales, demands can be partially or totally lost. To find a good lower bound, we use a Lagrangian relaxation of the capacity constraints, when single-item uncapacitated lot-sizing problems with lost sales have to be solved. Each subproblem is solved using an adaptation of the dynamic programming algorithm of [Aksen *et al.* 2003]. To find feasible solutions, we propose a non-myopic heuristic based on a probing strategy and a refining procedure. We also propose a metaheuristic based on the adaptive large neighborhood search principle to improve solutions. Some computational experiments showing the effectiveness and limitation of each approach are presented.

Chapter 3

Decomposition approaches for uncertain optimization problems

Contents

This chapter reports contributions for solving optimization problems with uncertain parameters. Section 3.1 presents a study made in collaboration with the French power producer EDF, within the PhD thesis of Rodolphe Griset. It is about the planning of nuclear plant outages, modeled as a two-stage stochastic problem. Dantzig-Wolfe and Benders decompositions are jointly used to produce high quality solutions to real size instances. In Section 3.2, we describe decomposition approaches for a class of two-stage robust problems with integer recourse, which has been developed in collaboration with Ayşe Nur Arslan during her postdoctoral residency. Reformulations based on the convexification of the recourse feasible set are proposed, leading to an effective exact solution method. Section 3.3 summarizes other contributions addressing uncertain optimization problems.

3.1 Double decomposition for the outage planing problem

This section is based on the journal paper [Griset *et al.* 2021]. We are interested in the nuclear outage planning problem at the French power producer EDF. This

problem is quite challenging given the specific operating constraints of nuclear units, the stochasticity of both the demand and non-nuclear units availability, and the scale of the instances. To tackle these difficulties we use a combined decomposition approach. The operating constraints of the nuclear units are built into a Dantzig-Wolfe pricing subproblem whose solutions define the columns of a demand covering formulation. The scenarios of demand and non-nuclear units availability are handled in a Benders decomposition. Our approach is shown to scale up to the real-life instances of the French nuclear fleet.

Section 3.1.1 presents the context of the study and some literature review. In Section 3.1.2, the nuclear outage planning problem is described. In Section 3.1.3, variants of the proposed extended formulation are presented. Section 3.1.4 gives the details of the Dantzig-Wolfe and Benders reformulations dedicated to handle real size stochastic instances through a row-and-column generation technique. In Section 3.1.5, the computational results attest the effectiveness of the proposed approach on real size (deterministic and stochastic) EDF instances of the problem.

3.1.1 Introduction

Nuclear production is a major source of electricity production for EDF, which operates 58 reactors at 19 locations in France. In the following, reactors are referred to as *nuclear units* and locations as *power plants*. Each nuclear unit must undergo a periodic outage to perform maintenance tasks and to reload nuclear fuel, thus leading to a complex industrial process. In particular, each outage of a nuclear unit must be scheduled in advance to allow for coordination of EDF's personnel and subcontractors. Furthermore, a nuclear outage induces an expensive substitute production by other means, e.g., gas-fired units or purchases on electricity market, to fulfill the electricity demand. For these reasons, scheduling nuclear outages is of major economic importance for EDF.

In this study, we consider a fixed number of outages to be scheduled within a given time horizon. The electricity demand is discretized over the time horizon and includes potential sales in the electricity market. At each time period, demand must be met by available generation units or market purchases. Nuclear outages are subject to *scheduling constraints* caused by limited resource availability as unit refueling and maintenance operations share the same resources in terms of personnel and equipment. Nuclear units must account for various *operational constraints*, which also impact nuclear outages, namely upper-bound on remaining fuel levels at outage starts, limiting operation at intermediate power, non-linear decreasing production profiles once the fuel level falls below a given threshold. As the time horizon is large, the data is subject to uncertainty in particular the demand, prices and capacities of exchanges on the market, but also the availability of non-nuclear units. The Nuclear Outage Planning Problem (NOPP) is to find a minimum cost plan for the nuclear refueling outages satisfying both scheduling and operational constraints to meet the demand in an uncertain future. The uncertainty is modeled by a set of scenarios in a stochastic setting. The NOPP can be formulated as a two-

stage decision problem. The first-stage decisions are nuclear outage dates, which have to be fixed before knowing the future; the second-stage decisions correspond to the production plan once uncertainty is revealed.

In the 90's, different approaches were investigated on a deterministic variant of the problem. [Edwin & Curtius 1990] and [Mukerji *et al.* 1991] consider an Integer Linear Programming (ILP) approach in which nuclear decreasing profile constraints are relaxed. Furthermore both studies limit their tests to one nuclear power plant with at most six nuclear units and a one-year horizon.

[Fourcade *et al.* 1997] report a high increase in solution-time when attempting to solve a compact ILP formulation on instances involving several power plants over a three-year horizon, while enforcing binding scheduling constraints between outages from units of different power plants. To reduce the solution time, they apply Lagrangian relaxation to the demand constraint. The corresponding decomposition scheme is solved through Uzawa's sub-gradient algorithm [Arrow *et al.* 1958]. Each sub-problem, corresponding to a nuclear power plant, is solved through a Branch & Bound algorithm which has been enhanced with a clique cut generation of local power plant scheduling constraints. However, this approach presents a computational limitation related to the use of Uzawa's algorithm. It also needs a re-dispatching phase to satisfy the demand, which could not ensure that solutions remain feasible.

In 2010, EDF submitted the stochastic variant of the problem to the academic community through the ROADEF Challenge [Porcheron *et al.* 2010]. In the proposed problem specification, the amount of nuclear fuel to be reloaded was a continuous decision, the time period was from four to six hours and there were up to 500 stochastic scenarios. Moreover, the solution time was limited to one hour.Given these characteristics, the best results were obtained by (meta-)heuristic approaches, thus reinforcing EDF's choice to use such methods in the current operational solution to the NOPP . However, most of the teams involved in the challenge took advantage of the natural two-stage structure of the problem by embedding an MIP or LP formulation in some specific phase of their solution scheme. To deal with the various scheduling constraints, [Jost & Savourey 2013] propose an ILP formulation to fix outages dates, whereas [Brandt 2010, Godskesen *et al.* 2013, Gavranović & Buljubašić 2013] apply constraint programming methods. Once first-stage decisions are fixed it is possible to solve the production dispatching problem through a Benders' like reformulation suitable for row generation. In [Lusby *et al.* 2010], such an approach is used, where specific constraints of the nuclear units are relaxed in a first phase before a so called reparation phase is performed. [Rozenknop *et al.* 2013] propose a dedicated state-space graph embedding nuclear operational constraints. A path in the graph corresponds to a feasible production plan with respect to the fuel level and the outage time-window constraints. The set of feasible plans is generated dynamically using a column generation scheme. The principle is that each plan prescribes fixed production levels for each week. The authors report computational issues as a lot of plans have to be generated, and generating each plan is computationally demanding.

Some characteristics did change over time in the definition of the problem since the ROADEF challenge. The amount of fuel to be reloaded was assumed to be variable. This assumption is no longer currently valid as the amount of fuel is fixed in the operational data. Hence the approaches presented in this article account for a fixed amount of fuel, even though they can account for a variable amount. Given the economic value at stake, a solution with guaranteed quality is of particular interest for EDF. To this end, some characteristics and specifications can be slighted amended is set back to a week and the solution time is extended to eight hours. These amendments make it possible to use approaches based on an exact solution procedure, in particular those combining reformulations and decompositions.

The principal focus of the current study is the stochastic aspect of the problem. A preliminary step is to deal with a single scenario in a deterministic setting. The idea is to find a suitable mathematical programming formulation to solve the corresponding NOPP instances at optimum. Then the number of scenarios considered is increased in a stochastic setting. The goal is then to solve NOPP instances with a quality guarantee whenever an exact solution is non-achievable. From an operational point of view the interest is to capture the impact of a given stochastic representation in the guaranteed solution quality.

Aside from ROADEF Challenge, a two-stage extended formulation, proposed in [Joncour 2011], involves a state-space graph in the first stage where an arc corresponds to either a production or an outage period. Our present work is derived from the latter formulation. Such an approach is computationally attractive because it involves network flow subsystems, thus leading to tight formulations. However, the size of the resulting model seriously impairs the scalability of the approach when facing real size instances. Our contribution is to propose an efficient, sparse, extended formulation for real size instances. When the number of scenarios increases, we resort to a double decomposition scheme to solve large-scale instances of the problem. Dantzig-Wolfe decomposition is used to manage the large-scale instances at first-stage and Benders' decomposition to exploit the independence of the sub-problems associated with different scenarios at second stage. We also study the effect of introducing second-stage variables representing a surrogate measure of the first-stage decisions. The problem is finally solved using a dedicated row-and-column generation algorithm. Our numerical study shows that, in the deterministic setting, real size instances can be solved to optimality directly using a commercial MIP solver with the proposed extended formulation. In the stochastic setting with up to 32 aggregated scenarios built from a 484 scenario database, the proposed row-and-column generation based method provides good quality solutions.

3.1.2 Problem description

The objective of the NOPP variant considered in this paper is to find a plan for the nuclear refueling outages that minimizes, on a given time horizon, the expectation cost of non-nuclear units on several scenarios of demand, market prices and capacities, and fossil unit availability, while satisfying the demand and both the scheduling

and operational constraints.

Stochastic aspects In its most general form, the NOPP is a multi-stage stochastic problem. The plan of nuclear refueling outages is re-optimized every month, given that the outages scheduled to occur in less than a year are almost fixed. The demand, availability of nuclear and non-nuclear units and costs of non-nuclear units are known weeks and sometimes only days in advance, and unit production is re-optimized up to 30 minutes before producing.

In this article, we assume that the availability of nuclear units is deterministic, while the demand and non-nuclear units are stochastic and subject to uncertainty. The uncertainty is represented through a set of aggregated scenarios built using an EDF library with a database of 484 original scenarios (see Section 3.1.5). These assumptions allow for modeling the NOPP as a two-stage problem. In the first stage the outage dates are the here-and-now decisions taken before uncertainty is revealed, while in the second stage, the aim is to have a feasible production plan – i.e., the demand is met and the fuel level constraints of nuclear units are satisfied – which minimizes the leasing cost of non-nuclear units.

In each scenario, fictitious units have been included to guarantee that production can meet the demand. A *failure unit* with a very high cost and infinite power capacity is added to the non-nuclear fleet to prevent any under-capacity of production with respect to the demand. Thus a solution with insufficient nuclear production will be feasible but expensive. Similarly a *load shedding unit* with cost matching expected selling prices on electricity markets and negative production is added to the non-nuclear fleet to prevent any overcapacity with respect to the demand.

Nuclear unit production constraints A nuclear unit operates in a cyclic but non-periodic way. Every *cycle* starts with a production campaign which might be divided into two phases. During the first phase, the unit may be operated at intermediate power in order to save fuel for later on. The total fuel saving, called *modulation*, is limited for each cycle. If the fuel level reaches a given threshold, called *BO* in the following, the unit operation enters its second phase where production has to follow a non-linear *decreasing profile* starting from full power and decreasing each week from about 3% until the next outage starts. Figure 3.1.1 gives an illustration of a nuclear cycle.

A cycle ends with an outage during which maintenance is performed and a given amount of nuclear fuel is reloaded. In practice, a fraction (a third or quarter) of the assemblies in the core of the reactor are replaced by fresh ones. Deciding to start an outage period during the first phase, i.e. with a fuel level greater than the BO level, is called *a stop by anticipation*; it can occur only provided the fuel level is below a given fuel level called *maximum anticipation*. Conversely, a stop during the second phase is called a *stop in prolongation*. Note that, from the fuel perspective, a stop in prolongation is economically more interesting than a stop by anticipation,

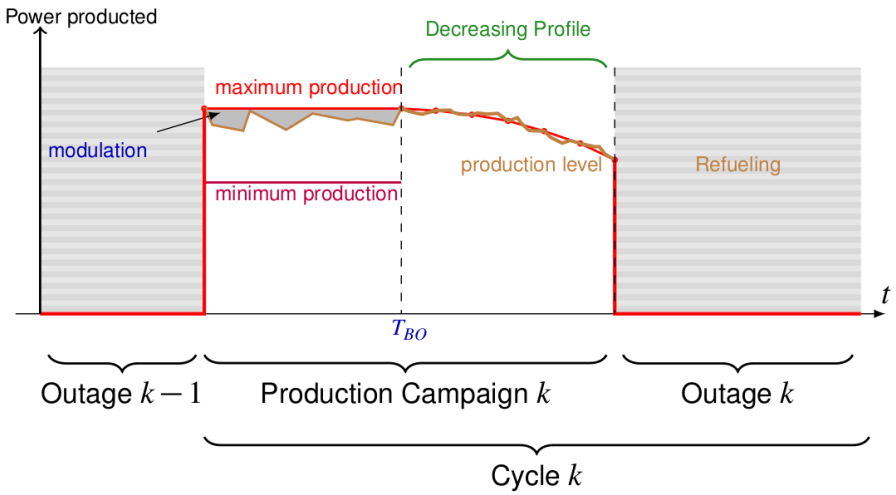


Figure 3.1.1: Illustration of the specific constraints related to nuclear units.

as a prolongation leads to a better use of the removed fuel assemblies. However, the unit must be stopped at the latest once the fuel level has reached a threshold called *maximum prolongation*.

Outage scheduling constraints Nuclear units refueling and maintenance operations are complex industrial tasks that require personnel with specific and possibly rare skills, and dedicated equipment. Hence, the number of outages sharing a specific resource is limited at any time. There is a large variety of such resource constraints, but the most widely used ones can be classified in one of the following two categories:

- Local power plant constraints: nuclear units are located on 19 power plants. All but one power plant comprise either two or four units. Each of such power plants has the required personnel to perform at most one outage at a time. In addition, the outages of any two units at the same power plant should be separated by several weeks. As for the six unit power plant, the outages of at most two units can be performed at a time.

- Global power plant constraints: when an outage duration is less than a month, only nuclear fuel can be reloaded. When an outage duration is more than a month, maintenance tasks are also performed and a very specific equipment is required. Hence, the number of simultaneous outages with a duration more than a month is limited for the whole nuclear fleet.

Nomenclature and notations

In this article, sets, vectors and matrices are indicated in boldface, whereas a scalar

is in non-boldface.

Indices

$t \in \mathbf{T}$ index of weeks

$j \in \mathbf{J}$ index of non-nuclear plants

$i \in \mathbf{I}$ index of nuclear units

$k \in \mathbf{K_i}$ cycle index of nuclear unit i

$c \in \mathbf{R}$ scheduling constraint index

$\omega \in \mathbf{\Omega}$ index of scenarios

Input Data

D_t^ω total demand for week t in scenario ω

For each nuclear unit i:

S_i initial fuel level

$\overline{P}_{it}$ maximum production during week t

For each nuclear unit i and each cycle k:

A_{ik} maximum fuel level at the beginning of the outage

M_{ik} maximum modulation during the production period

DP_{ik} minimum fuel level at the beginning of the outage

ED_{ik} early date of the outage

DD_{ik} due date of the outage

For each non-nuclear unit j:

$\overline{P}_{jt}^\omega$ maximum production during week t in scenario ω

C_{jt}^ω leasing cost during week t in scenario ω

For each scheduling constraint r:

N_{rt} maximum number of resources available during week t

$\boldsymbol{I}_r$ time interval (in weeks) during which constraint r is active

$\boldsymbol{O}_r$ set of outages defined by pair (i,k) involved in constraint r

$L_r(i,k)$ delay (in weeks) after the beginning of outage $(i,k) \in \boldsymbol{O}_r$ for the consumption of one resource for each nuclear unit i and cycle k

$D_r(i,k)$ duration of resource consumption for outage $(i,k) \in \boldsymbol{O}_r$

Energy is given in *Equivalent Full Power*, denoted by *EFP*, corresponding to the energy produced in one week at full power for the corresponding nuclear unit.

3.1.3 Extended formulations

In this section, the nuclear constraints of each unit are captured in a dedicated graph, thus leading to a *network flow formulation*. Such a formulation involves additional variables w.r.t. original variables, e.g., outage dates, thus leading to a so called *extended formulation*. Indeed an extended formulation of the original polyhedron is a polyhedron whose projection onto the original variables is the original polyhedron.

3.1.3.1 Assumptions

Assumption 3.1.1. *Similarly to [Rozenknop et al. 2013] and [Joncour 2011], we assume a discretization of the fuel levels that can be reached at the end of a production cycle.*

Assumption 3.1.2. *A production campaign with modulation greater than a week of production will contain a modulation corresponding to an integer number of weeks during which the unit is operated at minimum power instead of maximum power. It implies in particular that BO level will be reached at the end of a week, allowing a decreasing profile period to start at the beginning of the next week.*

Assumption 3.1.3. *A production campaign that involves a significant modulation, i.e more than one EFP, cannot be stopped before the fuel level reaches BO level.*

Assumption 3.1.4. *Inversely, a production campaign can be stopped by anticipation provided it involves no modulation, except the minimum required to end the cycle with the nearest discrete fuel level available. Note that the corresponding modulation must be lower than one EFP (Equivalent Full Power).*

Assumption 3.1.5. *The fuel level at the end of a given production period and the starting time of each decreasing profile are common to all scenarios. It implies in particular that the total energy saved by modulation is common for all scenarios as well.*

Assumption 3.1.1 is a mild assumption used to convert continuous fuel levels in discrete states decisions. Assumptions 3.1.2-3.1.4 are slightly stronger assumptions used to define production arcs corresponding to production decisions which translate into bounds on production variables in Section 3.1.3.3. Assumption 3.1.2 separates decreasing profile periods from production periods. Assumption 3.1.3 and Assumption 3.1.4 are standard from an economical point of view. The rationale behind is that using modulation before a stop with anticipation would increase the amount of fuel loss during an outage. Finally Assumption 3.1.5 is the strongest assumption used to separate first stage decisions, i.e., outages dates and fuel levels, from second stage production dispatching in each scenario. As for Assumption 3.1.1, it can be argued that matter, while production dispatching results from the former decisions. Note that the cost and feasibility w.r.t. the demand of a given scenario can be evaluated once production is dispatched. Assumption 3.1.5 is used to obtain an efficient formulation in Section 3.1.3.5 and in decomposition schemes presented in Section 3.1.4. Under these assumptions, we assume that there exists at least one feasible production plan for each nuclear unit.

3.1.3.2 Transition graph

Thanks to the previous assumptions, we can associate to each nuclear unit i a graph $\boldsymbol{G}_i = (\boldsymbol{V}_i, \boldsymbol{E}_i)$. Each path from the source to the sink of this graph corresponds to a plan satisfying the following constraints for unit i: time window for each outage, maximum fuel level for each refueling both minimum and maximum fuel level, and maximum modulation for each cycle.

Each vertex is uniquely associated with a flag f, a cycle index k, a fuel level index a (Assumption 3.1.1) and a week t, while each arc corresponds to either an outage or a production period. Vertices are partitioned in four classes:

$(f = \textbf{Begin}, k, a, t)$ called *Begin vertex* corresponds to the start of the k^{th} production period taking place at the beginning of week t, while a corresponds to the index of the fuel level at the end of the preceding production period.

$(f = \textbf{End}, k, a \geq 1, t)$ called *Anticipation vertex* corresponds to the end of the k^{th} production period taking place at the end of week $t-1$ with a positive fuel level corresponding to a weeks at full power.

$(f = \textbf{BO}, k, a = 0, t)$ called *BO vertex* corresponds to the case where the BO level is reached in the k^{th} cycle at the end of week $t-1$. An outage or a decreasing profile period may start at the beginning of week t.

$(f = \textbf{End}, k, a < 0, t)$ called *DP vertex* corresponds to the end of the k^{th} production period taking place at the beginning of week t following a decreasing profile of $-a$ weeks.

Note that the information defining vertices includes the exact fuel level of the unit in week t. The source node of $\boldsymbol{G}_i$ is a fictitious node whose data are adjusted in order to reflect the initial condition of the unit at the outset of the planning horizon.

An arc is defined by a starting week t_1 and an ending week $t_2 - 1$. Assumptions 3.1.3, 3.1.2 and 3.1.4 let us use only arcs in the following four categories:

Full-power arcs linking a *Begin vertex* to an *Anticipation vertex*. Each arc is defined only if t_2 is within the time window for the following outage and if the targeted fuel level is below the maximum anticipation of the current cycle.

Modulation arcs linking a *Begin vertex* to a *BO vertex*. Such arcs correspond to a fuel reload and a production leading to the *BO* level at the end of week $t_2 - 1$. It is defined if and only if the amount of modulation necessary to reach BO level at the end of week $t_2 - 1$ is below the maximum modulation value of the current cycle.

Decreasing profile arcs linking a *BO vertex* to a *DP vertex*. Such arcs correspond to the decreasing profile of the current cycle from week t_1 to $t_2 - 1$. Week t_2 has to be within the time window of the following outage.

Outage arcs linking a BO or End vertex to a Begin vertex relative to the next cycle. Such arcs represent an outage between weeks t_1 and $t_2 - 1$ with a fuel reload. Note that the length of the current cycle's outage should equal $t_2 - t_1 - 1$.

The sink of the graph is a fictitious vertex linked to all vertices going beyond the horizon t by a fifth kind of arc called *arc to sink*.

Let us consider an instance with a single unit as an illustrative example on the graph construction. For ease of presentation, the index of the unit is omitted. By convention, the initial cycle index and initial week are 0. The entries given in terms of EFP for the first cycle $k = 0$ are the following : $S = 9.8$, $A_0 = 2.5$, $M_0 = 3$, $DP_0 = 8$, while the early and due dates are $ED_0 = 7$ and $DD_0 = 11$. The first vertex with a positive fuel level is a Begin vertex. Hence, the source vertex s of the graph is with label (Begin,0,0,0). For full-power arcs, the unit must produce at full power at least for eight weeks for the fuel level initially at $S = 9.8$ to be less than $A_0 = 2.5$. The discretized final fuel levels correspond to integers in terms of EFP, hence every production period will at least contain a modulation of 0.2 EFP. At the beginning of week 8, the fuel level is 2 and the production period can end as $ED_0 \leq 8 \leq DD_0$. A full-power arc is added to the graph leading to vertex (End,0,2,8). Similarly, another full-power arc leads to vertex (End,0,1,9), which corresponds to produce during one additional week. No more full power arcs can be added. Reaching BO level through a modulation arc at the beginning of week 10 is possible with a modulation of 0.2 EFP between week 0 and 9, the corresponding vertex is (BO,0,0,10). With a modulation of 1.2 EFP between week 0 and 9, another vertex (BO,0,0,11) can be added. As the outage due date has been reached, no more vertices can be added, even though it would be possible w.r.t. M_0 to increase modulation. Decreasing profile periods correspond to keep producing below BO level, i.e., with a negative fuel level. The time window allows producing until the end of week 10, i.e., beginning of week 11. Hence a decreasing profile arc is created from vertex (BO,0,0,10) to a new vertex (End,0,-1,11). As the outage due date has been reached, no more vertices can be added, even though it would be possible w.r.t. DP_0 to keep producing along the decreasing profile. As for outage arcs, an outage can begin at each End and BO vertices. Figure 3.1.2 shows the resulting graph at the beginning of cycle 1.

In the remainder of the article, each arc is associated with an index e where $e = (u, v)$ represents the arc with origin vertex u and destination vertex v.

3.1.3.3 Additional precomputed data

In order to write our models, we introduce additional parameters computed from the input data.

First, we associate with each arc bounds that specify how energy production is constrained when it is chosen in a solution. Given an arc $e \in \boldsymbol{E}_i$, starting at the beginning of week $t_1(e)$ and ending at the end of the week $t_2(e) - 1$, we define $\bar{p}_e(t)$ (resp. $\underline{p}_e(t)$), $t \in \{t_1(e), \ldots, t_2(e) - 1\}$ such that $\sum_{t'=t_1}^{t} \bar{p}_e(t')$ (resp. $\sum_{t'=t_1}^{t} \underline{p}_e(t')$) is the maximum (resp. minimum) possible energy produced by plant i between weeks $t_1(e)$ and t, taking into account the type of arc e and its prescribed start and end fuel levels. For *full power* and *decreasing profile arcs*, $\bar{p}_e(t) = \underline{p}_e(t)$ for all t. Note that the use of those bounds allows modeling non-linear decreasing

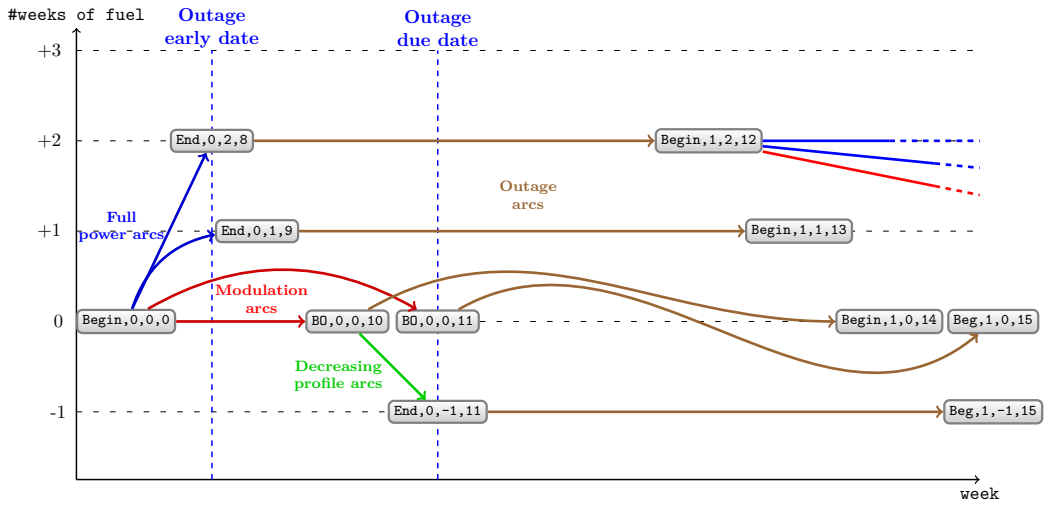


Figure 3.1.2: Graph relative to a single unit at the end of the first cycle.

profile constraint through linear constraints involving integer arc variables. For *outage arcs*, $\bar{p}_e(t) = \underline{p}_e(t) = 0$ for all t. For a *modulation arc*, $\bar{p}_e(t)$ (resp. $\underline{p}_e(t)$), $t \in \{t_1(e), \ldots, t_2(e)-1\}$, is computed by placing the maximum amount of production as soon as (resp. as late as) possible in the corresponding cycle (see Figure 3.1.3). Note that those bounds prescribe a precise cumulative energy produced over the time period spanned by arc e. By convention, we set $\bar{p}_e(t) = \underline{p}_e(t) = 0$ for all e, w such that $e, w \notin [t_1(e), t_2(e) - 1]$.

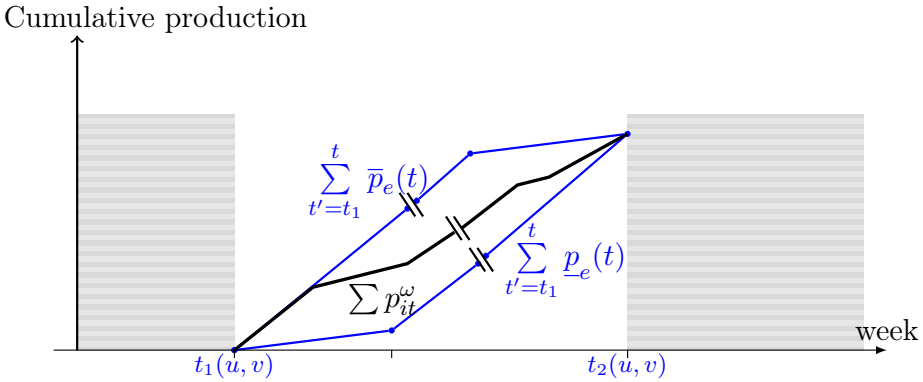


Figure 3.1.3: Bounds on cumulative energy produced for a given arc (u, v).

Second, we aim at writing a compact expression for the set $\boldsymbol{R}$ of various scheduling constraints. Each constraint $r \in \boldsymbol{R}$ involves a set $\boldsymbol{O}_r$ of outages each identified with plant i and the cycle k. When outage (i, k) starts at week $t \in \boldsymbol{I}_r$, one unit of resource r is used from week $t + L_r(i, k)$ to week $t + L_r(i, k) + D_r(i, k) - 1$. Based on those data, we derive the set of *outage arcs* that are involved in each constraint r at week t:

$\boldsymbol{O}_{rt} = \{e \text{ outage arc} : t_1(e) \leq t - L_r(i, k) \leq t_1(e) + D_r(i, k), e \in \boldsymbol{E}_i, (i, k) \in \boldsymbol{O}_r\}$.

3.1.3.4 Arc flow-based mixed integer programming formulation

We now introduce a natural formulation in the sense it is based on natural variables arising from the transition graph defined in Section 3.1.3.2 and the precomputed data in Section 3.1.3.3. This is a preliminary step before introducing additional variables leading to more efficient formulations.

For nuclear unit i and each arc e of graph $\boldsymbol{G}_i = (\boldsymbol{V}_i, \boldsymbol{E}_i)$, let δ_e be a 0-1 variable which takes on a value of 1 if the arc is used to define the production plan for nuclear unit i, 0 otherwise. We note respectively $s(\boldsymbol{G}_i)$ and $t(\boldsymbol{G}_i)$ the source and the sink of $\boldsymbol{G}_i$. For each scenario ω, let p^{ω}_{iw} be a continuous variable defining the energy produced by nuclear unit i during week w. Similarly let p^{ω}_{jw} be a continuous variable defining the energy produced by non-nuclear nuclear unit j for week w under scenario ω. An intermediary variable $\underline{q}_{it}$ (resp. $\overline{q}_{it}$) is introduced to set the lower bound (resp. upper bound) on the total energy produced by nuclear unit i up to time period t in any scenario ω, i.e., such energy is bound to lie in the prescribed envelope defined in Section 3.1.3.3. The resulting formulation for NOPP with $(\boldsymbol{\delta}, \boldsymbol{p})$ as mandatory variables and $(\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}})$ as intermediate variables is denoted by $F_{(\underline{\boldsymbol{q}},\overline{\boldsymbol{q}})}(\boldsymbol{\delta}, \boldsymbol{p})$ and is as follows.

$$\min \frac{1}{\boldsymbol{\Omega}} \sum_{j,t,\omega} C^{\omega}_{jt} p^{\omega}_{jt} \tag{3.1.1}$$

$$\sum_{(u,v)\in \boldsymbol{O}_{rt}} \delta_{(u,v)} \leq N_{rt} \qquad \forall r \in \boldsymbol{R}, t \in \boldsymbol{I}_r \tag{3.1.2}$$

$$\sum_{(u,v)\in \boldsymbol{E}_i} \delta_{(u,v)} - \sum_{(v,u)\in \boldsymbol{E}_i} \delta_{(v,u)} = 0 \quad \forall i, u \in \boldsymbol{V}_i \backslash \{s(\boldsymbol{G}_i), t(\boldsymbol{G}_i)\} \tag{3.1.3}$$

$$\sum_{(s(\boldsymbol{G}_i),v)\in \boldsymbol{E}_i} \delta_{(s(\boldsymbol{G}_i),v)} = 1 \qquad \forall i \tag{3.1.4}$$

$$\delta_{(u,v)} \in \{0,1\} \qquad \forall i, (u,v) \in \boldsymbol{E}_i \tag{3.1.5}$$

$$\underline{q}_{it} = \sum_{(u,v)\in \boldsymbol{E}_i} \underline{p}_{(u,v)}(t) \delta_{(u,v)} \qquad \forall i, t \tag{3.1.6}$$

$$\overline{q}_{it} = \sum_{(u,v)\in \boldsymbol{E}_i} \overline{p}_{(u,v)}(t) \delta_{(u,v)} \qquad \forall i, t \tag{3.1.7}$$

$$\sum_{t'=0}^{t} \underline{q}_{it'} \leq \sum_{t'=0}^{t} p^{\omega}_{it'} \leq \sum_{t'=0}^{t} \overline{q}_{it'} \qquad \forall i, t, \omega \tag{3.1.8}$$

$$\sum_{i} p^{\omega}_{it} + \sum_{j} p^{\omega}_{jt} = D^{\omega}_{t} \qquad \forall t, \omega \tag{3.1.9}$$

$$0 \leq p^{\omega}_{it} \leq \overline{P}_{it} \qquad \forall i, t, \omega \tag{3.1.10}$$

$$0 \leq p^{\omega}_{jt} \leq \overline{\mathrm{P}}^{\omega}_{jt} \qquad \forall j, t, \omega \tag{3.1.11}$$

Objective function (3.1.1) minimizes the average total production cost of non-nuclear units over all considered scenarios ω, (3.1.2) corresponds to the scheduling constraints with limited resource, (3.1.4) are flow constraints imposing that one feasible

plan should be assigned to each nuclear unit. Then (3.1.6) and (3.1.7) define bounds on energy production corresponding to arc flow decisions and enforced to nuclear production by (3.1.8) for each week w and each scenario ω. Finally (3.1.9) ensures that the demand is satisfied each week t under each scenario ω.

3.1.3.5 Dedicated arc flow-based MIP formulation

The arc flow formulation presented in Section 3.1.3.4 is quite straightforward bearing in mind that total energy to be produced by each nuclear unit is bound to lie in a given envelope as defined in Section 3.1.3.3. However, constraints (3.1.6)-(3.1.8) induce triangular dense structures linking $\boldsymbol{\delta}$ variables to nuclear production variables for each scenario ω. Such structure is likely to impair the performance of the formulation especially anticipating to solve large-scale NOPP instances.

In this section, the key idea is to take advantage more explicitly of the definition of arcs combined with Assumption 3.1.5, which states that the fuel level at the end of any production period is the same for all scenarios. Formally, for a given production arc $e \in \boldsymbol{E}_i$:

$$\sum_{t=t_1(e)}^{t_2(e)} \underline{p}_{it} = \sum_{t=t_1(e)}^{t_2(e)} \overline{p}_{it} = \sum_{t=t_1(e)}^{t_2(e)} p_{it}^{\omega} \quad ,\forall \omega \tag{3.1.12}$$

Let us introduce, for each nuclear unit i and week t, set $\boldsymbol{A}_{it} \subset \boldsymbol{E}_i$, the set of *active arcs* at time t, i.e., $\boldsymbol{A}_{it} = \{e \in \boldsymbol{E}_i | t_1(e) \leq t < t_2(e)\}$. Interested reader may check that the following proposition follows from the left equality in (3.1.12) combined with the definition of $\underline{\boldsymbol{p}}$ and $\overline{\boldsymbol{p}}$:

Proposition 3.1.1.

$$\sum_{t'=0}^{t} \sum_{e \in \boldsymbol{E}_i} \left(\overline{p}_e(t') - \underline{p}_e(t') \right) \delta_e = \sum_{e \in \boldsymbol{A}_{it}} \sum_{t'=t_1(e)}^{t} \left(\overline{p}_e(t') - \underline{p}_e(t') \right) \delta_e$$

Then we can rewrite equations (3.1.6)-(3.1.8) without q-variables for a given unit i, a time period t and a scenario ω:

$$\begin{aligned}
&\sum_{t'=0}^{t} \sum_{e \in \boldsymbol{E}_i} \underline{p}_e(t') \delta_e \leq \sum_{t'=0}^{t} p_{it'}^{\omega} \leq \sum_{t'=0}^{t} \sum_{e \in \boldsymbol{E}_i} \overline{p}_e(t') \delta_e \\
\Leftrightarrow &\sum_{t'=0}^{t} \sum_{e \in \boldsymbol{E}_i} \left(\overline{p}_e(t') - \underline{p}_e(t') \right) \delta_e \geq \sum_{t'=0}^{t} \left(\sum_{e \in \boldsymbol{E}_i} \overline{p}_e(t') \delta_e - p_{it'}^{\omega} \right) \geq 0 \\
\Leftrightarrow &\sum_{e \in \boldsymbol{A}_{it}} \sum_{t'=t_1(e)}^{t} \left(\overline{p}_e(t') - \underline{p}_e(t') \right) \delta_e \geq \sum_{t'=0}^{t} \left(\sum_{e \in \boldsymbol{E}_i} \overline{p}_e(t') \delta_e - p_{it'}^{\omega} \right) \geq 0
\end{aligned} \tag{3.1.13}$$

Note that only active arcs in $\boldsymbol{A}_{it}$ have a non-zero contribution in the left hand side of Equation (3.1.13). Moreover the contribution of the decreasing profile and outage arcs in this left hand side is zero; the same holds for full power arcs, except

in the case of arcs with marginal modulation for which the contribution stays close to zero (see Assumption 3.1.4).

For the right hand side of Equation (3.1.13) we need to use a difference equation in order to keep only contributions of active arcs. Let us introduce a new continuous variable s_{it}^{ω} defined as the difference between the upper bound on the total energy produced and the real production by nuclear unit i up to time period t in scenario ω. Then, we can represent the evolution of this variable by a difference equation involving active arcs $\boldsymbol{A}_{it}$ and production variable p_{it}^{ω}:

$$\begin{aligned} s_{it}^{\omega} &= \sum_{t'=0}^{t} \left(\sum_{e \in \boldsymbol{E}_i} \overline{p}_e(t)\delta_e - p_{it'}^{\omega} \right) \\ \Leftrightarrow s_{it}^{\omega} - s_{i,t-1}^{\omega} &= \sum_{e \in \boldsymbol{A}_{it}} \overline{p}_e(t)\delta_e - p_{it}^{\omega} \\ \Leftrightarrow s_{it}^{\omega} + p_{it}^{\omega} &= s_{i,t-1}^{\omega} + \sum_{e \in \boldsymbol{A}_{it}} \overline{p}_e(t)\delta_e \end{aligned} \tag{3.1.14}$$

Finally we can rewrite Equation (3.1.13) using variables s as:

$$\forall i, t, \omega \qquad \sum_{e \in \boldsymbol{A}_{it}} \sum_{t'=t_1(e)}^{t} \left(\underline{p}_e(t') - \overline{p}_e(t') \right) \delta_e \geq s_{it}^{\omega} \geq 0 \tag{3.1.15}$$

The following equivalent MIP is thus derived:

$$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p}) : \min \left\{ \frac{1}{\boldsymbol{\Omega}} \sum_{j,t,\omega} C_{jt}^{\omega} p_{jt}^{\omega} : (3.1.2) - (3.1.5), \right.$$
$$\left. (3.1.14) - (3.1.15), (3.1.9) - (3.1.11) \right\}$$

From preliminary experiments (see Section 3.1.5), formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ appears to outperform by far formulation $F_{(\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}})}(\boldsymbol{\delta}, \boldsymbol{p})$ introduced in Section 3.1.3.4. The rationale behind introducing formulation $F_{(\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}})}(\boldsymbol{\delta}, \boldsymbol{p})$ is twofold. First it helps deriving formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$, which is in comparison more involved. Second it provides a reference to assess the improved performance. For the rest of the article, formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ is retained and formulation $F_{(\underline{\boldsymbol{q}}, \overline{\boldsymbol{q}})}(\boldsymbol{\delta}, \boldsymbol{p})$ discarded.

3.1.3.6 Splitting first and second stages through capacity variables

In the vein of the reformulation for problems with a fixed technology matrix exposed in [Birge & Louveaux 2011], Chap. 3, Section 1, we project the first-stage variables $\boldsymbol{\delta}$ onto the capacity of production or modulation relative to the plant schedules corresponding to $\boldsymbol{\delta}$. Note that such a capacity is the only relevant information for the second stage problem. This technique is of special interest in the context of a double decomposition, where reformulations lead to an exponential (in terms of the size of input data) number of variables involved in an exponential number of constraints. The general solving process we design is based on the dynamic generation of the

model. That implies heavy computational burden when adding each constraint or column, since the number of new coefficients to set at this occasion is potentially huge. The step of the algorithm corresponding to setting those additional coefficients will be referred to as *projection*. Moreover, the overall number of non-zero coefficients in the MIP formulation is very large as well. We investigated two ways of projecting the first-stage variables $\boldsymbol{\delta}$ onto the second-stage constraints, leading to equivalent formulations but with different computational benefits.

The first option is to define first-stage variables $\overline{q}_{it}$, $i \in \mathbf{I}$, $t \in \mathbf{T}$, which are equal to the maximum possible total energy produced by i up to week t, restricted to the cycle at t, and m_{it} the difference between the upper and lower bounds of the production envelop and hence the upper bound on variable s^{ω}_{it}:

$$\sum_{e \in \boldsymbol{E}_i} \overline{p}_e(t).\delta_e = \overline{q}_{it} \qquad \forall i, t \tag{3.1.16}$$

$$\sum_{e \in \boldsymbol{A}_{it}} \sum_{t'=t_1(e)}^{t} \left(\overline{p}_e(t') - \underline{p}_e(t')\right).\delta_e = m_{it} \qquad \forall i, t \tag{3.1.17}$$

$$\overline{q}_{it} + s^{\omega}_{i,t-1} = s^{\omega}_{i,t} + p^{\omega}_{it} \qquad \forall i, t, \omega \tag{3.1.18}$$

$$s^{\omega}_{it} \leq m_{it} \qquad \forall i, t, \omega \tag{3.1.19}$$

The second option is to use cumulative capacity variables $\overline{cq}_{it}$ (resp. $\underline{cq}_{it}$), $i \in \mathbf{I}$, $t \in \mathbf{T}$, defined as the maximum (resp. minimum) total energy produced by unit i up to week t:

$$\sum_{e \in \boldsymbol{E}_i} \sum_{t'=t_1(e)}^{t} \underline{p}_e(t').\delta_e = \underline{cq}_{it} \qquad \forall i, t \tag{3.1.20}$$

$$\sum_{e \in \boldsymbol{E}_i} \sum_{t'=t_1(e)}^{t} \overline{p}_e(t').\delta_e = \overline{cq}_{it} \qquad \forall i, t \tag{3.1.21}$$

$$\overline{cq}_{it} - \overline{cq}_{i,t-1} + s^{\omega}_{i,t-1} = s^{\omega}_{i,t} + p^{\omega}_{it} \qquad \forall i, t, \omega \tag{3.1.22}$$

$$s^{\omega}_{it} \leq (\overline{cq}_{it} - \underline{cq}_{it}) \qquad \forall i, t, \omega \tag{3.1.23}$$

The corresponding resulting formulations are the following:

$$F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p}) : \min \left\{ \frac{1}{\boldsymbol{\Omega}} \sum_{j,t,\omega} C^{\omega}_{jt} p^{\omega}_{jt} : (3.1.2) - (3.1.5), \right.$$
$$\left. (3.1.16) - (3.1.18), (3.1.9) - (3.1.11) \right\}$$

$$F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p}) : \min \left\{ \frac{1}{\boldsymbol{\Omega}} \sum_{j,t,\omega} C^{\omega}_{jt} p^{\omega}_{jt} : (3.1.2) - (3.1.5), \right.$$
$$\left. (3.1.20) - (3.1.22), (3.1.9) - (3.1.11) \right\}$$

3.1.4 Solution approaches

This section describes the proposed solution algorithms for NOPP. Such algorithms are based on the mathematical programming formulations presented in Section 3.1.3.5 and 3.1.3.6. As the intent is to solve large-scale NOPP instances, a double decomposition approach is considered, thus leading to a so-called *reformulated problem*. The principle is to solve iteratively small subproblems, which prevents us from solving too large of a problem. To ease notation, the different formulations are cast into a generic linear matrix formulation. The idea is to make it possible to present the Dantzig-Wolfe and Benders decomposition schemes in a general setting suited for all formulations. The row-and-column generation algorithm optimizing the linear relaxation of the reformulated problem is then presented. Finally the way to get near-optimal feasible solutions for NOPP from this relaxation is explained.

3.1.4.1 Generic formulation

The proposed generic formulation, called F_{Gen}, is introduced to cast the mathematical program corresponding to all formulations in a general setting. Such generic formulation involves vectors to put together subsets of decision variables, and matrices to capture the structure of all formulations.

All formulations presented in Section 3.1.3.5 feature a planning for each nuclear plant as a common structure. Such planning is defined in constraints (3.1.3)-(3.1.5), which can be rewritten as (3.1.25)-(3.1.26) in F_{Gen}. More precisely matrix $\boldsymbol{\Delta}$ and vector $\boldsymbol{d}$ are used to rewrite the shortest path constraints (3.1.3)-(3.1.4). For a given scenario $\omega \in \boldsymbol{\Omega}$, nuclear and non-nuclear power production variables, $(p^{\omega}_{it})_{i\in\mathbf{I},t\in\mathbf{T}}$ and $(p^{\omega}_{jt})_{j\in\mathbf{J},t\in\mathbf{T}}$, respectively, are included into a single vector $\boldsymbol{p}^{\omega}$, so that demand constraints (3.1.9) and bounds on production (3.1.10)-(3.1.11) are cast as (3.1.30) in F_{Gen}. Cost vectors $\bar{\boldsymbol{C}}^{\omega}$, $\omega \in \boldsymbol{\Omega}$, showing in the objective (3.1.24), take value $\bar{C}^{\omega}_{it} = 0$ for $i \in \mathbf{I}$ and $t \in \mathbf{T}$, and $\bar{C}^{\omega}_{jt} = \frac{1}{|\boldsymbol{\Omega}|} C^{\omega}_{jt}$ for $j \in \mathbf{J}$ and $t \in \mathbf{T}$.

The proposed three formulations differ from one another in the way they link the plant plannings to the actual corresponding power production. We introduce a vector of abstract variables $\boldsymbol{\xi}$ in the sense they replace either variables $\boldsymbol{s}$ from formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ or $[\overline{\boldsymbol{q}}, \boldsymbol{s}]$ (resp. $[\underline{\boldsymbol{cq}}, \overline{\boldsymbol{cq}}, \boldsymbol{s}]$) from $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$) and their dimension changes accordingly. Matrices $\mathbf{A}_0$, $\boldsymbol{\Xi}_0$ and $\boldsymbol{\Theta}^{\omega}_0$ and vector $\boldsymbol{b}^{\omega}_0$, $\omega \in \boldsymbol{\Omega}$, are used to rewrite constraints (3.1.14)-(3.1.15) from $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ as constraints (3.1.27) in F_{Gen}. They are with zero dimension in the case of alternative formulations, namely $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$. Matrices $\mathbf{A}_1$ and $\boldsymbol{\Xi}_1$ and vector $\boldsymbol{b}_1$ are used in constraints (3.1.28) to rewrite constraints (3.1.2) relative to resource-constrained scheduling along with complementary constraints induced by abstract variables $\boldsymbol{\xi}$, namely constraints (3.1.16)-(3.1.17) (resp. (3.1.20)-(3.1.21)) in formulation $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$). Matrices $\boldsymbol{\Xi}_2$ and $\boldsymbol{\Theta}^{\omega}_2$ and vector $\boldsymbol{b}^{\omega}_2$, $\omega \in \boldsymbol{Omega}$ are used in constraints (3.1.29) to rewrite constraints (3.1.18)-(3.1.19) (resp. (3.1.22)-(3.1.23)) linking power production $\boldsymbol{p}^{\omega}$ to abstract variables $\boldsymbol{\xi}$ in formulation $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$). They are with zero dimension in the case of formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$. Note that generic constraints (3.1.28) involve

only first-stage variables, while (3.1.27) and (3.1.29) involve first- and second-stage variables. The correspondence between the abstract variables and the generic constraints in F_{Gen} and all formulations is summarized in Table 3.1.1.

$$F_{Gen}: \min \quad \sum_{\omega} \bar{\boldsymbol{C}}^{\omega\top} \boldsymbol{p}^{\omega} \tag{3.1.24}$$

$$s.t. \quad \boldsymbol{\Delta\delta} = \boldsymbol{d} \tag{3.1.25}$$

$$\boldsymbol{\delta} \in \{0,1\} \tag{3.1.26}$$

$$\mathbf{A}_0\boldsymbol{\delta} + \boldsymbol{\Xi}_0\boldsymbol{\xi} + \boldsymbol{\Theta}_0^{\omega}\boldsymbol{p}^{\omega} \geq \boldsymbol{b}_0^{\omega} \quad \forall\omega \tag{3.1.27}$$

$$\mathbf{A}_1\boldsymbol{\delta} + \boldsymbol{\Xi}_1\boldsymbol{\xi} \geq \boldsymbol{b}_1 \tag{3.1.28}$$

$$\boldsymbol{\Xi}_2\boldsymbol{\xi} + \boldsymbol{\Theta}_2^{\omega}\boldsymbol{p}^{\omega} \geq \boldsymbol{b}_2^{\omega} \quad \forall\omega \tag{3.1.29}$$

$$\mathbf{P}^{\omega}\boldsymbol{p}^{\omega} = \boldsymbol{D}^{\omega} \quad \forall\omega \tag{3.1.30}$$

$$\boldsymbol{p}^{\omega} \geq \mathbf{0} \quad \forall\omega \tag{3.1.31}$$

$$\boldsymbol{\xi} \geq \mathbf{0} \tag{3.1.32}$$

F_{Gen}	$F(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$	$F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$	$F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$
$\boldsymbol{\xi}$	$\boldsymbol{s}$	$[\overline{\boldsymbol{q}},\boldsymbol{m}]$	$[\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}},\boldsymbol{s}]$
(3.1.27)	(3.1.14), (3.1.15)	-	-
(3.1.28)	(3.1.2)	(3.1.2),(3.1.16),(3.1.17)	(3.1.2), (3.1.20),(3.1.21)
(3.1.29)	-	(3.1.18),(3.1.19)	(3.1.22),(3.1.23)

Table 3.1.1: Correspondence between the generic formulation F_{Gen} and all proposed formulations $F(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$, $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$ and $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$. Mark "-" indicates formulations for which the generic constraint (3.1.27) (resp. (3.1.29)) vanishes, i.e., $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$ and $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$ (resp. $F(\boldsymbol{\delta},\boldsymbol{s},\boldsymbol{p})$).

3.1.4.2 Dantzig-Wolfe reformulation

Constraints (3.1.25)-(3.1.28) present a classical structure which is suitable for column generation. As constraints (3.1.25)-(3.1.26) are independent for each power plant i, they define a subproblem with binding constraints (3.1.27)-(3.1.28).

Let $\mathcal{G}_i$ be the set of feasible paths satisfying constraints (3.1.25) for nuclear unit i. Let λ_{ig} be the decision variable associated with choosing path $g \in \mathcal{G}_i$ for nuclear unit i and $\boldsymbol{\Lambda}^g \in \{0,1\}^{|\boldsymbol{E}_i|}$ the binary vector such that $\Lambda_e^g = 1$ if and only if arc $e \in \boldsymbol{E}_i$ is part of path g. For $i \in \mathbf{I}$ and $e \in \boldsymbol{E}_i$, we can now write $\delta_e = \sum_{g\in\mathcal{G}_i} \Lambda_e^g \lambda_{ig}$, *i.e.*, in matrix form $\boldsymbol{\delta} = \boldsymbol{\Lambda\lambda}$. Moreover, let $\mathbf{H} \in \{0,1\}^{|\mathbf{I}|\times\sum_{\mathbf{i}}|\mathcal{G}_{\mathbf{i}}|}$, such that $H_i^g = 1$ if

and only if $g \in \mathcal{G}_i$. This leads to the following Dantzig-Wolfe reformulation of F_{Gen}:

$$F_{Gen}^{DW} : \min \quad \sum_{\omega} \bar{\boldsymbol{C}}^{\omega\top} \boldsymbol{p}^{\omega} \tag{3.1.33}$$

$$s.t. \quad \mathbf{H}\boldsymbol{\lambda} = \mathbf{1} \tag{3.1.34}$$

$$\mathbf{A}_0\boldsymbol{\Lambda}\boldsymbol{\lambda} + \boldsymbol{\Xi}_0\boldsymbol{\xi} + \boldsymbol{\Theta}_0^{\omega}\boldsymbol{p}^{\omega} \geq \boldsymbol{b}_0^{\omega} \quad \forall \omega \tag{3.1.35}$$

$$\mathbf{A}_1\boldsymbol{\Lambda}\boldsymbol{\lambda} + \boldsymbol{\Xi}_1\boldsymbol{\xi} \geq \boldsymbol{b}_1 \tag{3.1.36}$$

$$(3.1.29), (3.1.30), (3.1.31), (3.1.32)$$

$$\boldsymbol{\lambda} \in \{0,1\}^{\sum_i |\mathcal{G}_i|} \tag{3.1.37}$$

Exponentially-many $\boldsymbol{\lambda}$-variables are involved in constraints (3.1.35), which are replicated for each scenario ω. Recall constraints (3.1.35) is a reformulation of constraints (3.1.27) actually used only in formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$. Interestingly the use of additional variables in formulations $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ allows for elimination of $\boldsymbol{\lambda}$-variables from second-stage constraints. The benefit is even clearer when the second-stage problem is reformulated with an exponential number of constraints, as shown in the next section. The Dantzig-Wolfe reformulation of $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ or $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$) is denoted by $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F^{DW}_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ or $F^{DW}_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$).

3.1.4.3 Benders reformulation

We propose to cope with the second-stage part of the problem using Benders reformulation [Benders 1962]. To avoid dealing with both optimality and feasibility cuts, we first move the second-stage objective value into constraints. Hence while only feasibility cuts are written, some of them can be interpreted as optimality cuts. The feasibility cuts are derived from an appropriate feasibility subproblem (see *e.g.* [Slyke & Wets 1969]). We use the *multi-cut* approach, which is to deal with feasibility and optimality conditions for each scenario independently.

Given the first-stage solution $(\boldsymbol{\lambda}, \boldsymbol{\xi})$, let us introduce the recourse function $R^{\omega}(\boldsymbol{\lambda}, \boldsymbol{\xi})$, $\omega \in \boldsymbol{\Omega}$, equal to the optimal cost of the second-stage solution in scenario ω if one exists, or equal to ∞ if the second stage problem is infeasible. We also use new decision variables $\eta_{\omega} \in \mathbb{R}$, $\omega \in \boldsymbol{\Omega}$, equal to the value of the recourse function at optimality. Formulation F_{Gen}^{DW} now takes the form of the following mathematical program, where all second-stage constraints and costs are implicitly embedded in piecewise linear convex functions R^{ω} :

$$\min \left\{ \sum_{\omega \in \boldsymbol{\Omega}} \eta_{\omega} : \eta_{\omega} \geq R^{\omega}(\boldsymbol{\lambda}, \boldsymbol{\xi}) \ \forall \omega \in \boldsymbol{\Omega}, \boldsymbol{\eta} \in \mathbb{R}^{|\boldsymbol{\Omega}|}, (3.1.32), (3.1.34), (3.1.36), (3.1.37) \right\} \tag{3.1.38}$$

A feasibility question arises that needs to be answered. A given first-stage solution $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega)$ that satisfies (3.1.32), (3.1.34), (3.1.36), (3.1.37) might not be feasible for (3.1.38) if it induces an unavoidable over-production for at least one period in one scenario, or if the estimated second-stage cost $\bar{\eta}^\omega$ is lower than the actual cost $R^\omega(\boldsymbol{\lambda}, \boldsymbol{\xi})$. Formally solution $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega)$ is feasible for scenario ω if and only if the optimum of the following linear program is zero.

$$
\begin{aligned}
f^\omega(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega) = \min \quad & \mathbf{1}^\top [\tau^\omega_{obj}, \boldsymbol{\tau}^\omega_0, \boldsymbol{\tau}^\omega_2] && && (3.1.39)\\
s.t. \quad & \mathbf{P}^\omega \boldsymbol{p}^\omega && = \boldsymbol{D}^\omega && (3.1.40)\\
& \sum_\omega \bar{\boldsymbol{C}}^{\omega\top} \boldsymbol{p}^\omega - \tau^\omega_{obj} && \leq \bar{\eta}^\omega && (3.1.41)\\
& \boldsymbol{\Theta}^\omega_0 \boldsymbol{p}^\omega + \boldsymbol{\tau}^\omega_0 && \geq \boldsymbol{b}^\omega_0 - \mathbf{A}_0 \boldsymbol{\Lambda} \bar{\boldsymbol{\lambda}} - \boldsymbol{\Xi}_0 \bar{\boldsymbol{\xi}} && (3.1.42)\\
& \boldsymbol{\Theta}^\omega_2 \boldsymbol{p}^\omega + \boldsymbol{\tau}^\omega_2 && \geq \boldsymbol{b}^\omega_2 - \boldsymbol{\Xi}_2 \bar{\boldsymbol{\xi}} && (3.1.43)\\
& \boldsymbol{p}^\omega \geq \mathbf{0} && && (3.1.44)
\end{aligned}
$$

In this program, artificial variables τ^ω_{obj}, $\boldsymbol{\tau}^\omega_0$ and $\boldsymbol{\tau}^\omega_2$ are introduced to account for the violations of the second-stage constraints. Then the latter program mimics phase one of the simplex method where artificial variables required to be zero are allowed to be non negative. Taken apart, constraints (3.1.40) can always be satisfied, possibly resorting to exchanges on the spot market. That is why no artificial variables are needed for them.

Since LP (3.1.39)-(3.1.44) is feasible and bounded, one can use the strong duality theorem in linear programming to express $f^\omega(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega)$ as the optimal value of its dual program. Associating vectors of dual variables $\boldsymbol{\nu}^\omega$, μ^ω, $\boldsymbol{\rho}^\omega_0$ and $\boldsymbol{\rho}^\omega_2$ to constraints (3.1.40), (3.1.41), (3.1.42) and (3.1.43), respectively, the dual LP is as follows.

$$
f^\omega(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega) = \max \left\{ G^\omega_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega}(\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) : (\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) \in \mathcal{D}^\omega \right\} \tag{3.1.45}
$$

where $G^\omega_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega}$ is the objective function

$$
G^\omega_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega}(\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) = \boldsymbol{D}^{\omega\top} \boldsymbol{\nu}^\omega + \bar{\eta}^\omega \mu^\omega + \left[\boldsymbol{b}^\omega_0 - \mathbf{A}_0 \boldsymbol{\Lambda} \bar{\boldsymbol{\lambda}} - \boldsymbol{\Xi}_0 \bar{\boldsymbol{\xi}} \right]^\top \boldsymbol{\rho}^\omega_0 + \left[b^\omega_2 - \boldsymbol{\Xi}_2 \bar{\boldsymbol{\xi}} \right]^\top \boldsymbol{\rho}^\omega_2
$$

and $\mathcal{D}^\omega$ its feasible set

$$
\begin{aligned}
\mathcal{D}^\omega = \Big\{ (\boldsymbol{\nu}^\omega, \eta^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) : \mathbf{P}^{\omega\top} \boldsymbol{\nu}^\omega + \bar{\boldsymbol{C}}^\omega \mu^\omega + \Theta^{\omega\top}_0 \boldsymbol{\rho}^\omega_0 + \boldsymbol{\Theta}^{\omega\top}_2 \boldsymbol{\rho}^\omega_2 \leq \mathbf{0}, \\
\boldsymbol{\nu}^\omega \in \mathbb{R}^{|\mathbf{T}|}, -1 \leq \mu^\omega \leq 0, \mathbf{0} \leq \boldsymbol{\rho}^\omega_0 \leq \mathbf{1}, \mathbf{0} \leq \boldsymbol{\rho}^\omega_2 \leq \mathbf{1} \Big\}
\end{aligned}
$$

The dual LP being feasible and bounded, it admits an extreme optimal solution, and its feasibility set can be replaced with the finite set of its extreme points $\mathcal{D}^\omega_*$:

$$
f^\omega(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega) = \max \left\{ G^\omega_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega}(\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) : (\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) \in \mathcal{D}^\omega_* \right\} \tag{3.1.46}
$$

It follows that the condition $\eta^\omega \geq R^\omega(\boldsymbol{\lambda}, \boldsymbol{\xi})$ in (3.1.38) can be replaced with

$$
G^\omega_{\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\xi}}, \bar{\eta}^\omega}(\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) \leq 0 \ \forall (\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}^\omega_0, \boldsymbol{\rho}^\omega_2) \in \mathcal{D}^\omega_*.
$$

We then obtain the following Benders reformulation of model F_{Gen}^{DW}:

$$F_{Gen}^{DWB}: \min \quad \sum_{\omega} \eta^{\omega} \tag{3.1.47}$$

$$s.t. \quad \mathbf{H}\boldsymbol{\lambda} = \mathbf{1} \tag{3.1.34}$$

$$\mathbf{A}_1\boldsymbol{\Lambda}\boldsymbol{\lambda} + \boldsymbol{\Xi}_1\boldsymbol{\xi} \geq \boldsymbol{b}_1 \tag{3.1.36}$$

$$-\mu^{\omega}\eta^{\omega} + \boldsymbol{\rho}_0^{\omega\top}\mathbf{A}_0\boldsymbol{\Lambda}\boldsymbol{\lambda} + \left(\boldsymbol{\rho}_0^{\omega\top}\boldsymbol{\Xi}_0 + \boldsymbol{\rho}_2^{\omega\top}\boldsymbol{\Xi}_2\right)\boldsymbol{\xi} \geq \boldsymbol{\nu}^{\omega\top}\boldsymbol{D}^{\omega} + \boldsymbol{\rho}_0^{\omega\top}\boldsymbol{b}_0^{\omega} + \boldsymbol{\rho}_2^{\omega\top}\boldsymbol{b}_2^{\omega}$$
$$\forall \omega, (\boldsymbol{\nu}^{\omega}, \mu^{\omega}, \boldsymbol{\rho}_0^{\omega}, \boldsymbol{\rho}_2^{\omega}) \in \mathcal{D}_*^{\omega} \tag{3.1.48}$$

$$\boldsymbol{\xi} \geq 0, \boldsymbol{\lambda} \in \{0,1\}^{\sum_i |\mathcal{G}_i|}, \boldsymbol{\eta} \in \mathbb{R}^{|\boldsymbol{\Omega}|}$$

Note that the number of Benders cuts (3.1.48) is exponential in the size of the input data. Moreover, a single Benders cut may involve exponentially-many terms in $\boldsymbol{\lambda}$ in formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$. The use of additional variables in formulations $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and$F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ allows for splitting the first and second stages, thus avoiding all terms involving $\boldsymbol{\rho}_0^{\omega}$. This reduces drastically the number of non-zero coefficients in formulation F_{Gen}^{DWB}.

The combined Dantzig-Wolfe and Benders reformulation of $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ (resp. $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ or $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$) is denoted by $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ (resp. $F_{(\overline{\boldsymbol{q}},\boldsymbol{m})}^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ or $F_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$)

3.1.4.4 Row-and-column generation algorithm

Formulation F_{Gen}^{DWB} is with an exponential number of $\boldsymbol{\lambda}$-variables and Benders cuts (3.1.48). The principle is to solve dynamically its linear relaxation by combining a column generation for $\boldsymbol{\lambda}$-variables and a cutting-plane technique for Benders cuts. In this section, we describe the row-and column generation algorithm devised to solve the following *master program*, which is the linear relaxation of F_{Gen}^{DWB}.

$$[MP]: \min \left\{ \sum_{\omega} \eta^{\omega} : (3.1.34), (3.1.36), (3.1.48), \boldsymbol{\xi} \geq \mathbf{0}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\eta} \in \mathbb{R}^{|\boldsymbol{\Omega}|} \right\}$$

To this aim, we introduce the *relaxed master program* $MP(\mathcal{D}_\ell)$ at *row-iteration* ℓ, obtained from MP by including the collection of Benders cuts $\mathcal{D}_\ell = (\mathcal{D}_\ell^1, \ldots, \mathcal{D}_\ell^{|\boldsymbol{\Omega}|})$ already generated in the course of the algorithm. We also define the *partial master program* $MP(\mathcal{D}_\ell, \mathcal{E}_t)$ at row-iteration ℓ and *column-iteration* t. Such program is obtained from $MP(\mathcal{D}_\ell)$ by restricting the vector of $\boldsymbol{\lambda}$-variables to the vector $\boldsymbol{\lambda}^{(t)}$ of its components whose indices are in set $\mathcal{E}_t$. Submatrices $\mathbf{H}^{(t)}$ and $\boldsymbol{\Lambda}^{(t)}$ are obtained from $\mathbf{H}$ and $\boldsymbol{\Lambda}$ by selecting the corresponding columns. Note that $MP(\mathcal{D}_\ell, \mathcal{E}_t)$ is neither a relaxation nor a restriction of MP in general, even though $MP(\mathcal{D}_\ell)$ is a

relaxation of MP and thus of NOPP.

$$MP(\mathcal{D}_\ell, \mathcal{E}_t) : \min \quad \sum_{\omega} \eta^\omega$$

$$s.t.\ \mathbf{H}^{(t)}\boldsymbol{\lambda}^{(t)} = \mathbf{1} \tag{3.1.49}$$

$$\mathbf{A}_1\boldsymbol{\Lambda}^{(t)}\boldsymbol{\lambda}^{(t)} + \boldsymbol{\Xi}_1\boldsymbol{\xi} \geq \boldsymbol{b}_1 \tag{3.1.50}$$

$$-\mu^\omega\eta^\omega + \boldsymbol{\rho}_0^{\omega\top}\mathbf{A}_0\boldsymbol{\Lambda}^{(t)}\boldsymbol{\lambda}^{(t)} + \left(\boldsymbol{\rho}_0^{\omega\top}\boldsymbol{\Xi}_0 + \boldsymbol{\rho}_2^{\omega\top}\boldsymbol{\Xi}_2\right)\boldsymbol{\xi} \geq \boldsymbol{\nu}^{\omega\top}\boldsymbol{D}^\omega + \boldsymbol{\rho}_0^{\omega\top}\boldsymbol{b}_0^\omega + \boldsymbol{\rho}_2^{\omega\top}\boldsymbol{b}_2^\omega$$
$$\forall\omega, (\boldsymbol{\nu}^\omega, \mu^\omega, \boldsymbol{\rho}_0^\omega, \boldsymbol{\rho}_2^\omega) \in \mathcal{D}_\ell^\omega \tag{3.1.51}$$

$$\boldsymbol{\xi} \geq 0, \boldsymbol{\lambda}^{(t)} \in \mathbb{R}_+^{|\mathcal{E}_t|}, \boldsymbol{\eta} \in \mathbb{R}^{|\boldsymbol{\Omega}|} \tag{3.1.52}$$

Algorithm 3.1.1: Outer loop of the column-and-row generation algorithm to solve MP. This cutting-plane component iteratively calls the column generation algorithm to solve relaxed master programs $MP(\mathcal{D}_\ell)$, which are improved at each iteration until the relaxation is tight.

```
1  ℓ ← 0 ; t ← 0 ; Initialize E_0 and D_0 ; n_c ← |E_0|
2  repeat
3      Solve MP(D_ℓ)
4      Let (λ^(t)*, ξ*, η*) be the optimal solution
5      NewCuts ← false ; D_{ℓ+1} ← D_ℓ
6      foreach ω ∈ Ω do
7          Solve (3.1.45) to compute F^ω(λ^(t)*, ξ*, η*)
8          if F^ω(λ^(t)*, ξ*, η*) > 0 then
9              Let (ν^ω*, μ^ω, ρ_0^ω*, ρ_2^ω*) be the optimal solution of (3.1.45)
10             D_{ℓ+1} ← D_{ℓ+1} ∪ {(ν^ω*, μ^ω, ρ_0^ω*, ρ_2^ω*)}
11             NewCut ← true
12     ℓ ← ℓ + 1
13 until NewCuts = false
14 return (λ^(t)*, ξ*, η*)
```

The cutting-plane algorithm given in Algorithm 3.1.1 is the outer loop of the column-and-row generation procedure to solve MP. It calls iteratively the column generation algorithm given in Algorithm 3.1.2 to solve relaxed master programs $MP(\mathcal{D}_\ell)$. At each iteration, the objective function of such programs is improved until no more improvement is possible, thus leading to a tight relaxation. At the initial iteration, it starts with a minimal set of columns and Benders cuts (see line 1 of Algorithm 3.1.1). Many strategies can be designed in this purpose. To keep the partial master programs small, our implementation choice is to start without any Benders cuts, and with a single path for each nuclear unit, namely the shortest path in each transition graph with original costs. The main loop starts by solving the current relaxed master program $MP(\mathcal{D}_\ell)$ (see line 3), thus obtaining an optimal solution $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$. If it is feasible for MP, then it is also an optimal solution for

it as $MP(\mathcal{D}_\ell)$ is a relaxation of MP with the same objective function. The Benders *separation problem* checks whether all constraints (3.1.48) are satisfied or not. It decomposes for each scenario $\omega \in \mathbf{\Omega}$ into one independent subproblem, which is to compute $F^\omega(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \eta^{\omega *})$ using LP (3.1.45) (see line 7). If $F^\omega(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \eta^{\omega *}) > 0$, then the corresponding Benders cut is violated by solution $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$. For each scenario satisfying this condition, the associated cut is added to the formulation, thus defining $\mathcal{D}_{\ell+1}$ (see line 9). The algorithm iterates solving $[MP(\mathcal{D}_{\ell+1})]$ until no more violated cuts can be found. In the latter case, $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$ is a feasible and optimal solution of MP, and the algorithm stops.

Algorithm 3.1.2: Inner loop of the column-and-row generation algorithm to solve MP. This column generation component solves partial master programs $[MP(\mathcal{D}_\ell, \mathcal{E}_t)]$, columns being iteratively added until $MP(\mathcal{D}_\ell)$ is solved to optimality.

```
1  repeat
2      Solve MP(D_ℓ, E_t)
3      Let (λ^(t)*, ξ*, η*) and (π*, μ*, σ*) be the primal and dual optimal
         solutions, respectively
4      NewCols ← false ; E_{t+1} ← E_t
5      foreach i ∈ I do
6          Solve [Pricing_i(π*, μ*, σ*)]
7          Let δ* be the optimal solution
8          if Σ_{e∈E_i} c̃_e(μ*, σ*)δ_e − π_i < 0 then
9              n_c ← n_c + 1 ; NewCol ← true
10             E_{t+1} ← E_{t+1} ∪ {n_c}
11             (Λ^(t+1))^{n_c} ← δ* ; (H^(t+1))^{n_c} ← ε_i
12     if NewCols = true then
13         t ← t + 1
14 until NewCols = false
15 return (λ^(t)*, ξ*, η*)
```

In order to solve each relaxed master program $MP(\mathcal{D}_\ell)$ involved in the inner loop, Algorithm 3.1.2 solves to optimality the partial master program $MP(\mathcal{D}_\ell, \mathcal{E}_t)$ (with the restricted set of $\boldsymbol{\lambda}-$variables $\mathcal{E}_t$, see line 2). Let us consider an optimal solution $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$, and corresponding dual values $(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)$ associated with constraints (3.1.49), (3.1.50) and (3.1.51), respectively. Linear programming theory tells us that $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$ is an optimal solution of $MP(\mathcal{D}_\ell)$ if the reduced cost (*w.r.t.* $(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)$) of all $\boldsymbol{\lambda}$-variables in $MP(\mathcal{D}_\ell)$ are non-negative. For the sake of readability, we use the following generic form of the reduced cost for variable λ_{ig}, $i \in \mathbf{I}$, $g \in \mathcal{G}_i$ [1]: $\sum_{e \in \boldsymbol{E}_i} \tilde{c}_e(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)\Lambda_e^g - \pi_i^*$, where coefficient $\tilde{c}_e(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)$ is the contribution

[1] The detailed expression of the reduced cost of variable λ_{ig}, $i \in \mathbf{I}$, $g \in \boldsymbol{\mathcal{G}}_i$ is $-\boldsymbol{\pi}^{*\top}\mathbf{H}^g - \left(\boldsymbol{\mu}^{*\top}\mathbf{A}_1 + \sum_{\omega \in \mathbf{\Omega}} \sum_{(\boldsymbol{\nu}^\omega, \boldsymbol{\rho}_0^\omega, \boldsymbol{\rho}_2^\omega) \in \mathcal{D}_*^\omega} \boldsymbol{\sigma}_{\boldsymbol{\nu}^\omega, \boldsymbol{\rho}_0^\omega, \boldsymbol{\rho}_2^\omega}^{*\top} \boldsymbol{\rho}_0^{\omega\top} \mathbf{A}_0\right) \boldsymbol{\Lambda}^g$

of $\mathbf{\Lambda}^g$ in the dual LP of $MP(\mathcal{D}_\ell)$. A remarkable feature of this expression is that it is only related to nuclear unit i. It follows that the *pricing problem*, which is to find the minimum reduced cost $\boldsymbol{\lambda}$-variable, decomposes for each unit $i \in \mathbf{I}$ into one subproblem $[Pricing_i(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)]$ (solved in line 6) is as follows.

$$
\begin{aligned}
[Pricing_i(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)] : \min & \sum_{e \in \boldsymbol{E}_i} \tilde{c}_e(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)\delta_e - \pi_i^* \\
s.t \quad & \delta_{(u,v)} - \sum_{(v,u) \in E_i} \delta_{(v,u)} = 0 && \forall u \in V_i \backslash \{s(\boldsymbol{G}_i), t(\boldsymbol{G}_i)\} \\
& \sum_{(s(\boldsymbol{G}_i),v) \in \boldsymbol{E}_i} \delta_{(s(\boldsymbol{G}_i),v)} = 1 \\
& \delta_e \in \{0,1\} && \forall e \in \boldsymbol{E}_i
\end{aligned}
$$

Subproblem $[Pricing_i(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)]$ seeks for a shortest path in the transition graph of unit i with modified costs on the arcs. If negative reduced cost $\boldsymbol{\lambda}$-variables are found, they are added to the formulation, thus defining $\mathcal{E}_{t+1}$. More precisely, line 11 appends the corresponding vector to matrix $\mathbf{\Lambda}^{(t+1)}$ and registers this column into the set of variables for unit i (by appending the i^{th} canonical vector $\boldsymbol{\varepsilon}_i$ to matrix $\mathbf{H}^{(t+1)}$). The algorithm iterates solving $MP(\mathcal{D}_\ell, \mathcal{E}_{t+1})$ until no negative reduced cost $\boldsymbol{\lambda}$-variable is found. In the latter case, $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$ is an optimal solution of $MP(\mathcal{D}_\ell)$.

3.1.4.5 Obtaining integer feasible solutions

To obtain feasible solutions for NOPP, formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ can be solved directly with an MIP solver.

The master program MP, solved using the row-and-column generation algorithm described in Section 3.1.4.4, is a building block for several methods. A first option is to use an exact method based on a branch-and-price-and-cut procedure (see *e.g.*, [Desrosiers & Lübbecke 2011]) to solve the NOPP from the master program MP. Schematically it is a branch-and-bound algorithm in which the dual bound used at each node of the search tree is MP augmented with branching constraints. Its computation is quite similar to Algorithm 3.1.1, where the relaxed master program also contains branching constraints. When solution $(\boldsymbol{\lambda}^*, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$ of the relaxation does not satisfy integrality requirements (*i.e.*, some original variable $\delta_e = \sum_{g \in \mathcal{G}_i} \Lambda_e^g \lambda_{ig}^*$ is not integer), one creates two child nodes, in which additional branching constraints $\sum_{g \in \mathcal{G}_i} \Lambda_e^g \lambda_{ig} = 0$ and $\sum_{g \in \mathcal{G}_i} \Lambda_e^g \lambda_{ig} = 1$ are imposed, respectively. Unfortunately, this exact solving procedure is not appropriate to deal with the NOPP as the computation time required is prohibitive for large-scale instances.

A second option is to use branch-and-price-based heuristics (see [Sadykov *et al.* 2019] for the presentation of several heuristics). The *pure diving heuristic* is a greedy algorithm: it first solves MP. If the obtained solution is not integer, a greedy solution is constructed by choosing a branch in the branch-and-price search tree following a given criterion. At the given branch, the master program

is solved using the column generation algorithm. The heuristic stops when an integer solution is found, or when the current node of the search tree is infeasible. Following the implementation described in [Sadykov *et al.* 2019], we choose the $\boldsymbol{\lambda}-$variable whose value is closest to 1, and branch by fixing its value to 1. In the case of a column-and-row generation algorithm the principle is similar except MP is solved by column-and-row generation before each greedy selection of a branch in the branch-and-price-and-cut tree.

This heuristic often suffers from its myopic behavior and can be improved using *least discrepancy search* instead of a purely greedy search. The idea is to explore a larger part of the search tree by allowing limited backtracking. Given a maximum discrepancy parameter $maxDiscrepancy$, the algorithm explores paths of the search tree that are obtained by applying the greedy criterion of the pure diving procedure except for, at most, $maxDiscrepancy$ branching choices. The search is also limited by forcing the use of the greedy criterion at nodes whose depth in the search tree is larger than parameter $maxDepth$.

We also investigated the use of the *restricted master heuristic*, also called *price-and-branch* heuristic. In the context of dynamic generation of columns only, the principle is to first solve the linear relaxation of the reformulated problem using column generation, thus obtaining a *restricted master program* with a subset of all the columns. Integrality requirements are then reintroduced into the current formulation, and the restricted master program is solved as a static MIP, *i.e.*, without generating new columns. In the context of column-and-row generation, this algorithm must be customized to account for the dynamically generated constraints. Our strategy is to combine the price-and-branch heuristic with the *branch-and-Benders-cut* algorithm [Fortz & Poss 2009], leading to the so-called *price-and-branch-and-Benders-cut* algorithm. The procedure first solves formulation MP using Algorithm 3.1.1, thus generating columns and Benders cuts necessary to solve the linear relaxation of the considered reformulations. Then, it yields a partial master program $MP(\mathcal{D}_\ell, \mathcal{E}_t)$. The algorithm is a heuristic in the sense that no new columns are generated after the root node processing. However, in order to obtain a feasible solution, one needs to ensure that all Benders cuts are satisfied. Formally, we solve problem $MP(\mathcal{D}_*, \mathcal{E}_t)$, where $\mathcal{D}_*$ is the set of extreme points of the dual LP (3.1.39)-(3.1.44), with additional integrality restrictions $\boldsymbol{\lambda}^{(t)} \in \{0,1\}^{|\mathcal{E}_t|}$. The corresponding formulation is solved using an MIP solver, starting with the restricted set of cuts $\mathcal{D}_\ell$ explicitly included. Whenever an integer candidate solution $(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \boldsymbol{\eta}^*)$ is found during the search, the separation problems (3.1.45) are solved via the solver's callback interface, for all $\omega \in \boldsymbol{\Omega}$. If violated Benders cuts are identified, *i.e.*, $F^\omega(\boldsymbol{\lambda}^{(t)*}, \boldsymbol{\xi}^*, \eta^{\omega *}) > 0$ for some ω, they are dynamically added to the formulation and the candidate solution is rejected.

The quality of the solutions obtained using the price-and-branch heuristic improves as the number and the diversity of columns in the restricted master program increases. The *diving with sub-MIPing heuristic* exploits this idea by first calling the pure diving heuristic, and second solving a restricting master composed of all the columns generated during the diving (with integrality restrictions). This method

can suffer from a large number of columns, leading to elevated computation times.

3.1.5 Computational results

3.1.5.1 Instances

The proposed formulations and solution approaches are evaluated on a real data-set of the french electricity production with a time horizon ranging from January 2015 to December 2018. The data-set is composed of 58 nuclear units and 84 other sources accounting for non-nuclear units, and exchanges on the market spot.

The stochastic data are given through 5 sets of 96 scenarios of demand and non-nuclear unit characteristics (production bounds and leasing costs). From this original data-set, we derive several NOPP instances by reducing the number scenarios and possibly scaling down the fleet. From each set of scenarios and given an integer S, a dedicated Scenario Clustering Library provided by EDF generates S aggregated scenarios. This library clusters similar original scenarios using a norm based on a heuristic cost evaluation of demand satisfaction accounting for a set of random parameters attached to each scenario (demand, availability of non-nuclear units, prices/capacities on the electricity spot market).

The fleet is scaled down when needed by considering a subset of the nuclear units as non-nuclear units, thus reducing the nuclear share in the instances while the other data are not changed. Such instances are referred to as *scaled-down instances* in the sequel. To maintain local power plant scheduling constraints valid, we first select a subset of nuclear power plants that is kept unchanged in the scaled-down instances. In the data-set, a baseline planning defines, for each of the remaining nuclear units i, outage periods during which its production is set to zero, while they can produce between zero and $\overline{P}_{it}$ for other periods $t \in \mathbf{T}$. The corresponding constraints on modulation and fuel level are relaxed, and removed from all scheduling constraints. Last the leasing cost of the new non-nuclear units is derived from real data relative to EDF nuclear units.

Instances are classified into instance types, labeled using the following pattern *pAiB-C.D.EsS* where:

- A is the number of nuclear units. $A = 58$ corresponds to EDF current nuclear fleet, $A = 22$ and 40 are considered for scaled-down instances.
- B is the instance type tag number. Each unique value of B corresponds to a selection of a subset of nuclear units in the real data-set that are kept unchanged, and a number of aggregated scenarios.
- $C \in \{0,1\}$ is the number of 6-unit power plants in type B.
- $D \in \{0, 1, \ldots, 8\}$ is the number of 4-unit power plants in type B.
- $E \in \{0, 1, \ldots, 10\}$ is the number of 2-unit power plants in type B.
- S is the number of aggregated scenarios in type B.

Note that instances inside a same type differ by the set of considered scenarios.

The formulation of the first stage problem described in Section 3.1.3.2 has strong linear relaxation, but it comes at the price of a very large number of constraints and variables. For real instances p58i0-1.8.10sS (for any number of scenario S), the transition graphs lead to around 120.000 binary arc variables, 30.000 flow conservation constraints (corresponding to the number of nodes in all graphs) and 3.000 scheduling constraints binding outage arcs from different graphs. Hence, anticipating that it will not be possible to solve the formulation directly with a MIP solver for a large number of scenarios, we will study the use of Dantzig-Wolfe decomposition-based heuristics combined or not with Benders' decomposition. The Dantzig-Wolfe reformulation is not used here to obtain an improvement in the relaxation but for its ability to decompose the problem into smaller subproblems solved iteratively.

Tests are carried out on a Linux machine equipped with 2 × 12-core Haswell Intel Xeon E5-2680 v3 CPUs with 128 Go of RAM. Modeling and solving are done using BaPCod which is a black-box framework dedicated to solve MIPs using reformulation techniques such as Dantzig-Wolfe and Benders decompositions [Vanderbeck 2011]. At most one column for each nuclear unit is added to the partial master program at each iteration. To improve the convergence of the column generation procedure, we use stabilization by automatic smoothing the dual variables of the master program, as described in [Pessoa *et al.* 2018b]. We use `Lemon` library for modeling the graph structure and shortest path algorithm, and `Boost 1.56` library for parallelization of Benders subproblems (the solution of column generation subproblems is sequential). The MIP solver used is Cplex solver 12.7.1. In order to have a fair comparison between the different solution approaches we limit to 6 the number of threads used by the MIP solver in the tests of Sections 3.1.5.3 and 3.1.5.5. In Section 3.1.5.6 the solver is used with default settings.

3.1.5.2 Comparing arc flow-based formulations

In this section a few preliminary results is provided to show the comparative performance of the first two proposed MIP formulations, namely $F_{(\underline{q},\overline{q})}(\boldsymbol{\delta}, \boldsymbol{p})$ given in Section 3.1.3.4 and $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ in Section 3.1.3.5. Table 3.1.2 shows the computation time in seconds required to solve the LP relaxation of each formulation for the simplified NOPP instances with 22 nuclear units. The difference in terms of performance is quite significant in favor of formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$. Then the latter formulation should lead to better performance when used by a linear solver at each node of a branch and bound tree or at each column generation iteration.

3.1.5.3 Comparing heuristics to solve deterministic instances

In this section the aim is to compare different solution approaches on simplified deterministic instances to select the most promising approach anticipating real size stochastic instances. Note that the cut generation procedure is not needed to solve a deterministic instance, nor is the use of capacity variables defined in (3.1.16)-(3.1.17). It would only add variables in the master of the column generation pro-

Instance type	$F^{LP}_{(\underline{q},\overline{q})}(\delta, p)$	$F^{LP}(\delta, s, p)$
p22i0-1.4.0s1	142,2	55,8
p22i1-0.1.9s1	129,2	26,0
p22i2-1.3.2s1	182,4	47,6
p22i3-0.4.3s1	159,2	35,8
p22i4-0.4.3s1	102,0	28,0

Table 3.1.2: Comparative computation time (in seconds) to solve the LP relaxation of $F_{(\underline{\boldsymbol{q}},\overline{\boldsymbol{q}})}(\boldsymbol{\delta}, \boldsymbol{p})$ and $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$.

cedure without any benefit. Hence, we compare the performance using the original integer formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$, as defined in Section 3.1.3.5, solved directly with Cplex solver or using Dantzig-Wolfe reformulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ solved by a column generation algorithm, as described in Section 3.1.4.4, followed by a heuristic to obtain a feasible solution for NOPP.

Note that the work presented in this article focuses on solving efficiently the linear relaxation of several formulations to find a good primal solution in terms of quality through a heuristic. Hence we used the generic column generation-based heuristics of the literature without any further analysis of the internal solving process for improvement. Such work is beyond the scope of this article. The following benchmark of heuristics, described in Section 3.1.4.5, is considered:

- Price-and-branch: First price, i.e., solve formulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ by column generation at the root node, and then branch, i.e., enforce integrity constraints and solve the resulting restricted master program to optimality using Cplex solver.

- Pure diving: Formulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ is solved by column generation followed by a diving heuristic with maxDiscrepancy=0 and maxDepth=0, i.e., greedy construction of a solution alternating branching in the branch-and-price tree and solving the master problem.

- Diving23: Formulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ is solved by column generation followed by a diving heuristic with maxDiscrepancy=2 and maxDepth=3.

- Pure diving + price-and-branch: Formulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ is solved by column generation followed by a combination of a pure diving heuristic before price-and-branch. Then the initial solution from pure diving could lead to generate improving columns, thus possibly reducing the number of visited nodes during the price-and-branch.

Comparative results for simplified deterministic instances using the column generation based heuristics benchmark are presented in Table 3.1.3, which features for each instance type:

- $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ - Int.gap: average integrity gap $\frac{opt-b}{opt}$ between the integer optimum opt and the optimal linear bound b.

- $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ - CPU(s): average computation time in seconds to solve the mixed integer original formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ to optimality with Cplex solver;

- for each heuristic in the benchmark:

  - gap: average gap $\frac{p-opt}{p}$ between the integer value p found by the heuristic and the integer optimum *opt* found by $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$;

  - CPU(s): average computation time in seconds.

Note first that for this set of instances the optimal value is obtained with $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ solved directly by Cplex solver. Therefore, the gap being zero is not shown and replaced by the integrity gap in Table 3.1.3. Note also that b is the optimal value of both the linear relaxation $F^{LP}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and the Dantzig-Wolfe reformulation $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$.

	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$		Price-and-branch		Pure diving		Diving23		Pure diving +Price-and-branch	
Instance type	Int.gap(%)	CPU(s)	gap(%)	CPU(s)	gap(%)	CPU(s)	gap(%)	CPU(s)	gap(%)	CPU(s)
p22i0-1.4.0s1	0,67	68,2	0,06	54,4	0,06	97,4	0,06	680,2	0,05	64,4
p22i1-0.1.9s1	1,03	67,8	0,20	40,4	0,30	64,2	0,30	429,2	0,13	66,4
p22i2-1.3.2s1	0,59	71,4	0,01	34	0,10	44	0,06	336,2	0,00	40,8
p22i3-0.4.3s1	1,26	67,4	0,61	30,2	0,50	85	0,48	445,4	0,05	94,2
p22i4-0.4.3s1	0,68	53,5	0,01	18,75	0,05	43,5	0,04	210,75	0,01	42,75
p40i0-1.8.1s1	1,23	282,6	0,91	186,8	13,97^a	391,8	13,97^a	3494,6	0,16	489,6
p40i1-0.5.10s1	0,86	151,8	0,95	110	5,40^b	185,2	5,20^b	1848,2	0,21	216,6
p40i2-1.5.7s1	1,26	1052	0,46	406,4	0,83	403,8	0,70	4587,6	0,16	919,4

Table 3.1.3: Comparing performance of exact solutions and solutions obtained using the column generation based heuristics benchmark for simplified deterministic instances.

The performance results presented in Table 3.1.3 correspond to average with respect to sets of five instances. The average reflects quite well the performance results of each instance in the corresponding set for all sets of instances but for two sets relative to 40 nuclear unit instances. For these two latter instances, gaps in Table 3.1.3 appear with superscript ^{a} (resp. ^{b}) to indicate cases where a large variability in the gaps have been observed, namely 3 (resp. 1) out of 5 solutions are with a gap around 20-25%, whereas 2 (resp. 4) out of 5 solutions are with a gap close from 1%. In other words, diving heuristics either find solutions with a gap less than 1% or around 20-25%. An explanation for such behavior is twofold. First binding constraints are active on 40 unit instances while they are inactive on 22 nuclear unit instances. Second columns do not have coefficients in the objective function, thus leading to poor branching choices. Anticipating solving large size instances, the computation time should be less than that obtained for solving $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ directly by Cplex solver.

The ranking of the different heuristics relative to the computation time required to solve $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ directly by Cplex solver is as follows: diving23 is extremely slow, pure diving heuristic is slow, pure diving + price-and-branch is close to Cplex solver, and price-and-branch outperforms Cplex solver. The price-and-branch heuristic

always finds a solution with a very good gap less than 1% in average and with a computation time 30-50% less than that obtained by Cplex solver. Note that in terms of quality, pure diving + price-and-branch finds excellent solutions but with a computation time close to that obtained with Cplex solver.

This benchmark suggests to try solving real instances directly with Cplex solver whenever possible and to use price-and-branch heuristic otherwise.

3.1.5.4 Model validation

The main thrust of this work is that outage dates computed using the NOPP formulation will lead to operational savings when accounting for an increased number of scenarios.

As mentioned in Section 3.1.1, the operational outage planning process is a multi-stage procedure involving successive re-optimizations of outage dates and power productions on a rolling horizon. Designing a code to emulate this multi-stage process to evaluate a first-stage solution over a set of validation scenarios is beyond the scope of this paper. We use a dedicated tool developed at EDF to evaluate solutions in the limited framework of a two-stage process (consistent with the structure of NOPP). This library, referred to as *Checker*, takes as input a scenario and a given nuclear outage plan, and optimizes the production of the units while meeting the demand over the time horizon, thus emulating a NOPP second stage. This is modeled as a simple linear program, which allows refining the time discretization to six periods per day (instead of one per week used in the optimization models).

The cost of each first-stage solution is evaluated as the expected second-stage cost calculated with the checker, over 96 scenarios. For each of the 25 (resp. 15) scaled-down instances with 22 (resp. 40) nuclear units, we computed first-stage solutions with $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$. Recall that $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ is directly solved with Cplex solver as defined in Section 3.1.3.5, while $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ is to solve the linear relaxation with column generation and to use the Price-and-branch heuristic selected in section 3.1.5.3. The number of aggregated scenarios considered for the optimization ranges in $S \in \{1, 5, 10, 15\}$. This allows us to estimate the savings obtained when using a given method with a given number of aggregated scenarios, compared to the deterministic case, i.e., with one aggregated scenario. More precisely, the savings correspond to the expected cost of the first-stage solution obtained with a method and S scenarios minus the expected cost of the solution obtained with $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and $S = 1$. In Table 3.1.4 statistics relative to the savings are reported over the whole set of simplified 22-unit instances (resp. 40-unit instances), denoted by p22* (resp. p40*). Rows "avg. savings" and "stddev. savings" respectively show the average and standard deviation, over the considered instance type, of savings evaluated by the Checker over the 96 original scenarios. Note that the objective functions of the scaled-down instances is considerably increased compared to the original ones: this is a side effect of virtually converting nuclear units to thermal ones. To better emphasize the benefit of using more scenarios, absolute savings are reported. The order of magnitude of an absolute difference of 1 monetary unit

is, here, 0.001% in relative difference. A one-hour time limit is imposed for each run. Within this time limit, Cplex solver ($F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$) converges to optimality for all runs up to $S = 10$. For 22 units and $S = 15$, optimality was not proven but the optimality gap is less than 0.15%. We do not report results for instances with 40 units and $S = 15$, because both methods failed at finding feasible solutions for most instances.

Instance type	Statistic	Method	S=1	S=5	S=10	S=15
p22*	avg. savings	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	0.00	7.28	8.62	11.80
		$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-3.64	4.75	2.22	4.62
	stddev. savings	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	0.00	6.65	5.79	6.47
		$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	3.72	7.43	7.85	6.10
p40*	avg. savings	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	0.00	21.47	23.49	-
		$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-9.83	0.53	8.42	-
	stddev. savings	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	0.00	15.67	15.94	-
		$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	9.85	16.97	17.02	-

Table 3.1.4: EDF Checker evaluation of optimal and heuristic solutions, absolute difference w.r.t. the planning computed on deterministic instances.

We observe that the expected savings of optimal solutions ($F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$) increase with the number of aggregated scenarios. The same trend is obtained with heuristic solutions even if the corresponding expected savings are not as good as the one obtained with optimal solutions, they globally increase with the number of aggregated scenarios. However, the case of $S = 5$ with 22 units appears as an outlier that can be explained by the variability in terms of quality of the heuristic solutions.

3.1.5.5 Comparing formulations for simplified stochastic instances

This section aims at selecting, on simplified stochastic instances, the most promising approaches to solve real size stochastic instances. Several formulations and solution approaches might be efficient on stochastic instances depending on the number of aggregated scenarios taken into account. Note first that linear relaxation is a major component to any solution approach. Then to limit the number of experiments to be included in the article, the comparative results for stochastic scaled-down instances are performed with the linear relaxation of the problem using the following benchmark of formulations along with their solution approaches.

- $F^{LP}(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$: Linear relaxation of the MIP formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ described in Section 3.1.3.5 and solved directly using Cplex solver.

- $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$: the Dantzig-Wolfe reformulation, as described in Section 3.1.4.2, solved by column generation as described in Section 3.1.4.4.

- $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$: the Dantzig-Wolfe and Benders reformulation as described in

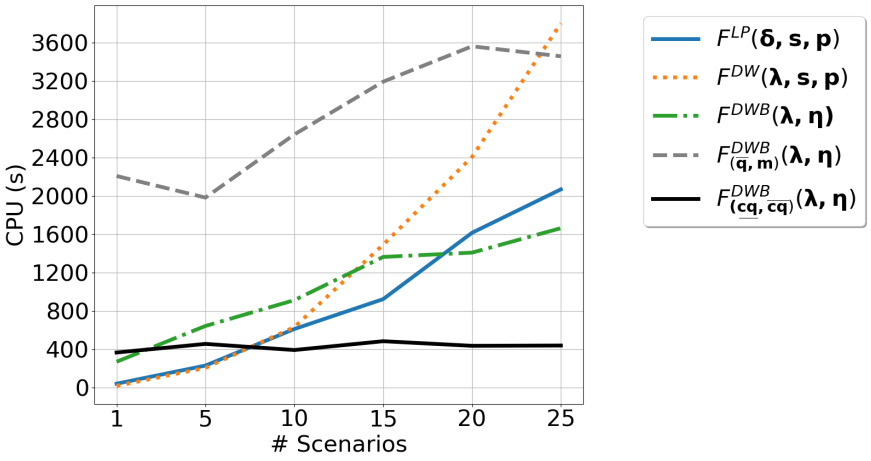


Figure 3.1.4: Comparison of computation times for solving the LP relaxation of 22 nuclear unit stochastic instances using a benchmark of formulations.

Section 3.1.4.2 and 3.1.4.3 and solved by column-and-cut generation algorithms as described in Section 3.1.4.4.

- $F^{DWB}_{(\overline{q},m)}(\lambda, \eta)$: The Dantzig-Wolfe and Benders reformulation using intermediary variables $\overline{q}$ and m linking first and second stage as described in Section 3.1.3.6. and solved by column-and-cut generation algorithms as described in Section 3.1.4.4.

- $F^{DWB}_{(\underline{cq},\overline{cq})}(\lambda, \eta)$: the Dantzig-Wolfe and Benders reformulation using intermediary variables $\underline{cq}$ and $\overline{cq}$ linking first and second stage as described in Section 3.1.3.6 and solved by column-and-cut generation algorithms as described in 3.1.4.4.

Figure 3.1.4 shows the average computation time on sets of 25 stochastic instances, five for each possible structure with 22 nuclear units. The stochastic instances are the same as the 22 nuclear unit deterministic instances presented in Table 3.1.3, but for the number of scenarios S which ranges from 1 to 25.

Approaches combining Dantzig-Wolfe and Benders reformulations $F^{DWB}(\lambda, \eta)$, $F^{DWB}_{(\overline{q},m)}(\lambda, \eta)$ and $F^{DWB}_{(\underline{cq},\overline{cq})}(\lambda, \eta)$ solved through column-and-cut generation are slower on instances with few scenarios. Whereas approaches without Benders decomposition $F^{LP}(\delta, s, p)$ and $F^{DW}(\lambda, s, p)$ are faster, but with rapid deterioration of performance as the number of scenarios increases. To be more specific, the solution time of $F^{LP}(\delta, s, p)$ (resp. $F^{DW}(\lambda, s, p)$) ranges from less than 60 seconds for one scenario to more than 2000 seconds for solving $F^{LP}(\delta, s, p)$ (resp. 3600 seconds for solving $F^{DW}(\lambda, s, p)$) with 25 scenarios. Beyond 15 scenarios, $F^{DWB}(\lambda, \eta)$ and $F^{DWB}_{(\underline{cq},\overline{cq})}(\lambda, \eta)$ become more efficient than $F^{LP}(\delta, s, p)$. Interestingly the solution time increases really slowly – indeed is almost constant – with respect to the number

of scenarios for $F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda},\boldsymbol{\eta})$. This clearly shows that the reformulation with cumulative capacity variables is the most efficient to solve the LP relaxation for stochastic instances with a large number of scenarios. The computation time appears to be linear in the number of scenarios up to 15 using $F^{DWB}(\boldsymbol{\lambda},\boldsymbol{\eta})$ (green plot in Figure 3.1.4) while it is almost constant using $F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda},\boldsymbol{\eta})$ (black plot in Figure 3.1.4). It is interesting to check in more details which components of the column-and-cut algorithms are making the difference. We consider a component-wise comparison of computation times to evaluate more closely the performances of each solution approach. Table 3.1.5 provides for each of the three formulations solved through cut-and-column generation the following entries evaluated on an average of the 22 nuclear unit stochastic instances :

- S is the number of aggregated scenarios;
- For column generation (resp. cut generation), denoted by $ColGen$ (resp. $CutGen$), $\#it$ is the number of iterations and $\#Col$ (resp. $\#Cuts$) is the total number of columns (resp. Cuts) generated.
- TotalLP is the total time spent solving the linear relaxation
- SolMaster is the total time spent solving the restricted master problem during column generation,
- SolSep is the time spent solving the separation subproblems and generating cuts which involves the projection,
- UpPric is the time to update arc costs with dual variables coming from the master solution,
- UpSep + SolPric is the sum of the time spent updating first-stage decisions in the separation subproblems and solving the shortest path problem – i.e., pricing subproblem – and generating new columns. The two components have been added together as they represent a really small part of the time spent in other steps.

First note that the number of generated columns variations does not seem to have a clear link with the number of aggregated scenarios for any formulation and for a given number of scenarios all formulations require a similar number of cuts and cut generation iterations. Second it appears that $F^{DWB}_{(\boldsymbol{q},\boldsymbol{m})}(\lambda,\eta)$ requires twice as many column generation iterations compared to others formulations and far more columns. This deeply affects the performance of the latter formulation. Formulation $F^{DWB}_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}$ requires the same number of column generation iterations as formulation F^{DWB} does but 1.3 times more columns, which explains the relative low efficiency of formulation $F^{DWB}_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}$ for few scenarios. Then an interesting question that arises is why the solving time is constant w.r.t. the number of scenarios with intermediary variables $(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})$, while it is linear without.

		ColGen		CutGen		Computation time component-wise (s)				
S	Formulation	# It	# Col	# It	# Cuts	TotalLP	SolMaster	SolSep	UpPric	UpSep + SolPric
1	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	410	1590	90	89	159,8	27,7	80,6	43,0	2,0
	$F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	433	2297	89	88	370,4	151,5	67,2	116,1	3,4
	$F^{DWB}_{(\overline{q},m)}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	868	5010	88	87	1510,2	1182,0	92,0	196,1	6,2
5	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	355	1539	52	245	445,7	78,9	215,3	124,0	6,2
	$F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	365	2228	51	240	374,5	145,1	71,1	108,0	5,8
	$F^{DWB}_{(\overline{q},m)}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	747	4613	52	242	1274,0	937,0	80,6	189,4	8,8
10	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	342	1545	43	405	715,7	102,1	365,6	193,9	9,3
	$F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	346	2242	44	406	421,8	157,6	87,7	109,1	8,6
	$F^{DWB}_{(\overline{q},m)}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	685	4391	44	405	1411,0	1050,0	94,9	178,3	10,8
15	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	360	1671	42	590	1036,9	159,3	499,3	288,7	12,7
	$F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	362	2283	41	576	526,0	125,0	105,0	131,7	3,9
	$F^{DWB}_{(\overline{q},m)}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	729	4453	41	579	1878,6	1460,0	105,8	193,6	14,5

Table 3.1.5: Comparative performance for solving the LP relaxation for 22 nuclear unit stochastic instances using the three Danzig-Wolfe and Benders formulations through cut-and-column generation.

Let us look at the time spent in each component of the algorithm. First note that the time spent in components SolMaster and UpPric is larger in a deterministic setting with intermediary variables $(\overline{\boldsymbol{q}}, \boldsymbol{m})$ or $(\underline{\boldsymbol{cq}}, \overline{\boldsymbol{cq}})$. This is consistent with the larger number of generated columns. However the time spent in components SolSep and UpPric is almost constant with intermediary variables $(\underline{\boldsymbol{cq}}, \overline{\boldsymbol{cq}})$ while it grows w.r.t. the number of scenarios without. The rationale behind is that without intermediary variables one needs to perform a projection as described in Section 3.1.3.6. The number of coefficients, involved in the generated cuts rewritten with the original variables or in the pricing update to add new cuts with dual values, increases rapidly with the number of aggregated scenarios (and hence the number of generated cuts) taken in account. This results in a significant increase in the time spent in components SolSep and UpPric, i.e., components of the algorithms where projection is performed, which is 82% of the total time increase. This clearly answers the question.

3.1.5.6 Solving the real large-scale instances

The aim in this section is to perform a final comparative evaluation among formulations and solution approaches, which have passed previous evaluations with reasonable chances to solve large size instances. Contrary to Section 3.1.5.5, we are looking to mixed integer solutions. The considered benchmark is:

- $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$: Original MIP formulation solved directly using Cplex solver.
- $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$: Dantzig-Wolfe reformulation solved by column generation, followed by the best promising heuristic from Section 3.1.5.3, namely price-and-branch as described in Section 3.1.4.5.

- $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$: Dantzig-Wolfe and Benders reformulation solved by column-and-cut generation algorithms, followed by the price-and-branch as described in Section 3.1.4.5.

- $F^{DWB}_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$: the Dantzig-Wolfe and Benders reformulation using intermediary variables $\underline{\boldsymbol{cq}}$ and $\overline{\boldsymbol{cq}}$ and solved by column-and-cut generation algorithms, followed by the price-and-branch as described in Section 3.1.4.5.

Note that for Dantzig-Wolfe and Benders reformulations we had to adapt the price-and-branch heuristic as it was originally designed for a Dantzig-Wolfe reformulation. The principle is to use a usercut callback in Cplex solver to keep generating cuts during the heuristic step as described in Section 3.1.4.5. All benchmark formulations are used to solve real size instances within a total time limit of 8 hours. Table 3.1.6 shows results over a set of five 58 nuclear unit stochastic instances corresponding to p58i0-1810sS with S ranging from 1 to 48. Table 3.1.6 uses the same entries as Table 3.1.5 in Section 3.1.5.5, but Gap which is the average gap $\frac{p-b}{b}$ between the integer value p found by the heuristic and the linear relaxation value b. Note that b is the same for all formulations, then it is not useful in the evaluation. Nor it is to show the computation time, as the 8-hour time limit is reached by all solution approaches, but $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ for S=1 in the deterministic case. In the latter case, the average computation time is 4667 seconds to find an optimal solution, whereas it is 1739 seconds to find a primal feasible solution. Finally whenever no integer feasible solution is found within the time limit for at least one of the five instances, this is indicated with symbol "-" in the corresponding cell of Table 3.1.6. Similarly whenever an entry in the table is not relevant for a formulation, e.g., # ColGen it in the case of formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ as it is solved directly with Cplex solver or # CutGen it in the case of formulation $F(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ as there is no Benders cut generation.

The sparse structure of formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ allows Cplex solver to find solutions with less than 1% to optimality within the 8-hour time limit up to 10 stochastic scenarios, whereas $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ provides good quality solutions in less than 30 minutes in the deterministic case. The time spent solving master problems grows rapidly to more than 6 hours 30 minutes for 10 scenarios, thus explaining why $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ and $F(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$ could not find a good feasible solution within the 8-hour time limit. This calls for the use of column-and-cut generation approaches.

Confirming results from Section 3.1.5.5 both $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ and $F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ requires roughly the same number of cut generation iterations and number of cuts to solve the linear relaxation. $F^{DWB}_{(\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ generates once again 30% more columns than $F^{DW}(\boldsymbol{\lambda}, \boldsymbol{\eta})$, thus spending more time to solve the master problems and likewise to solve the LP relaxation for few scenarios. It is worth noting that the time spent solving separation problems and generating cuts represent the largest part of the computation time for few scenarios. Not surprisingly this is the step taking most of the total time when searching for an integer solution with Cplex solver. This is also the reason why none of the formulations using column-and-cut generation

		LP relaxation						LP + heuristic (s)			
S	Formulation	# ColGen It	# Col	# CutGen It	# Cuts	TotalLP	SolMaster	SolSep	UpPric	UpSep + SolPric	Gap
1	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	0.49
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	79	1768	-	-	255	180	-	52	2	0,57
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	1146	7247	723	722	6462	3993	22332	1523	94	0.57
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	1021	10035	768	767	9740	8079	10819	772	113	0.66
5	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	0.71
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	79	1925	-	-	5199	4702	-	394	4	0,87
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	738	6168	227	1111	8032	3511	19297	2485	177	1.04
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	691	9454	227	1115	8570	7250	10309	279	283	0.94
10	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	0.89
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	69	1845	-	-	24268	23112	-	927	8	25,86
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	690	6004	173	1688	11319	5047	20339	3229	175	1.37
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	625	8586	155	2249	7264	5766	13556	572	363	1.61
15	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	649	5810	157	2295	15694	7317	19256	4133	254	1.74
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	610	8424	155	2249	9069	7256	14286	604	504	2.62
25	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	597	5537	137	3214	21744	10978	11674	4768	206	1.96
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	570	8101	139	3208	12202	10003	13805	560	379	2.22
32	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	567	5345	124	3863	24991	12823	9741	4939	230	2.22
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	522	7523	123	3816	11310	9202	13357	478	505	2.76
48	$F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DW}(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{p})$	-	-	-	-	-	-	-	-	-	-
	$F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	-	-	-	-	-	-	-	-	-	-
	$F^{DWB}_{\underline{\boldsymbol{cq}},\overline{\boldsymbol{cq}}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$	548	7334	124	5537	19893	17008	7573	482	414	4.48

Table 3.1.6: Final comparative performance for solving the real large-scale instances.

does finish within the time limit even for few scenarios. Formulation $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ solved through column-and-cut generation with projection leads to high quality solutions for instances with up to 32 stochastic scenarios. Even though $F^{DWB}_{(\underline{cq},\overline{cq})}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ solves the linear relaxation faster than $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ for more than five scenarios, it does not reflect on the gap as the performance of the heuristic is the other way around. This shows that Cplex solver performance is impaired by the up-sizing of the formulation induced by the additional columns and variables. Beyond 32 scenarios, the LP relaxation solution time using $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ raises up to 7 hours due to the time increase in components SolSep and UpPric by projection, as shown in Section 3.1.5.5. Then solving reformulation with cumulative capacity variables via column-and-cut generation becomes the best solution approach. In particular, it allows us to solve the LP relaxation faster and provides us with solutions within 4.48% of optimality.

The presented comparative results lead to define a strategy for solving real instances: for less than 10 aggregated scenarios, use formulation $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$ with Cplex solver whereas for more than 10 scenarios, use $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ instead. Introducing cumulative splitting variable, leading to formulation $F^{DWB}_{\underline{cq},\overline{cq}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$, tackled the projection issue, which is the bottleneck of formulation $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$. In particular, using $F^{DWB}_{\underline{cq},\overline{cq}}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ allows us to find integer solutions up to 48 scenarios, whereas $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ is limited to 32 scenarios. Hence, a perspective for future work would be to find a formulation having both properties of solving the linear relaxation within a time almost constant in the number of scenarios and efficient solution by Cplex solver.

Finally, we estimate the industrial potential savings of the approaches based on the NOPP formulations presented in this article using the EDF Checker. We evaluated the best solution found for each number of aggregated scenarios up to 32. Directly using Cplex solver on $F(\boldsymbol{\delta}, \boldsymbol{s}, \boldsymbol{p})$, the expected savings (computed as in Section 3.1.5.4) for ten scenarios over a three year horizon are evaluated to 0.56%. Solving $F^{DWB}(\boldsymbol{\lambda}, \boldsymbol{\eta})$ using the column-and-row generation algorithm followed by price-and-branch heuristic allows us considering 32 aggregated scenarios, thus leading to increased expected savings of 1.11%. To put this in perspective, 1% of gain corresponds to approximately 50 million euros.

3.2 Decomposition for two-stage robust problems with mixed integer recourse

This section is based on [Arslan & Detienne 2021]. We study a class of two-stage robust binary optimization problems with objective uncertainty where recourse decisions are restricted to be mixed-binary. For these problems, we present a deterministic equivalent formulation through the convexification of the recourse feasible region, that is obtained using either Dantzig-Wolfe relaxation (see Section 1.2.2.2)

or DP-based reformulation (see Section 1.2.2.1). We then explore this formulation under the lens of a relaxation, showing that the specific relaxation we propose can be solved using the branch-and-price algorithm. We present conditions under which this relaxation is exact, and describe alternative exact solution methods when this is not the case. Despite the two-stage nature of the problem, we provide NP-completeness results based on our reformulations. Finally, we present various applications in which the methodology we propose can be applied. We compare our exact methodology to those approximate methods recently proposed in the literature under the name $K-$adaptability. Our computational results show that our methodology is able to produce better solutions in less computational time compared to the $K-$adaptability approach, as well as to solve bigger instances than those previously managed in the literature.

From a methodological perspective, the main contribution of this work is providing an efficient algorithmic framework to solve a class of two-stage robust optimization problems to exact optimality. The methods developed outperform the existing approaches, either exact or approximate, for this specific class.

From a theoretical point-of-view, the complexity the reformulation presented therein paves the way to a theoretical complexity result that shows that the studied two-stage robust optimization problem is $\mathcal{NP}$-complete. The non-triviality of this result resides in the problem being *at most* $\mathcal{NP}$-complete (the exact complexity of such problems often being unclear, as discussed in Section 3.2.3).

The remainder of this section is organized as follows: Section 3.2.1 describes the problem in its context and related literature. In Section 3.2.2 we present a single-stage equivalent reformulation of (3.2.1) through convexification of the recourse feasible region, $\mathcal{Y}(\boldsymbol{x})$, and propose a computationally attractive relaxation of $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$. The reformulations we obtain lead to deterministic equivalent models that can be solved using either the branch-and-price or the branch-and-price-and-cut algorithm. In Section 3.2.3, we present related complexity results. In Section 3.2.4, we illustrate two different applications of our methodology, and present a numerical evaluation of the proposed column generation-based approaches, comparing them to the direct solution of extended and K-Adaptability formulations.

3.2.1 Introduction and literature review

Robust optimization is an approach to handling uncertainty in optimization where the probability distributions are replaced with uncertainty sets. In robust optimization, constraints are imposed for all realizations whereas the objective function is evaluated for the worst-case realization within the uncertainty set. As such, in applications where the effects of uncertainty can be catastrophic, robust optimization presents itself as a viable modeling approach. Further, robust optimization models with polyhedral or convex uncertainty sets lead to deterministic equivalent formulations that are often in the same complexity class as their deterministic counterparts. For these reasons, robust optimization has enjoyed and continues to enjoy a growing attention from the research community. Advances in static robust optimization are

presented in [Ben-Tal *et al.* 2009], [Bertsimas *et al.* 2011] and [Gabrel *et al.* 2014].

On the other hand, robust optimization can sometimes be "over-conservative", especially when uncertainty does not have a row-independent structure. To remedy this problem, when the underlying application permits it, one might consider introducing recourse (adjustability/adaptability) after the realization of uncertainty. Further, some applications may naturally involve a set of "wait-and-see" decisions that are taken after the realization of uncertainty. This is the case, for instance, where strategic design decisions are undertaken by evaluating the future operational conditions of a system under uncertainty. In robust optimization with recourse, first-stage decisions are evaluated by taking the possibility to "recover" a feasible solution after the realization of uncertainty into account. The difficulty of these problems has long been established in the literature even in the simple case of two-stage adjustable robust optimization with linear programming problems in both stages, and a polyhedral uncertainty set (see [Ben-Tal *et al.* 2004]). We remark that another approach to deal with the over-conservativeness of static robust optimization is to consider distributionally robust uncertainty models (e.g. [Goh & Sim 2010]). However, its discussion is out of the scope of this work.

Example 3.2.1. *Consider the static robust optimization problem*

$$\max_{x\in\{0,1\},\boldsymbol{y}\in\{0,1\}^3} -x + \min_{\xi\in[0,1]} \quad (3-2.5\xi)y_1 + (-1+4\xi)y_2 + (4-6\xi)y_3$$
$$s.t. \quad y_1 + y_2 + y_3 \leq x$$

Its optimal solution is $x = y_1 = y_2 = y_3 = 0$, with objective value 0 (as when $x = 1$ the objective value is -0.5). Consider now the possibility to take decisions $\boldsymbol{y}$ after the realization of uncertainty. We write the adjustable robust optimization problem

$$\max_{x\in\{0,1\}} -x + \min_{\xi\in[0,1]} \max_{\boldsymbol{y}\in\{0,1\}^3} (3-2.5\xi)y_1 + (-1+4\xi)y_2 + (4-6\xi)y_3$$
$$s.t. \quad y_1 + y_2 + y_3 \leq x$$

In Figure 3.2.1, we present the functions $(3-2.5\xi)$, $(-1+4\xi)$ and $(4-6\xi)$. The maximum of these three functions is to be minimized over $\xi \in [0,1]$ when $x = 1$. This is achieved at $\xi = 0.61$ and yields a value of 1.46. On the other hand when $x = 0$ the second-stage value is trivially 0. The optimal solution to the two stage problem is therefore $x = 1$ with value 0.46.

Example 3.2.1 highlights the value of incorporating wait-and-see decisions into robust optimization. Strategic decisions are improved, by making a more judicious evaluation of the effects of uncertainty. It also demonstrates why doing so is not straightforward. Indeed, for each first-stage solution, evaluating its second-stage cost requires the solution of a bilevel optimization problem where the optimal solution of the outer level is not necessarily an extreme point of the corresponding feasible region.

Existing methodologies in robust adjustable optimization can be categorized as exact or approximate. Exact approaches do not pose any assumptions on the

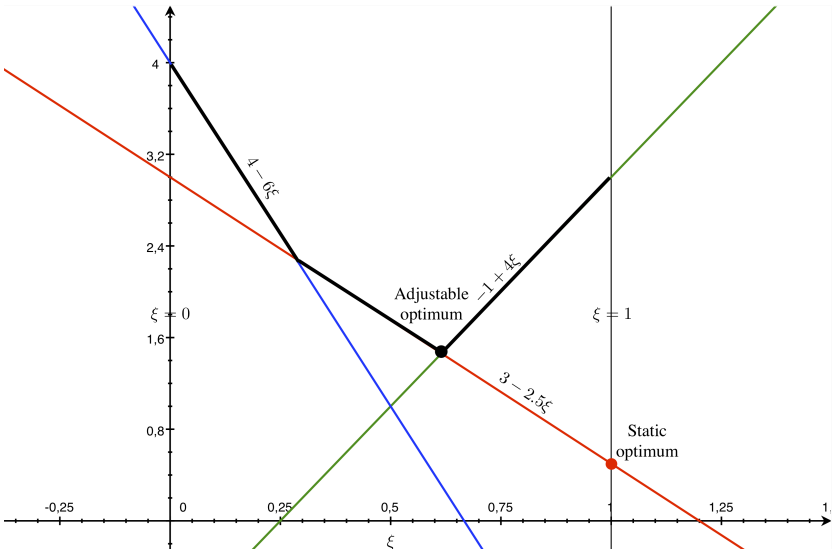


Figure 3.2.1: Graphical representation of Example 3.2.1. The horizontal axis represents the adversarial decision, while the adversarial objective function is over the vertical axis. The colored affine functions each represent a second-stage decision that restrains this value. The bold black line shows the value of the recourse function with respect to the decision of the adversary.

set of possible recourse actions aside from constraints imposed by the definition of the problem, whereas approximate approaches restrict possible recourse actions to either functions of the uncertain parameters or a preselected subset. On the other hand, all of the exact approaches in the literature consider two-stage models whereas approximations may extend to multiple stages.

Most approximate solution methods for adjustable robust optimization restrict the set of recourse solutions to more or less simple functions of uncertain parameters. These are referred to, in general, as "decision rules". In [Ben-Tal *et al.* 2004], authors study the complexity of the adjustable robust optimization problem with continuous recourse and propose a linear decision rule that they coin *affine-adjustability*. In affine adjustability, continuous recourse decisions are expressed as affine functions of uncertain parameters where the parameters of this affine function are to be optimized. The authors prove that, when the recourse matrix is fixed, if the uncertainty set is tractable then the affinely adjustable robust optimization is tractable. More elaborate decision rule schemes have been explored in the context of robust optimization as well as stochastic and distributionally robust optimization. Some examples of this literature include but are not limited to, [Chen *et al.* 2008] where deflected and segregated linear decision rules are considered, [Chen & Zhang 2009] where authors propose affine adaptability defined over an extended uncertainty set (extended affine adjustability), and [Goh & Sim 2010] where extended uncertainty sets are again used with piecewise linear decision rules (termed bi-deflected linear decision rules). In [Kuhn *et al.* 2011], authors apply linear decisions rules both in

the primal and dual spaces to evaluate the optimality gap resulting from using linear decision rules. In [Georghiou *et al.* 2015] this idea is further generalized to be applied in an extended probability space, encompassing piecewise linear, segregated, and nonlinear decision rules in the original space by a choice of the lifting operator while providing an a posteriori measure of the optimality gap resulting from using decision rules.

Although linear decision rules have desirable properties both from a theoretical and numerical perspective, their application is limited to adaptive problems with continuous recourse. Decision rules have therefore been extended to be able to incorporate binary and integer recourse decisions. In [Vayanos *et al.* 2011], authors develop a conservative approximation for multistage robust MILPs presented in the context of information discovery in multistage stochastic programming. They partition the uncertainty set into hyperrectangles and restrict the continuous and binary recourse decisions to piecewise affine and constant functions of the uncertain parameter over each hyperrectangle, respectively. The resulting conservative approximation can be formulated as an MILP; see also [Gorissen *et al.* 2015]. In [Bertsimas & Georghiou 2015], authors propose a piecewise constant decision rule for binary recourse variables used in conjunction with a scenario generation scheme resulting in an endogenous design of the decision rule. In [Bertsimas & Georghiou 2018], piecewise constant functions are again used to describe linearly parameterized binary decision rules, the reformulated problem is mapped to an extended probability space to obtain a deterministic equivalent formulation through linear programming duality. Although the generic reformulation does not scale polynomially in problem data, instances where this is the case are presented.

Another line of recent research in the approximate solution methods literature is based on the idea of restricting the recourse to a preselected set of policies termed $K-$*adaptability* or *finite adaptability*. The $K-$adaptability problem consists of selecting a first-stage solution along with K recourse solutions at the first stage. In the second stage, after the realization of uncertainty, the recourse problem reduces to selecting the best solution among these K solutions. As such, this approach is a restriction of the original two-stage problem where the flexibility of actions that can be taken at the second stage is reduced. We remark however that this method becomes exact for binary problems with only objective function uncertainty when K is sufficiently large. The authors of [Hanasusanto *et al.* 2015] extend the idea of finite adaptability ($K = 2$) considered in [Bertsimas & Caramanis 2010], for two-stage robust optimization with pure binary recourse under objective function and right-hand-side uncertainty. They propose a direct solution as a mixed integer program after reformulation for fixed K. In [Subramanyam *et al.* 2020], the concept of K-Adaptability is extended to problems with mixed-integer first- and second-stage feasible regions as well as uncertainty affected technology and recourse matrices. The authors propose a semi-infinite disjunctive programming formulation that imposes that at least one of the K recourse solutions be feasible for every realization of uncertainty. They then propose a branch-and-bound algorithm combining ideas from semi-infinite and disjunctive programming. In [Buchheim & Kurtz 2017], au-

thors study K-adaptability for combinatorial optimization problems in the special case where there are no first-stage decisions, coined "min-max-min". For the same type of problems, [Chassein *et al.* 2019] propose faster algorithms, when $K = 2$ and the Bertsimas-Sim budgeted uncertainty set is used. While in these papers, a partition of the uncertainty set and an assignment of K recourse policies to subsets defined by this partition is sought concurrently, it is also possible to define the finite adaptability problem for a given partition of the uncertainty set. This partition is then iteratively improved using the information from the solution obtained. This idea is explored in [Bertsimas & Dunning 2016] and [Postek & Hertog 2016] in the context of multi-stage adjustable robust mixed-integer optimization.

Most of the exact approaches developed in the literature concern two-stage adjustable robust optimization with continuous recourse and right-hand-side uncertainty. In this case an epigraph formulation can be defined through Benders' type cuts obtained based on the linear programming dual where the subproblem used to identify violated cuts is bilinear. Decomposition-based approaches have been used in [Jiang *et al.* 2014], [Bertsimas *et al.* 2013a], and [Zhao & Zeng 2012b] in the context of the well-known unit commitment problem under demand/wind uncertainty. They are presented in a generic framework based on Kelley's cutting plane algorithm in [Thiele *et al.* 2009]. In [Zhao & Zeng 2012b], authors additionally propose cutting planes expressed in the space of primal variables. They later develop this idea in [Zhao & Zeng 2012a], in a more general context and coin their methodology "constraint-and-column generation". This approach consists of adding a set of recourse variables and all the associated recourse constraints to the master problem for an identified uncertainty realization that is violated by the current restricted master. The subproblem in this case is a bilevel programming problem. This idea is further explored in [Ayoub & Poss 2016], which presents a mixed integer programming reformulation for the subproblem based on a Farkas system. Their reformulation is valid in the case where the uncertainty set can be represented as a projection of a binary set, the most important example being the Bertsimas-Sim budgeted uncertainty set. Finally, in [Atamtürk & Zhang 2007], authors consider a network design problem under demand uncertainty where the flow decisions on certain arcs can be delayed until after the realization of uncertainty. As such, the recourse problem is a network flow problem with right-hand-side uncertainty. The authors give a projection of the recourse polyhedron to the space of the first-stage variables through an exponential family of inequalities. They propose a cutting-plane algorithm for the solution of this model and show that the separation problem is NP-hard except for some special cases. Ideas based on projection are explored in a more generic framework in [Zhen *et al.* 2018] who show that two-stage robust optimization problems with continuous and fixed recourse can be cast as static problems via Fourier-Motzkin elimination.

There are few studies that consider exact approaches for two-stage adjustable robust optimization with integer recourse in the literature. In [Zhao *et al.* 2013], the ideas presented in [Zhao & Zeng 2012a] are extended to the mixed-integer recourse case where the authors propose a nested constraint-and-column generation scheme. Unfortunately, approaches based on on-the-fly generation of uncertainty realizations

are no longer finitely convergent when the uncertainty interferes in the objective function or the recourse matrix, as optimal solutions are not necessarily extreme points of the uncertainty set (see Figure 3.2.1). Further, their application seems to be restricted to the case where the optimal solution can be defined by adding a very small number of cuts. Independently of this work, [Kämmerling & Kurtz 2020] proposed an oracle-based solution method for two-stage robust binary optimization problems with objective uncertainty. We provide a brief qualitative comparison of their approach to the approaches presented in this here in Section 3.2.2.1.

As highlighted by the above review of the existing literature in the domain, there is a need for further research into exact solution methodologies for two-stage robust optimization problems with integer recourse. In this section, we consider two-stage robust optimization problems of the form

$$\min_{\boldsymbol{x}\in\mathcal{X}} \ \boldsymbol{c}^\top\boldsymbol{x} + \max_{\boldsymbol{\xi}\in\Xi} \min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} \ (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y} \tag{3.2.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^N_+$, $\mathcal{Y} \subseteq \mathbb{R}^M_+$ are bounded mixed binary sets, and $\Xi \subseteq \mathbb{R}^S$ is a polyhedral set with $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{f}, \boldsymbol{Q}, \boldsymbol{y}$ of conforming dimensions. We assume throughout the section that $\boldsymbol{x} = (x_1, \dots, x_{N_1}, \dots, x_N)^\top$. We denote $\boldsymbol{x}_1 = (x_1, \dots, x_{N_1})^\top$ with $\boldsymbol{x}_1 \in \{0,1\}^{N_1}$, and consider $\mathcal{Y}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathcal{Y} \mid \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1\}$. Therefore, the problems we consider in this section are two-stage robust mixed-binary optimization problems with objective uncertainty.

Remark 3.2.1. *We remark that both $\mathcal{X}$ and $\mathcal{Y}$ are mixed binary sets. However, the linking constraints in $\mathcal{Y}(\boldsymbol{x})$ involve only binary variables from the first-stage feasibility set.*

3.2.2 Methodological development

In this section, we present the main results underlying our solution approach. We first present a result that allows us to write (3.2.1) as an equivalent deterministic problem. This equivalent formulation is based on the convexification of the recourse feasible region $\mathcal{Y}(\boldsymbol{x})$ for a given first-stage solution $\boldsymbol{x} \in \mathcal{X}$.

Proposition 3.2.1. *Problem (3.2.1) is equivalent to*

$$\min_{\boldsymbol{x}\in\mathcal{X}, \boldsymbol{y}\in\text{conv}(\mathcal{Y}(\boldsymbol{x}))} \ \boldsymbol{c}^\top\boldsymbol{x} + \max_{\boldsymbol{\xi}\in\Xi} \ (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y} \tag{3.2.2}$$

Proof. For a given first-stage solution $\boldsymbol{x} \in \mathcal{X}$, and an uncertainty realization $\boldsymbol{\xi} \in \Xi$, we have that

$$\min_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} \ (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y} \ = \min_{\boldsymbol{y}\in\text{conv}(\mathcal{Y}(\boldsymbol{x}))} \ (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y} \tag{3.2.3}$$

as the inner minimization problem is a mixed integer linear programming problem. This allows for exchanging the order of optimization in $\max_{\boldsymbol{\xi}\in\Xi} \min_{\boldsymbol{y}\in\text{conv}(\mathcal{Y}(\boldsymbol{x}))} \ (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y}$ using the well-known minimax theorem (see [Neumann 1928]), as $(\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y}$ is convex in $\boldsymbol{y}$ and concave in $\boldsymbol{\xi}$, and the sets Ξ and $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ are convex by definition. □

Proposition 3.2.1 has two important implications:

(i) When $\text{conv}(\mathcal{Y}(\boldsymbol{x})) = \mathcal{Y}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}$, adaptability has no effect, *i.e.*, the two-stage problem has the same optimal solution as the static robust problem. This is, for instance, the case when the recourse problem is a linear program or is an integer program with a totally unimodular constraint matrix (e.g. network flow problems).

(ii) If one can express the conditions $\boldsymbol{y} \in \text{conv}(\mathcal{Y}(\boldsymbol{x}))$ for $\boldsymbol{x} \in \mathcal{X}$ as mixed integer linear constraints, then a deterministic equivalent mixed integer programming formulation of (3.2.1) can be obtained.

Although point (ii) above suggests a generalized approach to reformulating problem (3.2.1) as a deterministic equivalent problem, expressing the conditions $\boldsymbol{y} \in \text{conv}(\mathcal{Y}(\boldsymbol{x}))$ for $\boldsymbol{x} \in \mathcal{X}$ remains a challenge. In general, given $\boldsymbol{x} \in \mathcal{X}$, we may express $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ using its Dantzig-Wolfe reformulation, *i.e.*, as a convex combination of its extreme points. However, as $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ is dependent on $\boldsymbol{x}$, we additionally need to impose a relationship between $\boldsymbol{x}$ and $\text{conv}\mathcal{Y}(\boldsymbol{x})$ (in other words, a recourse solution from $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ can be selected only if the first-stage solution $\boldsymbol{x}$ is selected). In Figure 3.2.2, we illustrate the resulting deterministic equivalent feasible region with two possible first-stage solutions. As should be clear from this figure, the feasible region is made up of two disjunctions, each describing a recourse polyhedron based on the first-stage solution selected.

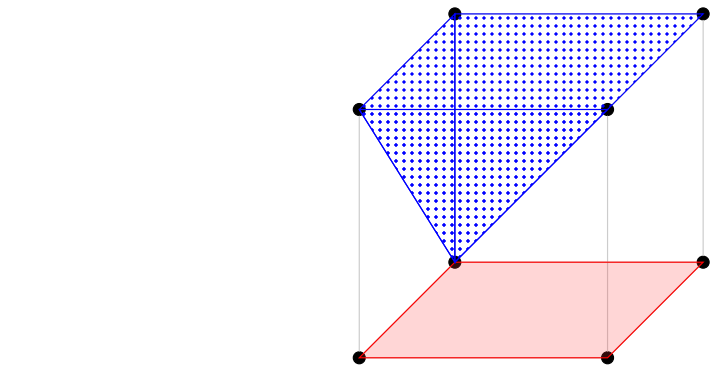


Figure 3.2.2: The feasible region of problem (3.2.1) after Dantzig-Wolfe reformulation. Here, $\mathcal{Y}(x) = \{y \in \{0,1\}^3 \mid y_2 \leq x, -y_1 + 2y_2 + y_3 \geq 3x - 2\}$, with $x \in \{0,1\}$.

Based on Figure 3.2.2, it is clear that the dependence of the set $\mathcal{Y}(\boldsymbol{x})$ on variables $\boldsymbol{x}$ is the main difficulty prohibiting the numerically efficient use of solution methods based on Dantzig-Wolfe decomposition. In Section 3.2.2.1, we present a relaxation of (3.2.2) that overcomes this difficulty. In Section 3.2.2.2, we use this relaxation to solve (3.2.1) to optimality under an assumption on the structure of the linking constraints. We also adapt the branch-and-price and branch-and-price-and-cut algorithms to this context. Finally, in Section 3.2.2.3, reposing on a reformulation

of (3.2.1) in an extended space, we generalize the results of Section 3.2.2.2 to cases where the assumption on the structure of linking constraints does not hold.

3.2.2.1 A computationally convenient relaxation

We define, for $\boldsymbol{x} \in \mathcal{X}$, the set

$$\bar{\mathcal{Y}}(\boldsymbol{x}) = \{\boldsymbol{y} \in \text{conv}(\mathcal{Y}) \mid \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1\}\,.$$

Figure 3.2.3 illustrates the sets $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ and $\bar{\mathcal{Y}}(\boldsymbol{x})$ on an example with two binary variables. As the illustration suggests, $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ is a subset of $\bar{\mathcal{Y}}(\boldsymbol{x})$. We formalize this result in the following proposition.

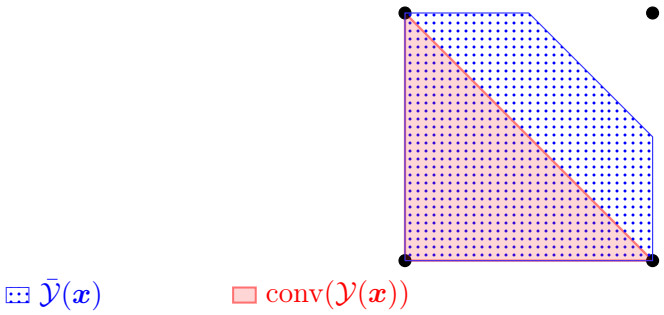


Figure 3.2.3: Sets $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ and $\bar{\mathcal{Y}}(\boldsymbol{x})$ illustrated on an example. Here, $\mathcal{Y}(x) = \{y \in \{0,1\}^2 \mid y_1 + y_2 \leq 1.5 - x\}$, $\bar{\mathcal{Y}}(x) = \{y \in [0,1]^2 \mid y_1 + y_2 \leq 1.5 - x\}$ and $x = 0$.

Proposition 3.2.2. *For $\boldsymbol{x} \in \mathcal{X}$, we have that $\text{conv}(\mathcal{Y}(\boldsymbol{x})) \subseteq \bar{\mathcal{Y}}(\boldsymbol{x})$.*

Proof. Let $\boldsymbol{y} \in \text{conv}(\mathcal{Y}(\boldsymbol{x}))$, we show that $\boldsymbol{y} \in \bar{\mathcal{Y}}(\boldsymbol{x})$. Firstly, as $\text{conv}(\mathcal{Y}(\boldsymbol{x})) \subseteq \text{conv}(\mathcal{Y})$, then it trivially holds that $\boldsymbol{y} \in \text{conv}(\mathcal{Y})$. Further, we have that $\boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1$ since $\boldsymbol{y}$ is a convex combination of points that satisfy this constraint. The result follows. □

Proposition 3.2.2 directly leads to a relaxation of problem (3.2.1) and its deterministic equivalent reformulation (3.2.2). We have:

$$\min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\text{conv}(\mathcal{Y}(\boldsymbol{x}))} \boldsymbol{c}^\top\boldsymbol{x} + \max_{\boldsymbol{\xi}\in\Xi} (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y} \;\geq\; \min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\bar{\mathcal{Y}}(\boldsymbol{x})} \boldsymbol{c}^\top\boldsymbol{x} + \max_{\boldsymbol{\xi}\in\Xi} (\boldsymbol{f}+\boldsymbol{Q}\boldsymbol{\xi})^\top\boldsymbol{y}.$$

By replacing $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ with $\bar{\mathcal{Y}}(\boldsymbol{x})$ in (3.2.2), we no longer perform the convexification operation on a set that is dependent on $\boldsymbol{x} \in \mathcal{X}$. We may therefore express the conditions $\boldsymbol{y} \in \bar{\mathcal{Y}}(\boldsymbol{x})$ by directly imposing the linking constraints $\boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1$ on $\text{conv}(\mathcal{Y})$.

To this end, let $\bar{\boldsymbol{y}}^j$ for $j \in \mathcal{L} = \{1, \ldots, L\}$ be the extreme point solutions of $\text{conv}(\mathcal{Y})$ and denote the n-dimensional simplex $\left\{\alpha \in [0,1]^n \middle| \sum_{j=1}^n \alpha^j = 1\right\}$ for $n \in \mathbb{N}$ by Δ^n. Using this notation, we have that

$$\bar{\mathcal{Y}}(\boldsymbol{x}) = \left\{ \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \boldsymbol{H}\sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1, \alpha \in \Delta^L \right\}.$$

We may therefore write the relaxation of (3.2.1) as:

$$(R): \min \quad \boldsymbol{c}^\top \boldsymbol{x} + \max_{\boldsymbol{\xi}\in\Xi} \ (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi})^\top \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.4}$$

$$\text{s.t.} \quad \boldsymbol{H} \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1 \tag{3.2.5}$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{\alpha} \in \Delta^L. \tag{3.2.6}$$

The computational advantage of relaxation (3.2.4)-(3.2.6) is clear. After reformulating the inner maximization problem in $\boldsymbol{\xi}$, this equivalent formulation can be solved using a branch-and-price algorithm, adding the columns $\bar{\boldsymbol{y}}^j \in \mathcal{Y}$ to the master problem as needed and branching when solutions $\boldsymbol{x}$ are fractional. We remark that unlike in a typical branch-and-price framework, here the variables $\boldsymbol{\alpha}$ are continuous, *i.e.*, no integrality restrictions are imposed on the reformulated variables $\boldsymbol{y}$. This is by definition of (3.2.1) where one can choose a different recourse solution $\boldsymbol{y}$ for each realization of uncertainty. For instance, for the problem of Example 3.2.1 the optimal recourse value is found as a convex combination of solutions $y_1 = 1$ and $y_2 = 1$.

An alternative relaxation building on Proposition 3.2.1 was proposed by [Kämmerling & Kurtz 2020], and used in a specialized implicit enumeration algorithm to determine optimal solutions. This relaxation can be expressed as an exponential-size LP model which is solved by a cutting plane algorithm. In this framework, each cut corresponds to a *complete* solution of the initial problem, *i.e.*, a pair of first- and second-stage solutions, and expresses its cost as a linear function of uncertain parameters. The separation problem identifies the best *complete* solution given a fixed vector of uncertain parameters. This approach can be used for non-linear optimization problems, as long as it is possible to express the cost of a solution as a linear function of the uncertain parameters. This flexibility comes at the price of having to solve subproblems in the $\mathcal{X} \times \mathcal{Y}$ space (*i.e.* optimize over $\{(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})\}$), and an ad-hoc branching scheme.

The approach presented in this section exploits the decomposition of the first and second stages that is naturally present in the structure of problem (3.2.1). As many decomposition approaches, it is well-suited when optimizing a deterministic function over sets $\mathcal{X}$ and $\mathcal{Y}$ separately is significantly easier than optimizing it over $\mathcal{X} \times \mathcal{Y}$. By reposing on the well-known paradigms of Dantzig-Wolfe decomposition and the branch-and-price algorithm, it is easier to implement in practice, especially with the increasing availability of automatic decomposition software. However, as the pricing of second-stage variables requires LP duality, naturally nonlinear formulations can be handled only after an appropriate linearization.

We conclude this section by presenting an equivalent deterministic MILP formulation of (R), that allows both a theoretical characterization of the complexity of the problem, and development of solution algorithms. To do so, we first dualize the inner maximization problem in (3.2.4)

$$\max_{\boldsymbol{\xi}\in\Xi} \ (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi})^\top \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.7}$$

which is a linear programming problem as we assume that Ξ is a polyhedral set. We write, without loss of generality, that $\Xi = \left\{\boldsymbol{\xi} \in \mathbb{R}^S \middle| \boldsymbol{A}\boldsymbol{\xi} \leq \boldsymbol{b}\right\}$ with $\boldsymbol{A} \in \mathbb{R}^{S' \times S}$ and $\boldsymbol{b} \in \mathbb{R}^{S'}$. Let $\boldsymbol{u} \in \mathbb{R}^{S'}_+$ be the dual variables associated with the constraints of the uncertainty set. We then have that

$$\max_{\boldsymbol{\xi} \in \Xi} \quad (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi})^\top \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j = \boldsymbol{f}^\top \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j + \min_{\boldsymbol{u} \in \mathbb{R}^{S'}_+} \quad \boldsymbol{u}^\top \boldsymbol{b} \tag{3.2.8}$$

$$\text{s.t.} \quad \boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{Q}^\top \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j. \tag{3.2.9}$$

Therefore the deterministic equivalent of the formulation (3.2.4)-(3.2.6) is expressed as:

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{f}^\top \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j + \boldsymbol{u}^\top \boldsymbol{b} \tag{3.2.10}$$

$$\text{s.t.} \quad \boldsymbol{H} \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1 \tag{3.2.11}$$

$$\boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{Q}^\top \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.12}$$

$$\sum_{j \in \mathcal{L}} \alpha^j = 1 \tag{3.2.13}$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{\alpha} \in \mathbb{R}^L_+, \boldsymbol{u} \in \mathbb{R}^{S'}_+. \tag{3.2.14}$$

3.2.2.2 Exact formulations based on relaxation (R)

In this section, we present two key results that enable the exact solution of problem (3.2.1) based on relaxation (R) and its deterministic equivalent model (3.2.10)-(3.2.14). The first result establishes certain cases where the relaxation (R) is exact, whereas the second result provides a family of valid inequalities that cut off infeasible solutions when this is not the case. To present them, we need the following additional assumption.

Assumption 3.2.1. *We let* $\boldsymbol{y} = (y_1, \ldots, y_{M_1}, \ldots, y_M)^\top$, *and we denote* $\boldsymbol{y}_1 = (y_1, \ldots, y_{M_1})^\top$ *with* $\boldsymbol{y}_1 \in \{0,1\}^{M_1}$. *We assume in the following that* $\mathcal{Y}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathcal{Y} \mid \boldsymbol{H}\boldsymbol{y}_1 \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1\}$.

Assumption 3.2.1 guarantees that the set of extreme points of $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ is a subset of the set of extreme points of $\text{conv}(\mathcal{Y})$ as illustrated in the following example.

Example 3.2.2. *Consider the second-stage feasibility set* $\mathcal{Y}(x) = \{\boldsymbol{y} \in \{0,1\}^2 \mid y_1 + y_2 \leq 1.5 + 0.5x\}$ *with* $x \in \{0,1\}$, *and its variant obtained by replacing the binary restrictions on* $\boldsymbol{y}$ *by* $\boldsymbol{y} \in [0,1] \times \{0,1\}$.

In Figure 3.2.4, we present the sets $\text{conv}(\mathcal{Y}(0))$ and $\bar{\mathcal{Y}}(0)$ for these two variants. As is clear from this figure, the set $\text{conv}(\mathcal{Y}(0))$ is described by the extreme points of $\text{conv}(\mathcal{Y})$ for the first variant, whereas the additional extreme point $(0.5, 1)$ is

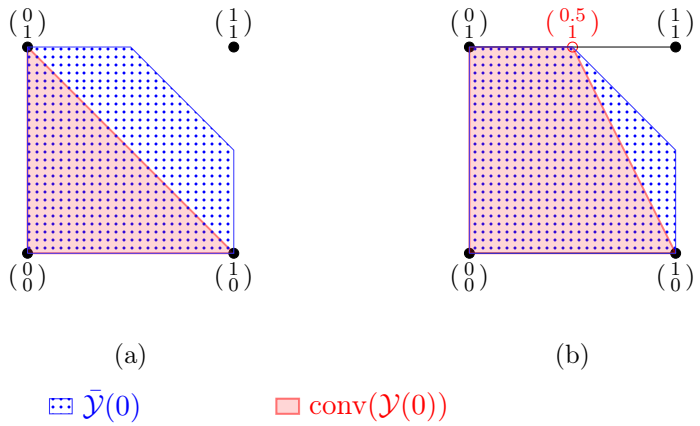


Figure 3.2.4: Illustration of Example 3.2.2 and the implications of Assumption 3.2.1. In case (a), $\mathcal{Y} = \{0,1\}^2$ and $\text{conv}(\mathcal{Y}(0)) = \text{conv}\{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}\}$, whose extreme points are a subset of the extreme points of $\text{conv}(\mathcal{Y})$. In case (b), $\mathcal{Y} = [0,1] \times \{0,1\}$ and $\text{conv}(\mathcal{Y}(0)) = \text{conv}\{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{0.5}{1}\}$. The point $\binom{0.5}{1}$ is not an extreme point of $\text{conv}(\mathcal{Y})$.

required in the description of $\text{conv}(\mathcal{Y}(0))$ for the second variant. We remark that this new extreme point is created by the intersection of the set $\mathcal{Y}$ with the linking constraint $y_1 + y_2 \leq 1.5 + x$. Intuitively, Assumption 3.2.1 guarantees that the linking constraints always intersect with a lattice free set (see e.g. [Wolsey & Nemhauser 1999]), therefore do not create additional extreme points.

Let $\boldsymbol{x}^i$ for $i \in \mathcal{K} = \{1, \ldots, K\}$ be the extreme point solutions of $\text{conv}(\mathcal{X})$. Further let $\bar{\boldsymbol{y}}^j$ for $j \in \mathcal{L} = \{1, \ldots, L\}$ be the extreme point solutions of $\text{conv}(\mathcal{Y})$ as before, and define, $\mathcal{L}_i = \{j \in \mathcal{L} \mid \boldsymbol{H}\bar{\boldsymbol{y}}_1^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1^i\}$ for $i \in \mathcal{K}$. We first present an intermediary result that allows the characterization of $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ in terms of the extreme points of the set $\text{conv}(\mathcal{Y})$.

Proposition 3.2.3. $\text{conv}(\mathcal{Y}(\boldsymbol{x}^i)) = \left\{\sum_{j\in\mathcal{L}_i} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \boldsymbol{\alpha} \in \Delta^{|\mathcal{L}_i|}\right\}$ *for* $i \in \mathcal{K}$.

Proof. Given $i \in \mathcal{K}$, let $\text{ext}(\mathcal{Y}(\boldsymbol{x}^i))$ denote the set of extreme point solutions of $\text{conv}(\mathcal{Y}(\boldsymbol{x}^i))$.

Let $\boldsymbol{y} \in \text{ext}(\mathcal{Y}(\boldsymbol{x}^i)) \subseteq \mathcal{Y}(\boldsymbol{x}^i) = \left\{\boldsymbol{y} \in \mathcal{Y} \;\middle|\; \boldsymbol{H}\boldsymbol{y}_1 \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1^i\right\}$. Since $\boldsymbol{y} \in \mathcal{Y}$, there exists $\boldsymbol{\alpha} \in \Delta^{|\mathcal{L}|}$ such that $\boldsymbol{y} = \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j$. Further, for all $r \in \mathcal{L}$ such that $\alpha^r > 0$, we must have that $\bar{\boldsymbol{y}}_1^r = \boldsymbol{y}_1$, as otherwise there exists a row index ℓ such that $\sum_{j\in\mathcal{L}} \alpha^j \left(\bar{\boldsymbol{y}}_1^j\right)_\ell \in]0,1[$. It follows that $\boldsymbol{H}\bar{\boldsymbol{y}}_1^r \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1^i$ for all $r \in \mathcal{L}$ such that $\alpha^r > 0$, and therefore $r \in \mathcal{L}_i$ following the definition of $\mathcal{L}_i$. This shows that $\text{ext}(\mathcal{Y}(\boldsymbol{x}^i)) \subseteq \left\{\sum_{j\in\mathcal{L}_i} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \boldsymbol{\alpha} \in \Delta^{|\mathcal{L}_i|}\right\}$, and as a consequence that $\text{conv}(\mathcal{Y}(\boldsymbol{x}^i)) \subseteq \left\{\sum_{j\in\mathcal{L}_i} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \boldsymbol{\alpha} \in \Delta^{|\mathcal{L}_i|}\right\}$.

Conversely, take any solution $\bar{\boldsymbol{y}}^j$ for $j \in \mathcal{L}_i$. By definition of the set $\mathcal{L}_i$, we have that $\bar{\boldsymbol{y}}^j \in \mathcal{Y}$, and that $\boldsymbol{H}\bar{\boldsymbol{y}}_1^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1^i$. It follows that $\bar{\boldsymbol{y}}^j \in \mathcal{Y}(\boldsymbol{x}^i)$ and therefore can be expressed as a convex combination of points $\boldsymbol{y} \in \text{ext}(\mathcal{Y}(\boldsymbol{x}^i))$. Therefore

$\left\{\sum_{j\in\mathcal{L}_i}\alpha^j\bar{\boldsymbol{y}}^j \;\middle|\; \boldsymbol{\alpha}\in\Delta^{|\mathcal{L}_i|}\right\}\subseteq \text{conv}(\mathcal{Y}(\boldsymbol{x}^i))$, proving the result. □

We next present a result that characterizes a sufficient condition for the equivalence between $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ and $\bar{\mathcal{Y}}(\boldsymbol{x})$ for $\boldsymbol{x}\in\mathcal{X}$.

Proposition 3.2.4. *If $\boldsymbol{H}=I$, $\boldsymbol{T}=-I$ and $\boldsymbol{d}=0$, then $\bar{\mathcal{Y}}(\boldsymbol{x})=\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ for $\boldsymbol{x}\in\mathcal{X}$.*

Proof. Under the given assumptions, $\mathcal{Y}(\boldsymbol{x})=\{\boldsymbol{y}\in\mathcal{Y}\mid \boldsymbol{y}_1\leq\boldsymbol{x}_1\}$. Assume now that there exists a solution $\boldsymbol{y}$ such that $\boldsymbol{y}=\sum_{j\in\mathcal{L}}\alpha^j\bar{\boldsymbol{y}}^j\in\bar{\mathcal{Y}}(\boldsymbol{x})$ and $\boldsymbol{y}\notin\text{conv}(\mathcal{Y}(\boldsymbol{x}))$. Then, we must have that $\sum_{j\in\mathcal{L}}\alpha^j=1$ and $\sum_{j\in\mathcal{L}|\bar{\boldsymbol{y}}^j_1\leq\boldsymbol{x}_1}\alpha^j<1$ by using the characterization of Proposition 3.2.3. This implies the existence of a linking constraint, say constraint i, and an index $k\in\mathcal{L}$ such that $\bar{y}^k_i>x_i$ and $\alpha_k>0$. Since $x_i\in\{0,1\}$ and $\bar{y}^k_i\in\{0,1\}$, we must have that $x_i=0$ and $\bar{y}^k_i=1$. We may therefore write,

$$y_i=\sum_{j\in\mathcal{L}}\bar{y}^j_i\alpha^j\geq\alpha_k\bar{y}^k_i=\alpha_k>0=x_i$$

Thus $\boldsymbol{y}\notin\mathcal{Y}(\boldsymbol{x})$, which contradicts the assumption that $\boldsymbol{y}\in\bar{\mathcal{Y}}(\boldsymbol{x})$. Therefore, we have proved $\bar{\mathcal{Y}}(\boldsymbol{x})\subseteq\text{conv}(\mathcal{Y}(\boldsymbol{x}))$. We additionally have that $\text{conv}(\mathcal{Y}(\boldsymbol{x}))\subseteq\bar{\mathcal{Y}}(\boldsymbol{x})$ by Proposition 3.2.2. As a result $\bar{\mathcal{Y}}(\boldsymbol{x})=\text{conv}(\mathcal{Y}(\boldsymbol{x}))$. □

Example 3.2.3. *Consider the second-stage feasibility set*

$$\mathcal{Y}(\boldsymbol{x})=\left\{\boldsymbol{y}\in\{0,1\}^2\mid y_i\leq x_i\quad\forall i=1,2\right\}$$

with $\mathcal{X}=\{0,1\}^2$. Let $\mathcal{L}=\{1,\ldots,4\}$ and $\bar{\boldsymbol{y}}^1=(1,0)$, $\bar{\boldsymbol{y}}^2=(0,1)$, $\bar{\boldsymbol{y}}^3=(0,0)$, and $\bar{\boldsymbol{y}}^4=(1,1)$. Associating $\alpha^j\geq0$ with $\bar{\boldsymbol{y}}^j$ for $j\in\mathcal{L}$, it is easy to confirm that $\bar{\mathcal{Y}}(\boldsymbol{x})=\left\{\sum_{j\in\mathcal{L}}\alpha^j\bar{\boldsymbol{y}}^j\middle|\alpha^1+\alpha^4\leq x_1,\alpha^2+\alpha^4\leq x_2,\boldsymbol{\alpha}\in\Delta^4\right\}$. Further, by Proposition 3.2.3, we have that $\text{conv}(\mathcal{Y}(\boldsymbol{x}))=\left\{\sum_{j\in\mathcal{L}}\bar{\boldsymbol{y}}^j\alpha^j\middle|\sum_{j\in\mathcal{L}\mid\bar{y}^j_i\leq x_i,\forall i=1,2}\alpha^j=1,\boldsymbol{\alpha}\in\Delta^4\right\}$. It can easily be verified that $\text{conv}(\mathcal{Y}(\boldsymbol{x}))=\bar{\mathcal{Y}}(\boldsymbol{x})$ for $\boldsymbol{x}\in\mathcal{X}$. This equivalence is a direct consequence of Proposition 3.2.4.

Problems of form (3.2.1) that satisfy Assumption 3.2.1 and have the structure presented in Proposition 3.2.4, can be solved using the deterministic equivalent model (3.2.10)-(3.2.14) based on relaxation (R), which, in this case, is exact. These include, by a substitution of variables, problems with linking constraints of type $\boldsymbol{y}_1\geq\boldsymbol{x}_1$, $\boldsymbol{x}_1+\boldsymbol{y}_1\leq\mathbf{1}$, $\boldsymbol{x}_1+\boldsymbol{y}_1\geq\mathbf{1}$ and $\mathbf{1}^\top\boldsymbol{y}_1\leq x$ that cover a wide range of applications. As the number of extreme points of the set $\mathcal{Y}$ are in general prohibitively large, we propose to solve this relaxation using the branch-and-price algorithm generating columns from the set $\mathcal{Y}$ and branching when $\boldsymbol{x}_1$ is fractional.

We next outline the main components of this branch-and-price algorithm starting with the restricted master problem. The complete branch-and-price scheme is presented in Algorithm 3.2.1. Let us recall that $\bar{\boldsymbol{y}}^j$ for $j\in\mathcal{L}=\{1,\ldots,L\}$ are the extreme point solutions of $\text{conv}(\mathcal{Y})$, and let $\mathcal{L}^{\text{R}}\subset\mathcal{L}$. Further, assume without loss

of generality that $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^{N_1} \times \mathbb{R}^{N-N_1} \mid \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{g}\}$. The restricted master problem is then written as:

$$(MP(\mathcal{L}^{\mathrm{R}})) \ : \ \min \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{f}^\top \sum_{j \in \mathcal{L}^{\mathrm{R}}} \alpha^j \bar{\boldsymbol{y}}^j + \boldsymbol{u}^\top \boldsymbol{b} \tag{3.2.15}$$

$$\text{s.t.} \quad \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{g} \tag{3.2.16}$$

$$\boldsymbol{H} \sum_{j \in \mathcal{L}^{\mathrm{R}}} \alpha^j \bar{\boldsymbol{y}}^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1 \tag{3.2.17}$$

$$\boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{Q}^\top \sum_{j \in \mathcal{L}^{\mathrm{R}}} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.18}$$

$$\sum_{j \in \mathcal{L}^{\mathrm{R}}} \alpha^j = 1 \tag{3.2.19}$$

$$\boldsymbol{x} \in [0,1]^{N_1} \times \mathbb{R}^{N-N_1}, \boldsymbol{\alpha} \in \mathbb{R}_+^{|\mathcal{L}^{\mathrm{R}}|}, \boldsymbol{u} \in \mathbb{R}_+^{S'}. \tag{3.2.20}$$

Let $\boldsymbol{\pi}^*$, $\boldsymbol{\mu}^*$, and λ^* be the optimal values of the dual variables associated with the constraints (3.2.17), (3.2.18), and (3.2.19), respectively. Then the pricing problem takes the form:

$$(Pricing(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \lambda^*)) \ : \ \min_{\boldsymbol{y} \in \mathcal{Y}} \ -\lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \boldsymbol{y} \tag{3.2.21}$$

Remark 3.2.2. *The pricing problem* (3.2.21) *is free of the first-stage variables* $\boldsymbol{x}$.

Once the restricted master problem $MP(\mathcal{L}^{\mathrm{R}})$ is solved to optimality, generating columns $\bar{\boldsymbol{y}}^j$ for $j \in \mathcal{L} \setminus \mathcal{L}^{\mathrm{R}}$ as needed, yielding the optimal relaxation solution $(\boldsymbol{x}^*, \boldsymbol{\alpha}^*)$, one typically needs to branch in order to obtain integer solutions. As the integrality of variables $\boldsymbol{y}$ is not required in this case, we branch only on fractional $\boldsymbol{x}_1$ variables. Let $i \in \{1, \ldots, N_1\}$ such that $\boldsymbol{x}_i^* \in]0,1[$. The branching constraints $\boldsymbol{x}_i^* \leq 0$ and $\boldsymbol{x}_i^* \geq 1$ are added to the restricted master problem, for the left and right children of the current node, respectively. These constraints do not affect the pricing problem.

We now turn our attention to the case where $\mathrm{conv}(\mathcal{Y}(\boldsymbol{x})) \neq \bar{\mathcal{Y}}(\boldsymbol{x})$ for some $\boldsymbol{x} \in \mathcal{X}$. We motivate our main result with the following example.

Example 3.2.4. *Consider the second-stage feasibility set*

$$\mathcal{Y}(\boldsymbol{x}) = \{\boldsymbol{y} \in \{0,1\}^2 \mid \mathbf{1}^\top \boldsymbol{y} \leq 1, x_1 + 2x_2 + y_1 + y_2 \geq 1.9\}$$

with $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^2 \mid \mathbf{1}^\top \boldsymbol{x} \leq 1\}$. *Let* $\mathcal{L} = \{1, \ldots, 3\}$ *and* $\bar{\boldsymbol{y}}^1 = (1,0)$, $\bar{\boldsymbol{y}}^2 = (0,1)$ *and* $\bar{\boldsymbol{y}}^3 = (0,0)$. *Let us consider* $\boldsymbol{x}^* = (1,0)$ *and associate* $\alpha^j \geq 0$ *with* $\bar{\boldsymbol{y}}^j$ *for* $j \in \mathcal{L}$.

We have that $\bar{\mathcal{Y}}(\boldsymbol{x}^*) = \left\{ \sum_{j \in \mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \alpha^1 + \alpha^2 \geq 0.9, \boldsymbol{\alpha} \in \Delta^3 \right\}$. *Further we have,*

by Proposition 3.2.3, that

$$\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}^*)) = \left\{ \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \sum_{j\in\mathcal{L}|\bar{y}_1^j+\bar{y}_2^j\geq 0.9} \alpha^j = 1, \alpha \in \Delta^3 \right\}$$
$$= \left\{ \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \;\middle|\; \alpha^1 + \alpha^2 = 1, \alpha \in \Delta^3 \right\}.$$

It is clear that $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}^*)) \neq \bar{\mathcal{Y}}(\boldsymbol{x}^*)$. *More specifically, in* $\bar{\mathcal{Y}}(\boldsymbol{x}^*)$, *one can use the extreme point* $\bar{\boldsymbol{y}}^3 = (0,0) \notin \mathcal{Y}(\boldsymbol{x}^*)$ *with* $\alpha^3 \leq 0.1$, *whereas* $\alpha^3 = 0$ *in* $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}^*))$. *To establish an equivalence between* $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$ *and* $\bar{\mathcal{Y}}(\boldsymbol{x})$, *one needs to forbid the use of extreme point* $\bar{\boldsymbol{y}}^3$ *in* $\bar{\mathcal{Y}}(\boldsymbol{x}^*)$. *This is achieved by adding the inequality* $\alpha_3 \leq 1 - x_1 + x_2$ *to* $\bar{\mathcal{Y}}(\boldsymbol{x})$. *It can be verified that, with the addition of this inequality,* $\bar{\mathcal{Y}}(\boldsymbol{x}) = \operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$ *for* $\boldsymbol{x} \in \mathcal{X}$.

In the following proposition, we generalize the inequality we have introduced in Example 3.2.4 to no-good cut type inequalities (see *e.g.* [Hooker 1994]). To this end, let $\mathcal{N} = \{1, \ldots, N_1\}$, and define $\mathcal{I}(\boldsymbol{x}) = \{i \in \mathcal{N} \mid x_i = 1\}$ for $\boldsymbol{x} \in \mathcal{X}$. Further, let for $\mathcal{I} \subseteq \mathcal{N}$, $\mathcal{L}(\mathcal{I}) = \left\{ j \in \mathcal{L} \;\middle|\; \boldsymbol{H}\bar{\boldsymbol{y}}_1^j \leq \boldsymbol{d} - \boldsymbol{T}\sum_{i\in\mathcal{I}} \boldsymbol{e}_i \right\}$ where $\boldsymbol{e}_i$ is the i^{th} unit vector of conforming dimensions.

Proposition 3.2.5. *The inequalities*

$$\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I})} \alpha^j \leq |\mathcal{I}| - \sum_{i\in\mathcal{I}} x_i + \sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i \qquad \forall \mathcal{I} \subseteq \mathcal{N} \tag{3.2.22}$$

are valid for $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x})) = \left\{ \sum_{j\in\mathcal{L}(\boldsymbol{x})} \alpha^j \bar{\boldsymbol{y}}^j \mid \boldsymbol{\alpha} \in \Delta^{|\mathcal{L}(\boldsymbol{x})|} \right\}$.

Proof. Given $\boldsymbol{x} \in \mathcal{X}$, we show that, if $\boldsymbol{\alpha} \in \Delta^{|\mathcal{L}|}$ and $\sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \in \operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$ then it satisfies inequalities (3.2.22). In this case, by definition of $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$, we must have that $\alpha^j = 0$ for all $j \in \mathcal{L} \setminus \mathcal{L}(\mathcal{I}(\boldsymbol{x}))$. Therefore, if $\mathcal{I}(\boldsymbol{x}) = \mathcal{I}$ then we have that $0 \leq 0$. Otherwise, we have that either $\sum_{i\in\mathcal{I}} x_i < |\mathcal{I}|$ or $\sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i > 0$. Therefore, we have on the left-hand-side $\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I})} \alpha^j$ which is not greater than 1, and on the right-hand-side $|\mathcal{I}| - \sum_{i\in\mathcal{I}} x_i + \sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i > 0$, which is greater than 1 by integrality of the terms involved, for all $\mathcal{I} \subseteq \mathcal{N}$, $\mathcal{I} \neq \mathcal{I}(\boldsymbol{x})$. □

Proposition 3.2.6. *Let* $\boldsymbol{\alpha} \in \Delta^{|\mathcal{L}|}$ *such that* $\sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \in \bar{\mathcal{Y}}(\boldsymbol{x})$ *and* $\alpha^j > 0$ *for some* $j \in \mathcal{L} \setminus \mathcal{L}(\mathcal{I}(\boldsymbol{x}))$, *then there exists an inequality of form* (3.2.22) *that is violated.*

Proof. Let, in this case, $\mathcal{I} = \mathcal{I}(\boldsymbol{x})$. We have on the left-hand-side $\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I}(\boldsymbol{x}))} \alpha^j > 0$ and on the right-hand-side $|\mathcal{I}(\boldsymbol{x})| - \sum_{i\in\mathcal{I}(\boldsymbol{x})} x_i + \sum_{i\notin\mathcal{I}(\boldsymbol{x})} x_i = 0$. Therefore, inequality (3.2.22) with $\mathcal{I} = \mathcal{I}(\boldsymbol{x})$ is violated. □

Propositions 3.2.5 shows that inequalities (3.2.22) are valid for $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$, and Proposition 3.2.6 establishes that they are sufficient to describe $\operatorname{conv}(\mathcal{Y}(\boldsymbol{x}))$ for $\boldsymbol{x} \in$

$\mathcal{X}$. As a result we may write an equivalent formulation for (3.2.2) as follows:

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{f}^\top \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j + \boldsymbol{u}^\top \boldsymbol{b} \tag{3.2.23}$$

$$\text{s.t.} \quad \boldsymbol{H} \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}_1^j \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1 \tag{3.2.24}$$

$$\boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{Q}^\top \sum_{j\in\mathcal{L}} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.25}$$

$$\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I})} \alpha^j \leq |\mathcal{I}| - \sum_{i\in\mathcal{I}} x_i + \sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i \qquad \forall \mathcal{I} \subseteq \mathcal{N} \tag{3.2.22}$$

$$\sum_{j\in\mathcal{L}} \alpha^j = 1 \tag{3.2.26}$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{\alpha} \in \Delta^L, \boldsymbol{u} \in \mathbb{R}_+^{S'}. \tag{3.2.27}$$

As this formulation has an exponential number of variables and constraints, we propose to couple column generation with cut generation within a branch-and-price-and-cut algorithm in its solution, which is described by Algorithm 3.2.1. We next outline the main changes to the branch-and-price algorithm proposed earlier. An initial restricted master problem and the associated pricing problem are given by (3.2.15)-(3.2.20) and (3.2.21), respectively, assuming that $\mathcal{L}^{\mathrm{R}}$ is used as the initial set of columns and no cuts are added. Compared to our previous algorithm, the identification of violated cuts (3.2.22), and the changes to the pricing problem resulting from their addition need to be addressed.

We first discuss the identification of violated cuts of type (3.2.22) given a candidate incumbent solution $(\boldsymbol{x}^*, \boldsymbol{\alpha}^*)$ with $\boldsymbol{x}_1^* \in \{0,1\}^{N_1}$. In this case, it suffices to verify whether or not inequality (3.2.22) with $\mathcal{I} = \mathcal{I}(\boldsymbol{x}^*)$ is satisfied as the proof of Proposition 3.2.6 suggests. If it is satisfied then $(\boldsymbol{x}^*, \boldsymbol{\alpha}^*)$ is accepted as an incumbent solution and the current upper bound is updated. Otherwise, the cut $\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I}(\boldsymbol{x}^*))} \alpha^j \leq |\mathcal{I}(\boldsymbol{x}^*)| - \sum_{i\in\mathcal{I}(\boldsymbol{x}^*)} x_i + \sum_{i\in\mathcal{N}\setminus\mathcal{I}(\boldsymbol{x}^*)} x_i$ is added to the current restricted master. We remark that, identifying columns that are not feasible for a given first-stage solution $\boldsymbol{x}$ ($j \in \mathcal{L} \setminus \mathcal{L}(\mathcal{I}(\boldsymbol{x}))$) is not complicated, as it suffices to verify whether or not $\boldsymbol{H}\bar{\boldsymbol{y}}_1^j \leq \boldsymbol{d} - \boldsymbol{T}\sum_{i\in\mathcal{I}(\boldsymbol{x})} \boldsymbol{e}_i$ for current columns in the restricted master.

On the other hand, handling the changes induced by the introduction of cuts (3.2.22) in the pricing problem is more computationally demanding. Consider a subset of cuts (3.2.22) added to the restricted master $MP(\mathcal{L}^{\mathrm{R}})$:

$$\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I})} \alpha^j \leq |\mathcal{I}| - \sum_{i\in\mathcal{I}} x_i + \sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i \qquad \forall \mathcal{I} \in \mathcal{N}^{\mathrm{R}}. \tag{3.2.28}$$

Let $\eta_{\mathcal{I}}^*$ be the optimal value of the dual variable associated to $\mathcal{I} \in \mathcal{N}^{\mathrm{R}}$. The pricing problem (3.2.21) takes the form

$$\min_{j\in\mathcal{L}} \; - \sum_{\mathcal{I}\in\mathcal{N}^{\mathrm{R}} | j\notin\mathcal{L}(\mathcal{I})} \eta_{\mathcal{I}}^* - \lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \bar{\boldsymbol{y}}^j. \tag{3.2.29}$$

Casting this problem as a mixed integer linear optimization problem over $\mathcal{Y}$ raises the issue of linearizing the first term in the objective function, which accounts for the dual values corresponding to added cuts (3.2.22) in which the new column will be involved. To this end, consider the indicator variable $z_{\mathcal{I}}$ for $\mathcal{I} \in \mathcal{N}^{\mathrm{R}}$ that takes value 1 if and only if $\bar{\boldsymbol{y}}^j \in \mathcal{L} \setminus \mathcal{L}(\mathcal{I})$, *i.e.*, $\boldsymbol{H}\bar{\boldsymbol{y}}_1^j + \boldsymbol{T}\sum_{i\in\mathcal{I}} \boldsymbol{e}_i - \boldsymbol{d} > 0$. The updated pricing problem then takes the form

$$(Pricing'(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \lambda^*, \boldsymbol{\eta}^*)) : \min \quad -\sum_{\mathcal{I}\in\mathcal{N}^{\mathrm{R}}} \eta^*_{\mathcal{I}} z_{\mathcal{I}} - \lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \boldsymbol{y} \tag{3.2.30}$$

$$\text{s.t.} \quad M z_{\mathcal{I}} \geq \boldsymbol{H}\bar{\boldsymbol{y}}_1^j + \boldsymbol{T}\sum_{i\in\mathcal{I}} \boldsymbol{e}_i - \boldsymbol{d} \qquad \forall \mathcal{I} \in \mathcal{N}^{\mathrm{R}} \tag{3.2.31}$$

$$\boldsymbol{y} \in \mathcal{Y}, z_{\mathcal{I}} \in \{0,1\}^{|\mathcal{N}^{\mathrm{R}}|} \tag{3.2.32}$$

where M is a sufficiently large constant.

Branch-and-price-and-cut algorithm Algorithm 3.2.1 summarizes the branch-and-price-and-cut procedure proposed to solve problem (3.2.1) through its reformulation (3.2.23)-(3.2.27). Line 1 initializes the set of columns, $\mathcal{L}^R$, so that the restricted master problem is feasible. To do so, one can use any feasible solution of (3.2.1). In the relatively rare applications where no trivial feasible solutions exist, one can solve the deterministic counterpart of the problem obtained by fixing an arbitrary scenario. Alternatively, the phase 1 simplex algorithm is used in that purpose for column generation approaches in more general contexts. The best primal bound found, *PrimalBound*, the best feasible solution found, S^*, and the subset of no-good cuts (3.2.22), $\mathcal{N}^R$, are initialized in Line 2. Each node is encoded as the set of branching constraints, $\mathcal{B}$, defining the set of solutions of that node. The list of open nodes, $\mathcal{Q}$, is thus initialized in Line 2 with the root node, that has no branching constraints. Loop 3-19 processes the open nodes. The solution of the relaxation at the current node is computed in Line 5. If the solution satisfies the integrality requirements (Line 10), we check whether it satisfies constraints (3.2.22) (Line 11-12). Note that this is always the case when Assumption 3.2.1 and the conditions of Proposition 3.2.4 hold, so that Lines 11-16 can be replaced by line 13 only, where *PrimalBound* and S^* are updated. If the current second-stage solution $\boldsymbol{y}^*$ is not compatible with the current first-stage solution $\boldsymbol{x}^*$, *i.e.*, $\boldsymbol{y}^* = \sum_{j\in\mathcal{L}} \boldsymbol{\alpha}^{*j}\bar{\boldsymbol{y}}^j \notin \mathrm{conv}\mathcal{Y}(\boldsymbol{x}^*)$, Line 15 adds the constraint excluding this solution. In this case, the node is put back in the list of open nodes in Line 16. When $\boldsymbol{x}_1^*$ is not integer, branching is performed in Lines 18 and 19.

Algorithm 3.2.2 depicts the column generation procedure used to compute the relaxation at each node of the search tree in Line 5 of Algorithm 3.2.1. The loop 1-8 adds new columns to the restricted master $MP(\mathcal{L}^R)$ until no negative reduced cost column is found. Model $MP(\mathcal{L}^R)$ is solved in Line 2, providing optimal dual variables that are used as input to the pricing problem in Line 4. Lines 6-7 add a

Algorithm 3.2.1: Branch-and-price-and-cut algorithm for solving problem (3.2.22)-(3.2.27). In the special case where Assumption 3.2.1 as well as Proposition 3.2.4 hold, the test in line 12 is always true and lines 11-16 can be replaced by line 13 only.

1 Choose $\mathcal{L}^R$ and $(\bar{y}^j)_{j\in\mathcal{L}^R}$ such that (3.2.15)-(3.2.20) is feasible
2 $PrimalBound \leftarrow \infty$, $S^* \leftarrow \emptyset$, $\mathcal{N}^R \leftarrow \emptyset$, $\mathcal{Q} \leftarrow \{\emptyset\}$
3 **while** $\mathcal{Q} \neq \emptyset$ **do**
4 Pop a node/set of branching constraints $\mathcal{B}$ from $\mathcal{Q}$
5 $(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{\alpha}^*) \leftarrow optimizeRelaxation(\mathcal{B}, \mathcal{L}^R, \mathcal{N}^R)$
6 $DualBound \leftarrow \boldsymbol{c}^\top \boldsymbol{x}^* + \boldsymbol{f}^\top \sum_{j\in\mathcal{L}} \boldsymbol{\alpha}^{*j} \bar{\boldsymbol{y}}^j + \boldsymbol{u}^{*\mathrm{T}} \boldsymbol{b}$
7 **if** $DualBound \geq PrimalBound$ **then**
8 *Current node is pruned by bound*
9 **else**
10 **if** $\boldsymbol{x}_1^* \in \{0,1\}^{N_1}$ **then**
11 $\mathcal{I} \leftarrow \{i \in \{1,\dots,N_1\} : x_i^* = 1\}$
12 **if** $\sum_{j\in\mathcal{L}\setminus\mathcal{L}(\mathcal{I})} \alpha^{*j} \leq |\mathcal{I}| - \sum_{i\in\mathcal{I}} x_i^* + \sum_{i\in\mathcal{N}\setminus\mathcal{I}} x_i^*$ **then**
13 Update $PrimalBound$ and S^* with $DualBound$ and $(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{\alpha}^*)$
14 **else**
15 Add the no good cut $\mathcal{N}^R \leftarrow \mathcal{N}^R \cup \{\mathcal{I}\}$
16 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{B}$
17 **else**
18 Choose $i \in \{1,\dots,N_1\}$ such that $x_i^* \in]0,1[$
19 Add two nodes $\mathcal{B}_0 = \mathcal{B} \cup \{x_i = 0\}$ and $\mathcal{B}_1 = \mathcal{B} \cup \{x_i = 1\}$ to $\mathcal{Q}$
20 **return** S^*, an optimal solution of (R)

new column to $MP(\mathcal{L}^R)$ if the pricing problem returns a column with a negative reduced cost. When Assumption 3.2.1 as well as Proposition 3.2.4 hold there is no need for Constraints (3.2.28), so that the pricing problem in Line 4 can be replaced with $(Pricing(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \lambda^*))$. The left-hand-side of the tests in Lines 6 and 8 takes the simpler form $-\lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \boldsymbol{y}^*$.

We conclude this section by comparing the branch-and-price-and-cut approach to the methodologies presented under the name constraint-and-column generation in the literature (see [Zhao *et al.* 2013]). In constraint-and-column generation, a deterministic equivalent formulation inspired by stochastic programming is dynamically constructed. In this scheme, a block of variables and constraints is appended to the master problem for each violated scenario. As a result, columns correspond to variables $\boldsymbol{y}^\xi$, associated to each violated scenario ξ, for which the values should be determined by the solution of the master. The constraints correspond to the set of linking constraints and constraints describing $\mathcal{Y}$ for each $\boldsymbol{y}^\xi$ (which renders the on-the-fly generation difficult). Further, violated scenarios are identified by solving

Algorithm 3.2.2: *optimizeRelaxation*$(\mathcal{B}, \mathcal{L}^R, \mathcal{N}^R)$: column generation algorithm for computing the dual bound at each node of the search tree when solving (3.2.22)-(3.2.27). In the special case where Assumption 3.2.1 as well as Proposition 3.2.4 hold, the algorithm takes a simpler form.

Input: $\mathcal{B}$: set of branching constraints, $\mathcal{L}^R$: set of indices of columns, $\mathcal{N}^R$: set of no-good cuts

repeat
  Solve $(MP(\mathcal{L}^R))$ with additional branching constraints $\mathcal{B}$ and no-good cuts $\mathcal{N}^R$
  Let $(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{\alpha}^*)$ be the optimal solution and $\boldsymbol{\pi}^*$, $\boldsymbol{\mu}^*$, λ^* and $\boldsymbol{\eta}^*$ be the optimal dual variables associated with the constraints (3.2.17), (3.2.18), (3.2.19) and (3.2.28)
  Solve $(Pricing'(\boldsymbol{\pi}^*, \boldsymbol{\mu}^*, \lambda^*, \boldsymbol{\eta}^*))$
  Let $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ be the optimal solution
  if $-\sum_{\mathcal{I} \in \mathcal{N}^{\mathrm{R}}} \eta^*_{\mathcal{I}} z^*_{\mathcal{I}} - \lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \boldsymbol{y}^* < 0$ **then**
    $\mathcal{L}^R \leftarrow \mathcal{L}^R \cup \{|\mathcal{L}^R| + 1\}$, $\bar{\boldsymbol{y}}^{|\mathcal{L}^R|} \leftarrow \boldsymbol{y}^*$
until $-\sum_{\mathcal{I} \in \mathcal{N}^{\mathrm{R}}} \eta^*_{\mathcal{I}} z^*_{\mathcal{I}} - \lambda^* + \left(\boldsymbol{f} - \boldsymbol{H}^\top \boldsymbol{\pi}^* + \boldsymbol{Q}^\top \boldsymbol{\mu}^*\right)^\top \boldsymbol{y}^* \geq 0$
return $(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{\alpha}^*)$

a bilinear problem, which can be very challenging. On the other hand, we propose to generate columns that correspond to the extreme points of $\mathcal{Y}$, and generate at most one cut of type (3.2.22) for each candidate solution. The difficulty of the pricing problem is the same as optimizing a linear function over the set $\mathcal{Y}$.

3.2.2.3 An exact extended deterministic equivalent formulation

The results of Section 3.2.2.2 allow the exact solution of problem (3.2.1) when Assumption 3.2.1 is satisfied, *i.e.*, when the linking constraints are expressed only in terms of the binary variables in the set $\mathcal{Y}$. In this section, we propose a reformulation in an extended space that enables us to exploit those results even when problem (3.2.1) does not naturally present in a form that satisfies this assumption. In other words, we present an alternative formulation of the recourse feasible region $\mathcal{Y}(\boldsymbol{x})$, that allows transforming problem (3.2.1) into a problem that satisfies Assumption 3.2.1. The reformulation we propose additionally produces problems that satisfy the assumptions of Proposition 3.2.4. The main result of this section is therefore showing that problem (3.2.1) can be solved using the branch-and-price algorithm of Section 3.2.2.2 regardless of the structure of the linking constraints after an appropriate reformulation of the set $\mathcal{Y}(\boldsymbol{x})$.

To this end, let $\boldsymbol{z} \in \{0,1\}^{N_1}$, and consider the reformulation of the recourse feasible region as

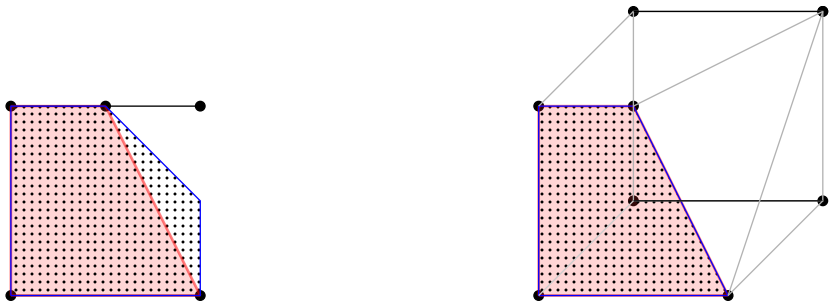
$$\mathcal{Y}'(\boldsymbol{x}) = \left\{\boldsymbol{y} \in \mathcal{Y}, \boldsymbol{z} \in \{0,1\}^{N_1} \middle| \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{z}, \boldsymbol{z} \leq \boldsymbol{x}_1, \boldsymbol{z} \geq \boldsymbol{x}_1\right\},$$

essentially incorporating a copy of the first-stage decision variables $\boldsymbol{x}_1$ and the linking constraints $\boldsymbol{Hy} \leq \boldsymbol{d} - \boldsymbol{Tx}_1$ to the recourse feasible region. We remark that it suffices to copy only those first-stage variables that are involved in the linking constraints. The advantage of this reformulation is that, the constraints $\boldsymbol{Hy} \leq \boldsymbol{d} - \boldsymbol{Tz}$ are expressed purely in terms of recourse variables and therefore can be incorporated to the subproblem. The new linking constraints $\boldsymbol{z} = \boldsymbol{x}_1$ ensure that the generated columns belong to the set $\text{conv}(\mathcal{Y}(\boldsymbol{x}))$ for $\boldsymbol{x} \in \mathcal{X}$.

Example 3.2.5. *Consider the second-stage feasibility set*

$$\mathcal{Y}(x) = \left\{ \boldsymbol{y} \in [0,1] \times \{0,1\} \;\middle|\; y_1 + y_2 \leq 1.5 + 0.5x \right\}$$

with $\mathcal{X} = \{0,1\}$. *Here the linking constraints do not follow Assumption 3.2.1. It turns out that* $\bar{\mathcal{Y}}(x) \neq \text{conv}(\mathcal{Y}(x))$ *for* $x \in \mathcal{X}$. *This is illustrated in Figure 3.2.5(a) for* $x^* = 0$, *where* $\text{conv}(\mathcal{Y}(0)) = \{\boldsymbol{y} \in [0,1]^2 \mid y_1 + 0.5y_2 \leq 1\}$ *and* $\bar{\mathcal{Y}}(0) = \{\boldsymbol{y} \in [0,1]^2 \mid y_1 + y_2 \leq 1.5\}$.



(a) (b)

— $\mathcal{Y}/\mathcal{Y}'$ $\bar{\mathcal{Y}}(x^*)/\bar{\mathcal{Y}}'(x^*)$ $\text{conv}(\mathcal{Y}(x^*))/\text{conv}(\mathcal{Y}'(x^*))$

Figure 3.2.5: Illustration of Example 3.2.5. Given the same first-stage solution $x^* = 0$, sets $\bar{\mathcal{Y}}(x^*)$ and $\text{conv}(\mathcal{Y}(x^*))$ are depicted for two different formulations of the second stage. In part (a), the formulation $\mathcal{Y}(x)$ does not follow Assumption 3.2.1, whereas the reformulation $\mathcal{Y}'(x)$ in part (b) does.

Consider now the extended formulation obtained by creating a copy of the variable x *in the second-stage feasible region, we write:*

$$\mathcal{Y}'(x) = \left\{ (\boldsymbol{y}, z) \in [0,1] \times \{0,1\}^2 \;\middle|\; \begin{array}{l} y_1 + y_2 \leq 1.5 + 0.5z \\ z = x \end{array} \right\}.$$

In this reformulation the constraint $y_1 + y_2 \leq 1.5 + 0.5z$ *is part of the definition of the set* $\mathcal{Y}$. *We have that* $\text{conv}(\mathcal{Y}) = \{(\boldsymbol{y}, z) \in [0,1]^3 \mid y_1 + 0.5y_2 \leq 1 + 0.5z\}$ *which is illustrated in grey in Figure 3.2.5(b). It follows that,* $\bar{\mathcal{Y}}'(x) = \{(\boldsymbol{y}, z) \in \text{conv}(\mathcal{Y}) \mid z = x\} = \{(\boldsymbol{y}, z) \in [0,1]^3 \mid y_1 + 0.5y_2 \leq 1 + 0.5z, z = x\}$. *As a result,* $\bar{\mathcal{Y}}'(x^*)$ *is the face highlighted in red in Figure 3.2.5(b) which is also equal to* $\text{conv}(\mathcal{Y}'(x^*))$. *Indeed, it follows that* $\bar{\mathcal{Y}}'(x) = \text{conv}(\mathcal{Y}'(x))$ *for* $x \in \mathcal{X}$ *by Proposition 3.2.4 as the new linking constraint* $z = x$ *follows its assumptions.*

Let $\mathcal{Y}' = \left\{\boldsymbol{y} \in \mathcal{Y}, \boldsymbol{z} \in \{0,1\}^{N_1} \middle| \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{z}\right\}$ and $(\bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})^j$ for $j \in \mathcal{L}' = \{1, \ldots, L'\}$ be the extreme point solutions of $\text{conv}(\mathcal{Y}')$. We may then write problem (3.2.2) as:

$$\min \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{f}^\top \sum_{j \in \mathcal{L}'} \alpha^j \bar{\boldsymbol{y}}^j + \boldsymbol{u}^\top \boldsymbol{b} \tag{3.2.33}$$

$$\text{s.t.} \quad \boldsymbol{x}_1 = \sum_{j \in \mathcal{L}'} \alpha^j \bar{\boldsymbol{z}}^j \tag{3.2.34}$$

$$\boldsymbol{A}^\top \boldsymbol{u} = \boldsymbol{Q}^\top \sum_{j \in \mathcal{L}'} \alpha^j \bar{\boldsymbol{y}}^j \tag{3.2.35}$$

$$\sum_{j \in \mathcal{L}'} \alpha^j = 1 \tag{3.2.36}$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{\alpha} \in \mathbb{R}_+^{L'}, \boldsymbol{u} \in \mathbb{R}_+^{\top}. \tag{3.2.37}$$

Formulation (3.2.33)-(3.2.37) provides a generic formulation for problem (3.2.1), where the linking constraints (3.2.34) involve only binary second-stage variables, *i.e.*, satisfy Assumption 3.2.1, and satisfy assumptions of Proposition 3.2.4 (writing $\boldsymbol{z} \leq \boldsymbol{x}_1$ and $\boldsymbol{z} \geq \boldsymbol{x}_1$). It can therefore be solved using the branch-and-price algorithm (Algorithm 3.2.1), generating columns from the set $\mathcal{Y}'$, and branching when the variables $\boldsymbol{x}_1$ are fractional.

Although formulation (3.2.33)-(3.2.37) has desirable theoretical properties, its numerical efficiency depends on the dimension of the reformulation $\mathcal{Y}'(\boldsymbol{x})$. To this end, we remark that the ideas presented in this section can be used for a subset of the $\boldsymbol{x}_1$ variables. Indeed, it suffices to create copies of those first-stage variables that are involved in linking constraints that do not satisfy Assumption 3.2.1 or the conditions of Proposition 3.2.4. We conclude this section with an example illustrating this idea.

Example 3.2.6. *Consider the second-stage feasibility set*

$$\mathcal{Y}(\boldsymbol{x}, x_0) = \left\{ \boldsymbol{y} \in \{0,1\}^2 \;\middle|\; \begin{array}{ll} y_1 + y_2 \leq 1.5 + 0.5x_0 \\ y_i \geq x_i \quad \forall i = 1,2 \end{array} \right\}$$

with $\mathcal{X} = \{0,1\}^3$. *Here the linking constraints follow Assumption 3.2.1. However, the linking constraint* $y_1 + y_2 \leq 1.5 + 0.5x_0$ *does not follow the assumption of Proposition 3.2.4, although constraints* $y_i \geq x_i$ *for* $i = 1, 2$ *do. We remark that* $\bar{\mathcal{Y}}(\boldsymbol{x}) \neq \text{conv}(\mathcal{Y}(\boldsymbol{x}))$ *for* $\boldsymbol{x} \in \mathcal{X}$.

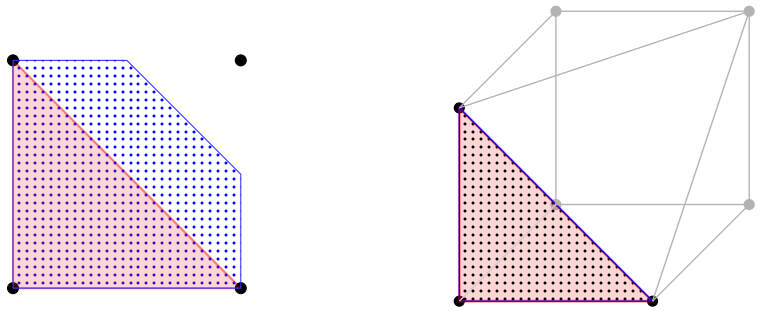
Consider $\boldsymbol{x}^* = (0,0,0)$. *We have that* $\text{conv}(\mathcal{Y}(\boldsymbol{x}^*)) = \{y \in [0,1]^2 \mid y_1 + y_2 \leq 1\}$ *and that* $\bar{\mathcal{Y}}(\boldsymbol{x}^*) = \{\boldsymbol{y} \in [0,1]^2 \mid y_1 + y_2 \leq 1.5\}$. *This is illustrated in Figure 3.2.6(a).*

Consider now the extended formulation obtained by creating a copy of the variable x_0 *in the second-stage feasible region, we write:*

$$\mathcal{Y}'(\boldsymbol{x}, x_0) = \left\{ (\boldsymbol{y}, z) \in \{0,1\}^3 \;\middle|\; \begin{array}{ll} y_1 + y_2 \leq 1.5 + 0.5z \\ y_i \geq x_i \quad \forall i = 1,2 \\ z = x_0 \end{array} \right\}.$$

In this reformulation the constraint $y_1 + y_2 \leq 1.5 + 0.5z_0$ *is part of the definition of the set* $\mathcal{Y}$. *We have that* $\text{conv}(\mathcal{Y}') = \{(\boldsymbol{y}, z) \in [0,1]^3 \mid y_1 + y_2 \leq 1 + z\}$ *which is illustrated*

in grey in Figure 3.2.5(b). It follows that, $\bar{\mathcal{Y}}'(\boldsymbol{x}) = \{(\boldsymbol{y}, z) \in \text{conv}(\mathcal{Y}) \mid y_i \geq x_i \ \ \forall i = 1, 2, z = x_0\} = \{(\boldsymbol{y}, z) \in [0,1]^3 \mid y_1 + y_2 \leq 1 + z, y_i \geq x_i \ \ \forall i = 1, 2, z = x_0\}$. *As a result,* $\bar{\mathcal{Y}}'(\boldsymbol{x}^*)$ *is the face highlighted in red in Figure 3.2.5(b) which is also equal to* $\text{conv}(\mathcal{Y}'(\boldsymbol{x}^*))$. *Indeed,* $\bar{\mathcal{Y}}'(\boldsymbol{x}) = \text{conv}(\mathcal{Y}'(\boldsymbol{x}))$ *for* $\boldsymbol{x} \in \mathcal{X}$ *by Proposition 3.2.4 as the new linking constraint* $z = x_0$, *along with the linking constraints* $y_i \geq x_i$ *for* $i = 1, 2$ *follow its assumptions.*



(a) (b)

— $\mathcal{Y}/\mathcal{Y}'$ $\bar{\mathcal{Y}}(\boldsymbol{x}^*)/\bar{\mathcal{Y}}'(\boldsymbol{x}^*)$ $\text{conv}(\mathcal{Y}(\boldsymbol{x}^*))/\text{conv}(\mathcal{Y}'(\boldsymbol{x}^*))$

Figure 3.2.6: Illustration of Example 3.2.6. Given the same first-stage solution $\boldsymbol{x}^* = (0, 0, 0)$, sets $\bar{\mathcal{Y}}(\boldsymbol{x}^*)$ and $\text{conv}(\mathcal{Y}(\boldsymbol{x}^*))$ are depicted for two different formulations of the second stage. In part (a), the formulation $\mathcal{Y}(\boldsymbol{x})$ does not follow Proposition 3.2.4, whereas the reformulation $\mathcal{Y}'(\boldsymbol{x})$ in part (b) does.

3.2.2.4 Reformulation through enumeration

In this section, we present an alternative formulations of (3.2.1) in certain special cases. It is based on the idea of enumerating the first- and second-stage feasible solutions and creating an uncertainty vector corresponding to each first-stage solution. Writing such a formulation in theory for pure binary sets $\mathcal{X}$ and $\mathcal{Y}$ is always possible but in practice is only viable when $\mathcal{X}$ and $\mathcal{Y}$ are easily enumerable. Although direct solution of this formulations is extremely time/memory consuming for larger instances, it provides benchmarks for evaluating the computational efficacy of the column generation-based approach proposed above. In this section, we assume that $\mathcal{X}$ and $\mathcal{Y}$ are pure binary sets. Let us denote by $\boldsymbol{x}^i$, for $i \in \mathcal{K} = \{1, \ldots, K\}$, the feasible solutions of $\mathcal{X}$. For $i = 1, \ldots, K$, let $\boldsymbol{y}^{i,j} \in \mathcal{Y}(\boldsymbol{x}^i)$ for $j \in \mathcal{L}_i = 1, \ldots, L_i$ be the feasible solutions of $\mathcal{Y}(\boldsymbol{x}^i)$. We write

$$\begin{aligned}
\max \quad & \theta && (3.2.38)\\
\text{s.t.} \quad & \theta \leq \theta^i \quad \forall i \in \mathcal{K} && (3.2.39)\\
& \theta^i \leq \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^i + (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}}\boldsymbol{y}^{i,j} \quad \forall i \in \mathcal{K}, j \in \mathcal{L}_i && (3.2.40)\\
& \boldsymbol{\xi}^i \in \Xi \quad \forall i \in \mathcal{K}. && (3.2.41)
\end{aligned}$$

Proposition 3.2.7. *Linear program* (3.2.38)-(3.2.41) *is a formulation of* (3.2.1).

Proof. Consider a first-stage feasible solution $\boldsymbol{x}^i \in \mathcal{X}$. We write the inner optimization problem corresponding to this solution:

$$\max_{\boldsymbol{\xi}^i \in \Xi} \min_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}^i)} (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}} \boldsymbol{y}$$

and its corresponding epigraph formulation by enumerating the feasible solutions of $\mathcal{Y}(\boldsymbol{x}^i)$:

$$\begin{aligned} \max_{\underline{\theta}^i \in \mathbb{R}, \boldsymbol{\xi}^i \in \Xi} \quad & \underline{\theta}^i \\ \text{s.t.} \quad & \underline{\theta}^i \leq (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}} \boldsymbol{y}^{i,j} \quad \forall j \in \mathcal{L}_i \end{aligned}$$

As a result, we may write the objective value of the solution $\boldsymbol{x}^i$ as

$$\begin{aligned} \max_{\theta^i \in \mathbb{R}, \boldsymbol{\xi}^i \in \Xi} \quad & \theta^i \\ \text{s.t.} \quad & \theta^i \leq \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^i + (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}} \boldsymbol{y}^{i,j} \quad \forall j \in \mathcal{L}_i. \end{aligned}$$

Problem (3.2.1) can then be expressed as

$$\begin{aligned} \min_{i \in \mathcal{K}} \quad & \theta^i \\ \text{s.t.} \quad & \theta^i \leq \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^i + (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}} \boldsymbol{y}^{i,j} \quad \forall j \in \mathcal{L}_i \\ & \boldsymbol{\xi}^i \in \Xi \end{aligned}$$

or equivalently,

$$\begin{aligned} \max \quad & \theta \\ \text{s.t.} \quad & \theta \leq \theta^i \quad \forall i \in \mathcal{K} \\ & \theta^i \leq \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^i + (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi}^i)^{\mathrm{T}} \boldsymbol{y}^{i,j} \quad \forall i \in \mathcal{K}, \forall j \in \mathcal{K}_i \\ & \boldsymbol{\xi}^i \in \Xi \quad \forall i \in \mathcal{K} \end{aligned}$$

□

3.2.2.5 Reformulation through dynamic programming

In this section, we present another reformulation assuming that an extended linear programming formulation for the recourse problem exists. This is, for instance, the case if there exists a dynamic programming equivalent (presumably in a much higher dimensional space) that can be cast as a shortest path problem on an appropriately defined network (see e.g. [Martin *et al.* 1990]). As we assume that the linear programming relaxation is integral, in this case $\mathrm{conv}(\mathcal{Y}(\boldsymbol{x})) = \mathcal{Y}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{X}$. Therefore we may directly write the deterministic equivalent (3.2.2) replacing $\mathrm{conv}(\mathcal{Y}(\boldsymbol{x}))$ by its linear programming equivalent. We illustrate this idea below in the case of dynamic programs written as equivalent shortest path formulations on an appropriately defined network.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be the network associated with the dynamic programming formulation of $\mathcal{Y}(\boldsymbol{x})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. Let further $\boldsymbol{F}$ be the incidence matrix associated to $\mathcal{G}$, ϕ_a be the flow variable associated with arc $a \in \mathcal{A}$ and $\bar{y}_i^a$ be the value of variable y_i on arc $a \in \mathcal{A}$ for $i = 1, \dots, M$. Then a deterministic equivalent formulation for (3.2.1) can be written as

$$\min \quad \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{b} \tag{3.2.42}$$

$$\text{s.t.} \quad \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1 \tag{3.2.43}$$

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{u} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} \tag{3.2.44}$$

$$\boldsymbol{F}\boldsymbol{\phi} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{3.2.45}$$

$$y_i = \sum_{a \in \mathcal{A}} \bar{y}_i^a \phi_a \quad \forall i \in \mathcal{I} \tag{3.2.46}$$

$$\boldsymbol{x} \in \mathcal{X}, \boldsymbol{u} \in \mathbb{R}_+^T, \boldsymbol{y} \in [0,1]^I, \boldsymbol{\phi} \in [0,1]^{\mathcal{A}}. \tag{3.2.47}$$

Clearly, the computational tractability of (3.2.42)-(3.2.47) is dependent on the size of the network $\mathcal{G}$. Unfortunately, in most applications, this size is pseudo-polynomial at best and exponential at worst. However, one might still consider dynamic aggregation/disaggregation (see e.g. [Clautiaux *et al.* 2017]) or state-space relaxation techniques (see Section 1.2.3.3) to efficiently solve these models.

Therefore, formulation (3.2.42)-(3.2.47) is potentially prohibitive for practically sized instances. However, we use it as a baseline to assess the computational efficacy of our column generation approach.

3.2.3 Complexity results

Min-max-min problems, even with linear objective and constraints, are in general (most probably) harder than NP-complete problems (*i.e.*, in the second or third level of the polynomial hierarchy). Falling into this category, two-stage robust problems with integer recourse are often suspected to be outside of class NP. For example, in [Bertsimas *et al.* 2013b], the authors study such a robust network flow problem which is shown to be NP-hard and yet admits a strong dual. To explain this odd situation, they conjecture that the problem and its dual are respectively Σ_2^P- and Π_2^P-complete. As a second example, [Deineko & Woeginger 2010] show that a special case of the min-max regret knapsack problem with uncertain objective is Σ_2^P-complete.

In this section, we show that despite the three-level structure of problem (3.2.1), it is NP-complete. First, formulation (3.2.10)-(3.2.14) reveals that the problem of solving (R) actually lies inside the class NP (Proposition 3.2.8). Second, Section 3.2.2.3 shows that any problem of form (3.2.1) can be reformulated (by adding a polynomial number of variables and constraints) as a problem for which relaxation (R) is exact. Therefore, problem (3.2.1) is NP-complete (Theorem 3.2.10) as it can be solved through its (exact) relaxation (R) using model (3.2.10)-(3.2.14) or

model (3.2.33)-(3.2.37). In the following, we formalize these results starting with the decision problem associated to solving (R).

PROBLEM: RELAXED TWO STAGE ROBUST MILP (R2SRMILP)
INPUT DATA: Integer η, positive integers $N = p + p', N', M = q + q', M', M'', S, S'$, first-stage data $\boldsymbol{G} \in \mathbb{Z}^{N' \times N}$, $\boldsymbol{g} \in \mathbb{Z}^{N'}$, $\boldsymbol{c} \in \mathbb{Z}^{N}$, second-stage data $\boldsymbol{E} \in \mathbb{Z}^{M'' \times M}$, $\boldsymbol{e} \in \mathbb{Z}^{M''}$, $\boldsymbol{f} \in \mathbb{Z}^{M}$, $\boldsymbol{Q} \in \mathbb{Z}^{M \times S}$, linking constraints data $\boldsymbol{H} \in \mathbb{Z}^{M' \times M}$, $\boldsymbol{d} \in \mathbb{Z}^{M'}$, $\boldsymbol{T} \in \mathbb{Z}^{M' \times N}$, uncertainty set data $\boldsymbol{A} \in \mathbb{Z}^{S' \times S}$, $\boldsymbol{b} \in \mathbb{Z}^{S'}$, such that $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{N}^p \times \mathbb{R}_+^{p'} : \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{g}\}$, $\mathcal{Y} = \{\boldsymbol{y} \in \mathbb{N}^q \times \mathbb{R}_+^{q'} : \boldsymbol{E}\boldsymbol{y} \leq \boldsymbol{e}\}$, and $\Xi = \{\boldsymbol{\xi} \in \mathbb{R}^S : \boldsymbol{A}\boldsymbol{\xi} \leq b\}$ are bounded polyhedral sets. Let $\bar{\mathcal{Y}}(\boldsymbol{x}) = \text{conv}(\mathcal{Y}) \cap \{\boldsymbol{y} : \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}\}$.
QUESTION: Does $\min_{\boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \bar{\mathcal{Y}}(\boldsymbol{x})} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \max_{\boldsymbol{\xi} \in \Xi} \ (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi})^{\mathrm{T}}\boldsymbol{y} \leq \eta$?

Proposition 3.2.8. *Problem* R2SRMILP *is NP-complete.*

Proof. The problem is trivially NP-hard, since choosing $M = 0$ makes the problem a general MILP. To show that it lies inside NP, let us assume that the answer of R2SRMILP is YES. It follows that there is a solution to (3.2.10)-(3.2.14), whose cost is not larger than η. Moreover, using Carathéodory's theorem, there exists such a solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{\alpha}})$ where the number Θ of non-zero entries of $\hat{\boldsymbol{\alpha}}$ is bounded above by a polynomial of M' and S. Let $\theta(j)$ be the j^{th} non-zero entry in $\hat{\boldsymbol{\alpha}}$. Then, we can build a certificate C by concatenating the significant elements of $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{\alpha}})$ and the description of the associated columns: $C = \left(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \left(\hat{\boldsymbol{\alpha}}^{\theta(j)}\right)_{1 \leq j \leq \Theta}, \left(\hat{\boldsymbol{y}}^{\theta(j)}\right)_{1 \leq j \leq \Theta}\right)$. The description of each column $\hat{\boldsymbol{y}}^{\theta(j)}$ requires at most M integers, so that the size of C is bounded by a polynomial of the input data.

In order to prove the existence of a solution from a certificate C, one could verify that it provides a solution to (3.2.10)-(3.2.14). However, checking that $\left(\hat{\boldsymbol{y}}^{\theta(j)}\right)_{1 \leq j \leq \Theta}$ is indeed part of the model would require solving a NP-hard problem (checking that each $\hat{\boldsymbol{y}}^{\theta(j)}$ is indeed an extreme point of $\mathcal{Y}$). In order to design a polynomial-time algorithm processing C, we therefore write the equivalent formulation of (3.2.10)-(3.2.14):

$$\min \left\{ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{b} : \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}, \boldsymbol{A}^{\mathrm{T}}\boldsymbol{u} = \boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y}, \right.$$
$$\left. \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \text{conv}(\mathcal{Y}), \boldsymbol{u} \in \mathbb{R}_+^{S'} \right\}. \quad (3.2.48)$$

From C, one can check that $\dot{\boldsymbol{y}} = \sum_{j=1}^{\Theta} \hat{\boldsymbol{\alpha}}^{\theta(j)} \hat{\boldsymbol{y}}^{\theta(j)} \in \text{conv}(\mathcal{Y})$ by verifying that $\hat{\boldsymbol{y}}^{\theta(j)} \in \mathcal{Y}$ for all $j \in \{1, \ldots, \Theta\}$ and $\sum_{j=1}^{\Theta} \hat{\boldsymbol{\alpha}}^{\theta(j)} = 1$. This can trivially be done in polynomial time with help of the mathematical programming representation of $\mathcal{Y}$ provided as an input of R2SRMILP. It then suffices to check the cost and the feasibility of $(\hat{\boldsymbol{x}}, \dot{\boldsymbol{y}}, \hat{\boldsymbol{u}})$ for the rest of (3.2.48). □

Remark 3.2.3. *This NP-completeness result does not depend on the restrictions over $\boldsymbol{x}_1$ imposed in the definition of* (3.2.1).

As a result of Proposition 3.2.4, problems of form (3.2.1) that satisfy its assumptions are equivalent to (3.2.4)-(3.2.6), *i.e.*, problem R2SRMILP, which leads to the

following complexity result.

Corollary 3.2.9. *If* $\boldsymbol{H} = I$, $\boldsymbol{T} = -I$ *and* $\boldsymbol{d} = 0$, *or if* $\boldsymbol{H} = I$, $\boldsymbol{T} = I$ *and* $\boldsymbol{d} = \mathbf{1}$, *then solving problem* (3.2.1) *is NP-complete.*

Finally, the reformulation of (3.2.1) proposed in Section 3.2.2.3 allows us to show that this problem lies inside class NP as well.

PROBLEM: TWO STAGE ROBUST MILP (2SRMILP) WITH BINARY FIRST-STAGE VARIABLES
INPUT DATA: Integer η, positive integers $N = N_1 + p + p', N', M = q + q', M', M'', S, S'$, first-stage data $\boldsymbol{G} \in \mathbb{Z}^{N' \times N}$, $\boldsymbol{g} \in \mathbb{Z}^{N'}$, $\boldsymbol{c} \in \mathbb{Z}^{N}$, second-stage data $\boldsymbol{E} \in \mathbb{Z}^{M'' \times M}$, $\boldsymbol{e} \in \mathbb{Z}^{M''}$, $\boldsymbol{f} \in \mathbb{Z}^{M}$, $\boldsymbol{Q} \in \mathbb{Z}^{M \times S}$, linking constraints data $\boldsymbol{H} \in \mathbb{Z}^{M' \times M}$, $\boldsymbol{d} \in \mathbb{Z}^{M'}$, $\boldsymbol{T} \in \mathbb{Z}^{M' \times N}$, uncertainty set data $\boldsymbol{A} \in \mathbb{Z}^{S' \times S}$, $\boldsymbol{b} \in \mathbb{Z}^{S'}$, such that $\mathcal{X} = \{\boldsymbol{x} \in \{0,1\}^{N_1} \times \mathbb{N}^p \times \mathbb{R}_+^{p'} : \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{g}\}$, $\mathcal{Y} = \{\boldsymbol{y} \in \mathbb{N}^q \times \mathbb{R}_+^{q'} : \boldsymbol{E}\boldsymbol{y} \leq \boldsymbol{e}\}$, and $\Xi = \{\boldsymbol{\xi} \in \mathbb{R}^S : \boldsymbol{A}\boldsymbol{\xi} \leq b\}$ are bounded polyhedral sets. Let $\boldsymbol{x}_1 \in \{0,1\}^{N_1}$ be the subset of binary first-stage variables, and $\mathcal{Y}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathcal{Y} : \boldsymbol{H}\boldsymbol{y} \leq \boldsymbol{d} - \boldsymbol{T}\boldsymbol{x}_1\}$.
QUESTION: Does $\min_{\boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \max_{\boldsymbol{\xi} \in \Xi} \ (\boldsymbol{f} + \boldsymbol{Q}\boldsymbol{\xi})^{\mathrm{T}}\boldsymbol{y} \leq \eta$?

Therorem 3.2.10. *Problem* 2SRMILP *is NP-complete.*

Proof. From any instance Π of 2SRMILP, one can build an equivalent instance Π' satisfying Assumption 3.2.1, as shown in Section 3.2.2.3, whose size is polynomial in the size of Π. Instance Π' is then equivalent to the related instance of R2SRMILP. Thus, solving any instance of 2SRMILP is equivalent to solving an appropriately constructed instance of R2SRMILP, whose size is polynomial in the size of Π. □

Remark 3.2.4. *This NP-completeness result does not depend on the restrictions over variables* $\boldsymbol{y}$. *However, the restrictions posed on variables* $\boldsymbol{x}_1$ *in the definition of* (3.2.1) *are necessary.*

3.2.4 Numerical results

In this section we demonstrate the application of the methodologies developed in Section 3.2.2 in various contexts. We present numerical results on two applications, presenting a comparison between the branch-and-price (B&P) algorithm, and the approaches from the $K-$adaptability literature with $K = 2, 3, 4$: the monolithic reformulation approach presented in [Hanasusanto *et al.* 2015] (2-,3-,4-AdaptM), and the semi-definite programming-based branch-and-bound algorithm presented in [Subramanyam *et al.* 2020] (2-,3-,4-AdaptBB). A first application is a knapsack variant that gives the possibility to reject/produce or repair items in the second stage. It involves a recourse problem with knapsack-type constraints expressed purely in terms of recourse variables along with linking constraints of type $\boldsymbol{y} \leq \boldsymbol{x}$. In this case, we may apply directly the result of Proposition 3.2.4 to show that the deterministic equivalent model (3.2.10)-(3.2.14) based on relaxation (R) is exact. A second application is centered around a variant of the capital budgeting problem

studied previously in K-Adaptability literature (see e.g. [Hanasusanto *et al.* 2015], [Subramanyam *et al.* 2020]). In this case, it turns out that the assumptions of Proposition 3.2.4 are not verified. As a result, we repose on the extended formulation proposed in Section 3.2.2.3.

To obtain optimal integer solutions to our formulations based on models (3.2.10)-(3.2.14) and (3.2.33)-(3.2.37), we use the C++ branch-and-price library BaPCod[2]. At each node of the search tree, the linear relaxation of the problem is solved using column generation. The pricing sub-problems are solved using dynamic programming algorithms, using simple array-based forward label-correcting. At most one column is added to the master program at each iteration. To improve the convergence of the column generation procedure, we use stabilization by automatic smoothing of the dual variables of the master program, as described in [Pessoa *et al.* 2018b]. When the optimal solution of the corresponding relaxation does not satisfy the integrality requirements of first-stage variables $\boldsymbol{x}$, one such fractional variable is chosen and two child nodes are created in order to exclude its current value from the search space. In order to reduce the size of the search tree, the branching choices are made with help of the strong branching technique, as described in [Sadykov *et al.* 2019]. The open nodes are processed according to the best first rule. At the root node, and each tenth processed node, a diving heuristic [Sadykov *et al.* 2019] is used to derive a feasible solution and try to improve the best known primal bound. The diving heuristic is used only at nodes whose depth is at most ten. The implementations of the branch-and-price and pricing sub-problem solvers are sequential.

The branch-and-bound algorithm of [Subramanyam *et al.* 2020] is adapted to each application using the authors' implementation available online[3]. All mixed integer linear programs, including mathematical models resulting from the monolithic reformulation of [Hanasusanto *et al.* 2015] and linear programs inside the column generation procedures, are solved using IBM ILOG Cplex 12.9, through the C callable library, using default parameters and four threads.

All our experiments are conducted using a 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz machine with 128Go RAM running Linux OS. The resources of this machine are strictly partitioned using Slurm Workload Manager[4] to run several tests in parallel. The resources available for each run (algorithm-instance) are set to 4 threads and a 20 Go RAM limit (we remark that our branch-and-price algorithms do not benefit from parallel processing). This virtually creates six independent machines, each running one single instance at a time.

Sections 3.2.4.1 and 3.2.4.2 introduce the applications in detail, present the details of instances generated, and the results obtained.

[2] `https://realopt.bordeaux.inria.fr/?page_id=2` (accessed June 2019)

[3] `https://github.com/AnirudhSubramanyam/KAdaptabilitySolver/tree/v1.0` (accessed June 2020)

[4] `https://slurm.schedmd.com/` (accessed June 2019)

3.2.4.1 Two-stage robust knapsack problem

Consider a two-stage robust knapsack problem with the set of items $\mathcal{I} = \{1, \dots, I\}$. Each item has a weight c_i and an uncertain profit $\tilde{p}_i \in [\bar{p}_i - \hat{p}_i, \bar{p}_i]$ for $i \in \mathcal{I}$ where $\bar{p}_i$ is the expected profit and $\hat{p}_i$ is its maximum deviation. In this problem, a first stage decision is to choose a subset of items to produce. After this choice, a profit degradation factor $\boldsymbol{\xi} \in \Xi = \{\boldsymbol{\xi} \in \mathbb{R}^I_+ \mid \sum_{i\in\mathcal{I}} \xi_i \leq \Gamma, 0 \leq \xi_i \leq 1\}$ is revealed. We define $p_i(\boldsymbol{\xi}) = \bar{p}_i - \xi_i \hat{p}_i$ for $i \in \mathcal{I}$. After observing the profit degradation, there are three possible recourse actions:

(i) Produce the item as is, using c_i units of the knapsack capacity, with the degraded profit $\bar{p}_i - \xi_i \hat{p}_i$.

(ii) Repair the item, using an additional t_i units of the knapsack capacity, recovering the original profit $\bar{p}_i$.

(iii) Outsource the item, with associated profit $\bar{p}_i - f_i$ where f_i is the cost of outsourcing the item.

We next give a mathematical formulation for this problem. Let, for $i \in \mathcal{I}$, x_i denote the decision to produce item i in the first-stage, and $y_i = 1$, $r_i = 1$ and $y_i = 0$ denote the decisions to produce without repair, repair or outsource item i in the second-stage, respectively. We then write,

$$\min_{\boldsymbol{x}\in\{0,1\}^I} \sum_{i\in\mathcal{I}} (f_i - \bar{p}_i)x_i + \max_{\boldsymbol{\xi}\in\Xi} \min_{\boldsymbol{y},\boldsymbol{r}\in\mathcal{Y}(\boldsymbol{x})} \sum_{i\in\mathcal{I}} (\hat{p}_i\xi_i - f_i)y_i - \hat{p}_i\xi_i r_i \tag{3.2.49}$$

where

$$\mathcal{Y}(\boldsymbol{x}) = \left\{ \boldsymbol{y} \in \{0,1\}^I, \boldsymbol{r} \in \{0,1\}^I \;\middle|\; \begin{array}{ll} \sum_{i\in\mathcal{I}} c_i y_i + t_i r_i \leq C & \\ y_i \leq x_i & \forall i \in \mathcal{I} \\ r_i \leq y_i & \forall i \in \mathcal{I} \end{array} \right\}.$$

As the linking constraints $y_i \leq x_i$ for $i \in \mathcal{I}$ conform with the assumption of Proposition 3.2.4, we can directly solve the deterministic equivalent formulation (3.2.10)-(3.2.14), generating the columns from the generalized knapsack set:

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in \{0,1\}^I, \boldsymbol{r} \in \{0,1\}^I \;\middle|\; \begin{array}{ll} \sum_{i\in\mathcal{I}} c_i y_i + t_i r_i \leq C & \\ r_i \leq y_i & \forall i \in \mathcal{I} \end{array} \right\}.$$

In this case, the pricing problem takes the form

$$-\lambda + \min_{\boldsymbol{y},\boldsymbol{r}\in\mathcal{Y}} \sum_{i\in\mathcal{I}} (-f_i + \hat{p}_i\pi_i - \mu_i)y_i - \sum_{i\in\mathcal{I}} \hat{p}_i\pi_i r_i$$

which can be solved using an extension of the pseudo-polynomial dynamic programming algorithm for the classical knapsack problem. We additionally test a K-Adaptability approach to this problem, the associated model can be found in [Arslan & Detienne 2021].

Instance generation For our numerical tests we generate instances inspired by those presented by [Pisinger 2005]. These instances are categorized as uncorrelated (UN), weakly correlated (WC), almost strongly correlated (ASC), and strongly correlated (SC). For given number of items I, and parameters $R = 1000$, $H = 100$, $h \in \{40, 80\}$, $\delta \in \{0.1, 0.5, 1\}$, we generate $c_i \in [1, R]$ for $i \in \mathcal{I}$ and $C = \frac{h}{H+1} \sum_{i \in \mathcal{I}} c_i$. The profit $\bar{p}_i$ for $i \in \mathcal{I}$ is then generated based on the category of instances as follows: $\bar{p}_i = [1, R]$ for UN, $\bar{p}_i \in [c_i - \frac{R}{20}, c_i + \frac{R}{20}]$ for WC, $\bar{p}_i \in [c_i + \frac{R}{10} - \frac{R}{1000}, c_i + \frac{R}{10} + \frac{R}{1000}]$ for ASC, and $\bar{p}_i = c_i + \frac{R}{10}$ for SC. Based on $\bar{p}_i$, the maximum degradation factor $\hat{p}_i$ for $i \in \mathcal{I}$ is generated in the interval $[\bar{p}_i(1-\delta)/2, \bar{p}_i(1+\delta)/2]$ and the penalty of rejecting an item f_i for $i \in \mathcal{I}$ is generated in the interval $[1.1\bar{p}_i, 1.5\bar{p}_i]$. Finally, repair capacity t_i for $i \in \mathcal{I}$ is generated depending on the category of instances as follows: $t_i \in [1, c_i]$ for UN, $t_i \in [0.5\hat{p}_i - \frac{R}{40}, 0.5\hat{p}_i + \frac{R}{40}]$ for WC, $t_i \in [0.5\hat{p}_i + \frac{c_i}{10} - \frac{R}{1000}, 0.5\hat{p}_i + \frac{c_i}{10} + \frac{R}{1000}]$ for ASC, and $t_i = 0.5\hat{p}_i + \frac{c_i}{10}$ for SC.

Results In this section, we present our results on instances generated as described above. We solve instances with $I \in \{20, 30, 40, 50, 60, 70, 80\}$ and generate 72 instances for each value of I, 18 in each instance category (UN,WC,ASC,SC), corresponding to each combination of the parameters $h \in \{40, 80\}$, $\delta \in \{0.1, 0.5, 1\}$, and the uncertainty budget $\Gamma \in \{0.1I, 0.15I, 0.2I\}$.

We start by presenting, in Table 3.2.1, our results for 20-item instances with all solution methods considered. For these instances the time limit was set to 1 hour. In Table 3.2.1, #Conv represents the number of instances for which each method converged within the time limit. Time* represents the solution time (when converged) divided by the solution time of the branch-and-price algorithm. ConvGap(%) represents the percentage optimality gap reported by each method, that is the gap between the best primal and dual bounds found by the corresponding algorithm. OptGap(%) represents the percentage gap between the best primal solution reported by each method versus the optimal solution of the branch-and-price algorithm. #BestSol represents the number of instances for which each method was able to find the best primal solution, and #NotBestSol represents the number of instances for which this was not the case.

As can be seen in the first column of Table 3.2.1, the branch-and-price algorithm converged for all instances considered, while the monolithic approach of [Hanasusanto *et al.* 2015] converged for 41 when $K = 2$ and 8 when $K = 3$. This method did not converge for any of the instances considered when $K = 4$. Similarly, the branch-and-bound algorithm of [Subramanyam *et al.* 2020] converged for 44, 12 and 2 instances when $K = 2, 3$, and 4, respectively. A comparison of the average solution time for the branch-and-price algorithm to that of the other methods reveals the numerical promise of this approach. Not only did it solve the instances considered exactly, but it did so at least two orders of magnitude faster than other methods. On the other hand, although the gaps reported (ConvGap(%)) by $K-$adaptability methods were large, the primal solutions provided by these methods were on average within 1% (OptGap(%)) of the optimal solution provided by the branch-and-price

	#Conv	Time*	ConvGap(%)	OptGap(%)	#BestSol	#NotBestSol
B&P	72	1	0	0	72	0
DP-based	35	2611.55	50.92	23.95	41	28
2-AdaptM	41	468.26	9.72	0.61	16	56
3-AdaptM	8	5019.24	29.52	0.53	17	55
4-AdaptM	0	-	40.32	0.66	16	56
2-AdaptBB	44	613.21	2.87	0.61	14	58
3-AdaptBB	12	3951.08	3.87	0.32	16	56
4-AdaptBB	2	3145.44	4.59	0.24	19	53

Table 3.2.1: A summary of 20-item instance results with 8 different solution methods and 1 hour time limit. Solution time ratios are reported as an average over instances solved to convergence within the time limit. Convergence gaps are reported as an average over instances not solved to convergence within the time limit. Optimality gaps are reported as an average over 72 instances.

algorithm.

The column #BestSol provides further insight to this observation. It reveals that the optimal solution of the branch-and-price algorithm coincided with that of $K-$adaptability methods for a number of instances, which may happen when there exists an optimal solution which can be represented with at most K active columns. This reveals finite adaptability as a good approximation method for finding near-optimal solutions, at least for the problem, and the 20-item instances considered.

Finally, we remark that although the average gap of the dynamic programming based formulation is very large, this method was able to find the optimal solution for 41 of the 72 instances considered (and was able to prove optimality for 35). The large gap reported is explained by a few instances where this method was not able to find a primal solution and therefore had poor relative gaps. Indeed, if the calculation of this gap is restricted only to those instances where a primal solution is found then the gap for the DP-based method is %5.04. Additionally, for 3 of the instances considered, the memory limitations were reached, hence the discrepancy in the sum of the columns #BestSol and #NotBestSol ($41 + 28 \neq 72$).

In Figure 3.2.7, we provide the performance profile for five of the methods considered, plotting the number of 20-item instances for which the method converged versus the time (expressed in logarithmic scale). We remark that the K-Adaptability approaches with $K = 4$ were excluded as the number of instances solved to convergence was very small. This plot further confirms the numerical promise of the branch-and-price algorithm. It also shows that among the other methods considered the most promising is $2-$Adaptability.

Despite its lack of scalability, the DP-based formulation could be considered as an interesting alternative for solving the problem exactly, since it provides better solutions than 3-Adaptability in a shorter computing time.

We therefore further compare these three methods (B&P, 2-AdaptM, 2-AdaptBB) on 30-item instances, with a 2-hour time limit. The results are presented in Ta-

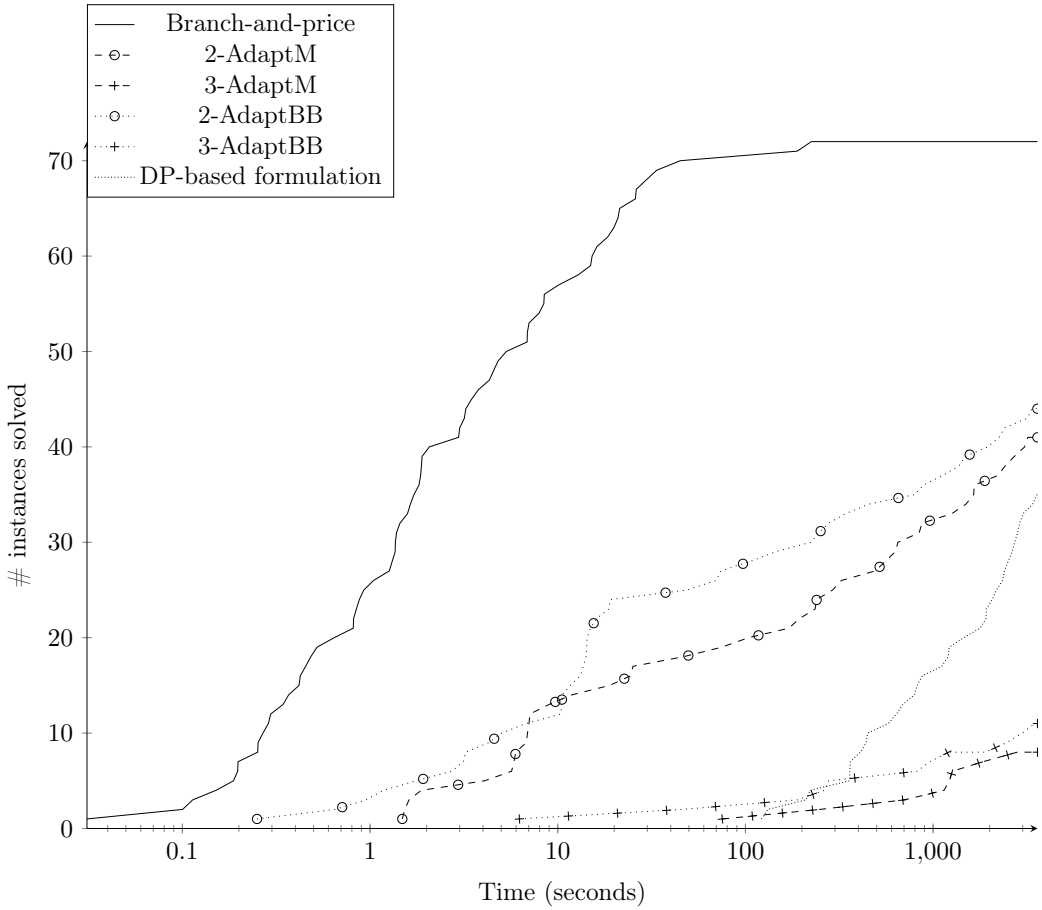


Figure 3.2.7: Performance profile comparing six methods for 20-item instances of the Robust Knapsack problem. Results for 4-Adaptability model are not shown since this method does not converge for any of the 20-item instances within the one-hour time limit.

ble 3.2.2 where the headers are kept the same as Table 3.2.1. The branch-and-price approach converges for 65 of the 72 instances considered whereas 2-AdaptM and 2-AdaptBB methods are able to converge for only 12 and 11 instances, respectively. Interestingly, the primal solutions provided by the 2-Adaptability approach are still within %1 of optimality, with each method providing a better solution in 6 of the instances.

We next explore the limits of the column generation-based approach with increasing number of items. In Table 3.2.3, we report our results for instances with $I \in \{20, 30, 40, 50, 60, 70, 80\}$, where starting from 40-item instances the time limit was set to 3 hours. We report the number of instances solved to optimality (#Opt), the average optimality gap reported (AvgGap (%)), and the maximum optimality gap reported among the instances considered (MaxGap(%)). The results show that starting from 50-item instances the 3-hour time limit was not sufficient although

	#Conv	Time*	ConvGap(%)	OptGap(%)	#BestSol	#NotBestSol
B&P	65	1	3.24	0	67	5
2-AdaptM	12	2886.42	21.98	0.63	6	66
2-AdaptBB	11	191.07	3.67	0.60	6	66

Table 3.2.2: A summary of 30-item instance results with branch-and-price algorithm and 2-Adaptability, and 2-hour time limit. Solution time ratios are reported as an average over instances solved to convergence before the time limit only. Convergence gaps are reported as an average over instances not solved to convergence before the time limit. Optimality gaps are reported as an average over 72 instances.

the average optimality gap was below %5, with the maximum optimality gap being around %11. Moreover, the branch-and-price algorithm found a feasible solution for all the instances considered. We finally remark that, as observed by [Pisinger 2005], introducing correlation between the parameters of the problem renders the solution more difficult. Indeed, starting from 60-item instances no correlated instances were solved to optimality within the time limit.

	$I = 20$	$I = 30$	$I = 40$	$I = 50$	$I = 60$	$I = 70$	$I = 80$
#Opt	72	65	49	35	22	18	18
AvgGap(%)	0	3.24	3.37	3.92	3.43	4.16	4.49
MaxGap(%)	0	6.75	7.8	9.44	11.21	10.97	9.91

Table 3.2.3: Summary of results for the branch-and-price algorithm: number of instances solved to optimality within the time limit (#Opt), average (AvgGap) and maximum (MaxGap) optimality gaps for unsolved instances.

We conclude this section with an analysis of the value gained by considering an exact solution method rather than the $K-$adaptability approach. In Table 3.2.4, we present the percentage difference between the objective values of the best primal solution found by the branch-and-price algorithm versus the $K-$Adaptability methods at the end of the time limit. This difference is calculated as $2 * \frac{z_{\text{CG}} - z_{\text{K}}}{z_{\text{CG}} + z_{\text{K}}}$, and is reported only for those instances where the optimal branch-and-price solution had at least 2 active columns. Based on these results, the average percentage gap is below %1, whereas the maximum gap is around %2 for the 2-Adaptability method. One can expect the gaps to be smaller for 3- and 4-adaptable solutions when the method is given enough time to converge.

Although the gains reported in Table 3.2.4 are rather small, we would expect this number to increase as the number of items increases. Indeed, an indicator of the difference between $K-$adaptability and an exact approach is the number of active columns (*i.e.*, second-stage solutions that are active for the optimal solution of the adversarial problem). In our experiments, for 20-item instances, most exact solutions used less than 9 active columns. On the other hand, for larger instances, the best primal solution of the branch-and-price algorithm mostly used between 10 and 40 active columns.

	Average		Max	
	$I = 20$	$I = 30$	$I = 20$	$I = 30$
2-AdaptM	0.78	0.75	1.98	2.12
3-AdaptM	0.67	-	4.19	-
4-AdaptM	0.79	-	4.81	-
2-AdaptBB	0.78	0.73	1.98	2.35
3-AdaptBB	0.41	0.54	1.39	2.29
4-AdaptBB	0.31	0.53	1.40	2.35

Table 3.2.4: Percentage profit gain by an exact approach when the number of active columns is at least 2.

3.2.4.2 Capital budgeting problem

Consider an investment planning problem where a company can allocate an investment budget of B to a subset of projects $i \in \mathcal{N} = \{1, \ldots, N\}$ with possible extensions to the budget with loans. Each project $i \in \mathcal{N}$ has cost c_i and uncertain profit $\tilde{p}_i$ which is modeled as a function of uncertain vector $\boldsymbol{\xi} \in \Xi$ of risk factors. The company can invest in a project before or after observing the risk factors $\boldsymbol{\xi} \in \Xi$. We assume that it is also possible to obtain loans before or after the realization of uncertainty, however the interest rate will be higher in the latter case. Here, we suppose that there is one loan option each before and after the realization of uncertainty, for an amount of C_1 and C_2, respectively. To model the uncertainty associated with this problem, we assume M risk factors that reside in the hyper-rectangle $\Xi = [-1, 1]^M$ with $M << N$. We model the project profits as affine functions of these factors, $\tilde{p}_i(\boldsymbol{\xi}) = (1 + Q_i^\top \boldsymbol{\xi}/2)\bar{p}_i$. Here $\bar{p}_i$ is the nominal cost of project i and Q_i represents the i^{th} row of the factor loading matrix $Q \in \mathbb{R}^{N \times M}$ as a column vector. Early investments enjoy a first-mover advantage whereas a postponed investment in project i generates only a fraction $f \in [0, 1)$ of the profits $\tilde{p}_i$. An initial formulation of the problem is given as

$$\max_{(\boldsymbol{x}, x_0) \in \mathcal{X}} -\lambda x_0 + \sum_{i \in \mathcal{N}} \bar{p}_i (x_i + f y_i) + \min_{\boldsymbol{\xi} \in \Xi} \max_{(\boldsymbol{y}, y_0) \in \mathcal{Y}(\boldsymbol{x})} \sum_{i \in \mathcal{N}} \left(\sum_{j=1}^{M} \frac{Q_{i,j} \xi_j}{2} \right) \bar{p}_i (x_i + f y_i) - \lambda \mu y_0$$

with $\mu > 1$, where $\mathcal{X} = \left\{ (\boldsymbol{x}, x_0) \in \{0, 1\}^{N+1} \mid \boldsymbol{c}^\top \boldsymbol{x} \leq B + C_1 x_0 \right\}$ and

$$\mathcal{Y}(\boldsymbol{x}) = \left\{ (\boldsymbol{y}, y_0) \in \{0, 1\}^{N+1} \;\middle|\; \begin{array}{ll} \boldsymbol{c}^\top \boldsymbol{y} - C_2 y_0 \leq B + C_1 x_0 - \boldsymbol{c}^\top \boldsymbol{x} & \\ y_i \leq 1 - x_i & \forall i \in \mathcal{N} \end{array} \right\}.$$

It is easy to verify in this case that $\text{conv}(\mathcal{Y}(\boldsymbol{x})) \neq \bar{\mathcal{Y}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{X}$. We alternatively describe how columns can be generated in an extended space to create a relaxation $\bar{\mathcal{Y}}(\boldsymbol{x})$ that is exact. To this end, we begin with reformulating the recourse

problem above. We write

$$\mathcal{Y}'(\boldsymbol{x}) = \left\{ (\boldsymbol{y}, y_0) \in \{0,1\}^{N+1} \;\middle|\; \begin{array}{ll} \boldsymbol{c}^\top \boldsymbol{y} \leq B + C_1 x_0 + C_2 y_0 & \\ y_i \geq x_i & \forall i \in \mathcal{N} \end{array} \right\}.$$

In the set $\mathcal{Y}'(\boldsymbol{x})$, variable y_i takes value 1 if the corresponding first-stage variable x_i is equal to 1, therefore implicitly counting for the budget already allocated to first-stage investments. With this redefinition of the set $\mathcal{Y}(\boldsymbol{x})$, we also change the objective function to replace variable y_i with the difference $y_i - x_i$ to differentiate between investment decisions in the first and second stage. The problem then becomes

$$\begin{aligned} \max_{(\boldsymbol{x},x_0)\in\mathcal{X}} -\lambda x_0 + \sum_{i\in\mathcal{N}} \bar{p}_i((1-f)x_i + f y_i) \\ + \min_{\boldsymbol{\xi}\in\Xi} \max_{(\boldsymbol{y},y_0)\in\mathcal{Y}'(\boldsymbol{x})} \sum_{i\in\mathcal{N}} \left(\sum_{j=1}^{M} \frac{Q_{i,j}\xi_j}{2} \right) \bar{p}_i((1-f)x_i + f y_i) - \lambda\mu y_0. \end{aligned}$$

We next proceed to extending the recourse feasible region so as to include a copy of the variable x_0 reposing on the results of Section 3.2.2.3. To this end we may write

$$\mathcal{Y}''(\boldsymbol{x}) = \left\{ (\boldsymbol{y}, y_0, z_0) \in \{0,1\}^{N+2} \;\middle|\; \begin{array}{ll} \boldsymbol{c}^\top \boldsymbol{y} \leq B + C_1 z_0 + C_2 y_0 & \\ y_i \geq x_i & \forall i \in \mathcal{N} \\ z_0 = x_0 & \end{array} \right\},$$

therefore defining the budget constraint $c^\top \boldsymbol{y} \leq B + C_1 z_0 + C_2 y_0$ purely in terms of recourse variables. Let the extreme point solutions of the set $\mathcal{Y}'' = \left\{ (\boldsymbol{y}, y_0, z_0) \in \{0,1\}^{N+2} \middle| c^\top \boldsymbol{y} \leq B + C_1 z_0 + C_2 y_0 \right\}$ be denoted by $(\bar{\boldsymbol{y}}, \bar{y}_0, \bar{z}_0)^l$ for $l \in \mathcal{L} = \{1, \ldots, L\}$. When applying the branch-and-price algorithm to this modified problem, the budget constraint $\boldsymbol{c}^\top \boldsymbol{y} \leq B + C_1 x_0 + C_2 y_0$ is removed from the master problem while the constraint $x_0 = \sum_{l\in\mathcal{L}} \alpha^l z_0^l$ is added. Under this reformulation the linking constraints are $x_0 = \sum_{l\in\mathcal{L}} \alpha^l z_0^l$ and $\sum_{l\in\mathcal{L}} \alpha^l y_i^l \geq x_i$ for $i \in \mathcal{N}$, and they follow the assumptions of Proposition 3.2.4. We may therefore obtain an optimal solution of the capital budgeting problem by directly solving a deterministic equivalent model based on the set $\bar{\mathcal{Y}}''(\boldsymbol{x})$ with the branch-and-price algorithm without the need to add any cuts.

We additionally test a K-Adaptability approach to this problem, the associated model can be found in [Arslan & Detienne 2021].

Instance generation For our numerical tests, we generate instances inspired by those presented in [Hanasusanto *et al.* 2015]. For given number of items N, and parameters $R = 100$ and $H = 100$, $h \in \{20, 40, 60, 80\}$, we generate the costs c_i for $i \in \mathcal{N}$ uniformly from the interval $[1, R]$, and we set the nominal profits $\bar{\boldsymbol{p}} = \boldsymbol{c}/5$. The components in each row of Q are generated uniformly from the unit simplex in $\mathbb{R}^M$, which implies that the profits of each project can deviate up to %50 from their nominal values. We set the investment budget to $B = \frac{h}{H+1} \sum_{i\in\mathcal{I}} c_i$ and we assume

that postponed investments only generate %80 of the profits, that is $f = 0.8$. The loans C_1 and C_2 are set to %20 of the initial budget B, and the cost parameters are chosen as $\lambda = \frac{0.12}{5}$ and $\mu = 1.2$.

Results In this section, we present our results on instances generated as described above. We solve instances with $N \in \{10, 20, 30, 40, 50, 100\}$ and generate 60 instances for each value of N, corresponding to 5 replications for each combination of the parameters $h \in \{20, 40, 60, 80\}$ and $M \in \{4, 6, 8\}$. All instances are solved with a 1-hour time limit.

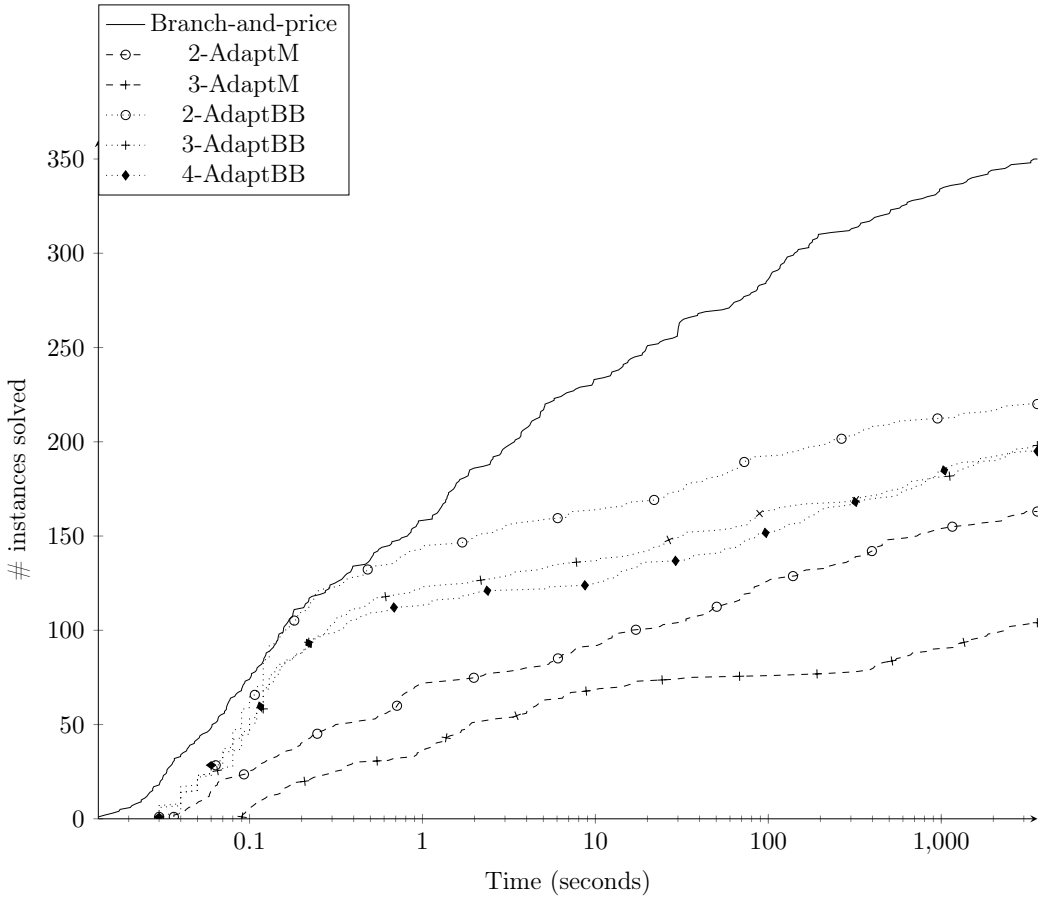


Figure 3.2.8: Performance profile comparing all three methods for all instances for the Robust Capital Budgeting problem.

We first present in Table 3.2.5 the results comparing the number of instances for which each method has converged. Similar to our results for the robust knapsack instances, our results reveal the column generation-based approach to be very effective for this problem. The exact solution method scales up very well and is able to solve more than %95 of the instances considered. It is also 3 to 4 orders of magnitude faster than the $K-$adaptability methods for larger instances. This can be seen from the performance profile presented in Figure 3.2.8 where the number

of instances for which each method has converged is plotted over time presented in logarithmic scale. Additionally, the branch-and-price algorithm found a feasible solution for all the instances considered.

	$N = 10$	$N = 20$	$N = 30$	$N = 40$	$N = 50$	$N = 100$
B&P	60	60	58	55	60	57
2-AdaptM	60	60	21	15	7	0
3-AdaptM	60	31	13	0	0	0
2-AdaptBB	60	54	30	28	20	28
3-AdaptBB	59	37	25	19	23	35
4-AdaptBB	58	27	25	24	26	35

Table 3.2.5: Number of instances solved to convergence by each method.

Finally, similar to what was done for the knapsack instances, we investigate the number of active columns in the best primal solution reported by the branch-and-price algorithm. In all the instances we considered the number of active columns required was smaller than 10 (although for most large instances at least 5 columns were required). Additionally, for instances with smaller number of items, the percentage of instances that required 3 columns or less was high. Accordingly, one would expect the $K-$adaptability methods to provide very good approximations for the instances considered. Indeed, our results confirmed that the 2-Adaptability gaps were always below %1 on average (except for $N = 10$) although higher maximum gaps were present.

3.3 Other contributions in optimization under uncertainty

Extension of the convexification approach to handle arbitrary linking constraints and convex non-linear problems In [Detienne *et al.* Submitted], we extend the results of [Arslan & Detienne 2021] with help of Fenchel duality and spatial branching.

We study optimization problems where some cost parameters are not known at decision time and the decision flow is modeled as a two-stage process within a robust optimization setting. We address general problems in which all constraints (including those linking the first and the second stages) are defined by convex functions and involve mixed-integer variables, thus extending the existing literature to a much wider class of problems. We show how these problems can be reformulated using Fenchel duality, allowing to derive an enumerative exact algorithm, for which we prove-convergence in a finite number of operations. An implementation of the resulting algorithm, embedding a column generation scheme, is then computationally evaluated on two different problems, using instances that are derived starting from the existing literature. To the best of our knowledge, this is the first approach providing results on the practical solution of this class of problems.

Application of the convexification approach to two-stage robust scheduling Minimizing the weighted number of tardy jobs is a classical and intensively studied scheduling problem. In [Lefebvre *et al.* Under revision], we develop a two-stage robust approach, where exact weights are known after accepting to perform the jobs, and before sequencing them on the machine. This assumption allows diverse recourse decisions to be taken in order to better adapt one's tactical plan.

The contribution of this paper is twofold: first, we introduce a new scheduling problem and model it as a min-max-min optimization problem with mixed-integer recourse by extending existing models proposed for the classical problem where all the costs are assumed to be known. Second, we take advantage of the special structure of the problem to propose two solution approaches based on results from the recent robust optimization literature, namely finite adaptability [Bertsimas & Caramanis 2010] and a convexification-based approach [Arslan & Detienne 2021]. We also study the cost of fixing the sequence of jobs before the uncertainty is revealed. Computational experiments to analyze the effectiveness of our approaches are reported.

Application of robust optimization in agriculture [Arslan & Detienne In revision] deals with robust planning and scheduling of activities in agriculture and in particular the application of phytosanitary treatments. The crops are subject to many diseases that may arise during different time windows of the planning horizon. In response, a phytosanitary treatment can be applied to protect against a subset of these diseases. However, the effective duration of some treatments is uncertain, it depends on the type of treatment applied as well as on the weather conditions. In this study we introduce a penalty function based approach to handle this uncertainty without being overly conservative akin to light robustness approach proposed in the literature. We discuss different forms for this penalty function and elaborate on solution methodologies for the resulting models. We test the effectiveness of our approach with realistically-sized instances, which correspond to a typical vineyard in Bordeaux area, and present a numerical analysis of different optimization models and solution methods.

Acceleration of Benders decomposition [Blanchot *et al.* Submitted] introduces a new exact algorithm based on Benders decomposition to solve two-stage stochastic linear programs. We propose to solve only a small number of subproblems at each iteration, and develop a simple and exact framework thanks to the multicut formulation of Benders decomposition. We propose two primal stabilization methods for the algorithm and perform an extensive computational study on six large-scale benchmarks from the stochastic optimization literature. Results show the efficiency of the method compared to five classical alternative algorithms and significant time savings provided by its primal stabilization. We show acceleration factors up to 10 times faster than the best method from the literature we use for comparison, and up to 800 times faster than IBM ILOG CPLEX 12.10 built-in

Benders decomposition.

Bilevel programming with risk-averse stochastic objective for energy In [van Ackooij *et al.* 2018] we consider energy management optimization problems in a future wherein an interaction with micro-grids has to be accounted for. We will model this interaction through a set of contracts between the generation companies owning centralized assets and the micro-grids. We will formulate a general stylized model that can, in principle, account for a variety of management questions such as unit-commitment. The resulting model, a bilevel stochastic mixed integer program will be numerically tackled through a novel preprocessing procedure. As a result the solution for the bilevel (or single leader multiple follower) problem will be neither "optimistic" nor "pessimistic". We will numerically evaluate the difference of the resulting solution with the "optimistic" solution. We will also demonstrate the efficiency and potential of our methodology on a set of numerical instances.

Chapter 4

Perspectives

The work about robust optimization started a few years ago raises many questions.

In the short term, robust optimization will be studied with the aim of making contributions from an application point-of-view.

- In the fields of reliable network design and post-disaster management where human lives are engaged, worst-case optimization is particularly relevant. This study will start in the framework of ANR project DESIDE, in collaboration with Sobolev Institute and Kedge Business School and with the PhD thesis of Komlanvi Parfait Ametana. The most appropriate settings of network-design problems will be determined. Easy and hard cases will be identified, depending on the set of constraints, the definition of the uncertainty sets and the possible second-stage actions, with a focus on integer recourse. Our objectives are to propose solution approaches (based on the robust literature in general), but also to identify loopholes in current methodologies that need to be closed to effectively address these kinds of problems.

- With the combination of the shortage of resources and the urging need for more sustainable societies, the efficient production of energy is becoming more and more important. This motivates research about the stability of the power plant schedules in case of random events. To address these kinds of problems, progress will have to be made more generally on the stability of solutions of combinatorial problems when the uncertainty affects the structure of the instance (for example, in case of significant change in a state/transition graph). The first steps on this path will be made in collaboration with EDF through a grant from the PGMO funding program.

In the medium term, methodological advances are sought, mainly pursuing the work about convexification approaches.

- Constraint uncertainty in two-stage robust programs will be studied under the angle of Lagrange and Fenchel duality. Several issues need to be resolved investigating this approach, such as the optimization of the (potentially infinite size-) dual problems and how to close the duality gap.

- $K-$adaptability [Hanasusanto *et al.* 2015] or *min-max-min* approaches [Buchheim & Kurtz 2017] make perfect sense in some decision processes. Moreover, our results on fully adjustable problems show that $K-$adaptability approaches also provide excellent heuristic solutions [Arslan & Detienne 2021, Lefebvre

et al. Under revision]. However, existing approaches are limited when the number K of allowed second-stage solutions increases. This invites to investigate convexification techniques for these types of problems.

In the long term, more theoretical questions and relations with related scientific fields add up to the methodological challenges.

- Under the technical restrictions considered in [Arslan & Detienne 2021], the problems are proven to be $\mathcal{NP}$-complete. We aim at better understanding where the frontier is between $\mathcal{NP}$-complete robust optimization problems with integer recourse and those that are (probably) outside the class $\mathcal{NP}$. Also, is this frontier tangible in terms of numerical difficulty?

- Some similar concepts are used in combinatorial game theory and robust optimization (both solve min-max-min... problems). Bridging the gap between multi-stage integer robust optimization and combinatorial games might allow to disseminate our techniques in another field. But above all, learning about this domain of computer science could help answering the questions about the complexity of robust problems.

- Exactly solving the general version of two-stage or even multi-stage robust mixed integer programs is an exciting challenge, that could help in the realistic contexts described for the short-term work plan above. Although our tool of choice is convexification today, an alternative – hopefully complementary – point-of-view on those problems from the combinatorial game theory perspective will be investigated, for the purpose of creating synergies between its techniques and those of mathematical programming.

It seems clear that SSR methods work well when the problem is structured around a single sequential decision process, when one can build strong relaxations based on an LP reformulation. Assuming that a good primal bound is known, variable fixing techniques help reduce the size of the manipulated models. Still, answering the following questions would make these types of methods more effective and more popular.

- In Iterative State-Space Relaxation algorithms, there remains an important ingredient to make the method effective even when the bounds are not strong: which constraints should be added to the DP when the solution of the current relaxation is not feasible? In the literature, a compromise between the increase in the dual bound and the increase in the size of the model is heuristically found for each problem. Finding the best set of constraints to reintroduce seems a very hard problem, akin to finding the best branching choices in a branch-and-bound search. Several lines of research can be followed. Unifying SSDP algorithm with Decision Diagrams would bring techniques to control the size of the DP at each iteration, among other things. First results for predicting the size of the model are proposed in [Clautiaux *et al.* 2021], but more precise estimation, in a general setting and without building the corresponding

DP, a priori requires an in-depth understanding of the structure of the recurrence equations and is very likely to be an $\mathcal{NP}$-hard problem. Assuming a satisfactory answer to this question is known, choosing the constraints to add, that will yield the best dual bound at the next iteration can be viewed as an *interdiction problem*: the problem of the attacker is to build the network of limited size and that satisfies some construction rules such that the cost of the shortest path of the defender is maximized. Pushing the logic of variable fixing, a last line of research would be to find synergies with CP techniques.

- Subgradient-based algorithms are easy to implement and provide good solutions to the Lagrangian dual problem – provided that they are well-tuned. But implementing Iterative State-Space Relaxation methods also require the management of very large DP models. This is definitely a hurdle to the testing of this type of algorithmic strategies. A generic and effective implementation of such methods could help researchers in this task. The implementation of such a framework has started a few years ago, in collaboration with François Clautiaux, Aurélien Froger and Gaël Guillot, leading to the numerical results reported in Section 2.2. This generic code also allowed benchmarking the DP-based reformulation (Section 3.2.2.5) for the robust knapsack problem using only 350 additional lines of C++ code, in order to input the recurrence relations and additional constraints. Only 60 more lines were necessary to setup a generic DP pricing oracle and connect with the branch-and-price library BapCod (also developed at Inria team RealOpt), in order to obtain a first version of the branch-and-price algorithm proposed in Section 3.2.4. The generic algorithms developed within this Iterative State-Space Relaxation (ISSR) framework are, unfortunately, still not competitive with dedicated implementations like those described in [Tanaka & Fujikuma 2012, Detienne *et al.* 2016], or like the ad-hoc label-correcting algorithms developed for the final version of the pricing oracles reported in Section 3.2.4. Writing a generic code that allows the efficient use of specific dominance rules requires a considerable software engineering work. Achieving this goal would allow releasing an interesting tool for the community of operations researchers.

Bibliography

[Abdul-Razaq & Potts 1988] T. S. Abdul-Razaq and Chris N. Potts. *Dynamic Programming State-Space Relaxation for Single-Machine Scheduling.* Journal of the Operational Research Society, vol. 39, no. 2, pages 141–152, 1988. (Cited on pages 35, 50 and 53.)

[Absi *et al.* 2013] Nabil Absi, Boris Detienne and Stéphane Dauzère-Pérès. *Heuristics for the multi-item capacitated lot-sizing problem with lost sales.* Computers & Operations Research, vol. 40, no. 1, pages 264–272, 2013. (Cited on pages 28 and 94.)

[Ahmed & Shapiro 2008] Shabbir Ahmed and Alexander Shapiro. *Solving Chance-Constrained Stochastic Programs via Sampling and Integer Programming.* In State-of-the-Art Decision-Making Tools in the Information-Intensive Age, INFORMS TutORials in Operations Research, pages 261–269. INFORMS, September 2008. Section: 12. (Cited on page 6.)

[Akkan & Karabati 2004] Can Akkan and Selçuk Karabati. *The two-machine flow-shop total completion time problem: Improved lower bounds and a branch-and-bound algorithm.* European Journal of Operational Research, vol. 159, no. 2, pages 420–429, December 2004. (Cited on pages 39, 42, 43, 44, 47, 48, 49 and 58.)

[Aksen *et al.* 2003] Deniz Aksen, Kemal Altınkemer and Suresh Chand. *The single-item lot-sizing problem with immediate lost sales.* European Journal of Operational Research, vol. 147, no. 3, pages 558–566, June 2003. (Cited on page 94.)

[Allahverdi *et al.* 1999] Ali Allahverdi, Jatinder N.D Gupta and Tariq Aldowaisan. *A review of scheduling research involving setup considerations.* Omega, vol. 27, no. 2, pages 219 – 239, 1999. (Cited on page 43.)

[Allahverdi 2000] Ali Allahverdi. *Minimizing mean flowtime in a two-machine flow-shop with sequence-independent setup times.* Computers and Operations Research, vol. 27, no. 2, pages 111 – 127, 2000. (Cited on page 45.)

[Arkin & Silverberg 1987] E. M. Arkin and E.B. Silverberg. *Scheduling with fixed start and end times.* Discrete Applied Mathematics, vol. 18, pages 1–8, 1987. (Cited on page 64.)

[Arrow *et al.* 1958] Kenneth J Arrow, Leonid Hurwicz and Hirofumi Uzawa. Studies in linear and non-linear programming. Stanford Math. Stud. Social Sci. Cambridge Univ. Press, 1958. (Cited on page 97.)

[Arslan & Detienne 2021] Ayşe Arslan and Boris Detienne. *Decomposition-based approaches for a class of two-stage robust binary optimization problems.* INFORMS Journal on Computing, 2021. (Cited on pages 16, 18, 34, 130, 157, 163, 165, 166, 169 and 170.)

[Arslan & Detienne In revision] Ayşe Arslan and Boris Detienne. *Robust Strategic Planning of Phytosanitary Treatments in Agriculture.* November In revision. (Cited on page 166.)

[Artzner *et al.* 1999] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber and David Heath. *Coherent Measures of Risk.* Mathematical Finance, vol. 9, no. 3, pages 203–228, 1999. (Cited on page 5.)

[Atamtürk & Gomez 2020] Alper Atamtürk and Andres Gomez. *Safe screening rules for L0-regression from Perspective Relaxations.* In Proceedings of the 37th International Conference on Machine Learning, pages 421–430. PMLR, November 2020. ISSN: 2640-3498. (Cited on page 27.)

[Atamtürk & Zhang 2007] Alper Atamtürk and Muhong Zhang. *Two-stage robust network flow and design under demand uncertainty.* Operations Research, vol. 55, no. 4, pages 662–673, 2007. (Cited on page 135.)

[Ayoub & Poss 2016] Josette Ayoub and Michael Poss. *Decomposition for adjustable robust linear optimization subject to uncertainty polytope.* Computational Management Science, vol. 13, no. 2, pages 219–239, 2016. (Cited on page 135.)

[Baptiste *et al.* 2004] Ph. Baptiste, J. Carlier and A. Jouglet. *A Branch-and-Bound Procedure to Minimize Total Tardiness on One Machine with Arbitrary Release Dates.* European Journal of Operational Research, vol. 158, no. 3, pages 595–608, 2004. (Cited on page 56.)

[Barahona & Anbil 2000] Francisco Barahona and Ranga Anbil. *The volume algorithm: producing primal solutions with a subgradient method.* Mathematical Programming, vol. 87, no. 3, pages 385–399, May 2000. (Cited on pages 23, 25 and 73.)

[Bard 1991] J. F. Bard. *Some properties of the bilevel programming problem.* Journal of Optimization Theory and Applications, vol. 68, no. 2, pages 371–378, February 1991. (Cited on page 3.)

[Bartlett *et al.* 2005] Mark Bartlett, Alan M. Frisch, Youssef Hamadi, Ian Miguel, S. Armagan Tarim and Chris Unsworth. *The Temporal Knapsack Problem and Its Solution.* In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3524, pages 34–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. (Cited on page 64.)

[Beale 1955] Evelyn ML Beale. *On minimizing a convex function subject to linear inequalities.* Journal of the Royal Statistical Society: Series B (Methodological), vol. 17, no. 2, pages 173–184, 1955. (Cited on page 8.)

[Belgacem & Amir 2018] Rachid Belgacem and Abdessamad Amir. *A new modified deflected subgradient method.* Journal of King Saud University - Science, vol. 30, no. 4, pages 561–567, October 2018. (Cited on page 25.)

[Bellman 1954] Richard Ernest Bellman. *The Theory of Dynamic Programming.* Technical report, January 1954. (Cited on page 13.)

[Ben-Tal & Nemirovski 1999] A. Ben-Tal and A. Nemirovski. *Robust solutions of uncertain linear programs.* Operations Research Letters, vol. 25, no. 1, pages 1–13, August 1999. (Cited on page 8.)

[Ben-Tal & Nemirovski 2000] Aharon Ben-Tal and Arkadi Nemirovski. *Robust solutions of Linear Programming problems contaminated with uncertain data.* Mathematical Programming, vol. 88, no. 3, pages 411–424, September 2000. (Cited on page 3.)

[Ben-Tal *et al.* 2004] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer and Arkadi Nemirovski. *Adjustable robust solutions of uncertain linear programs.* Mathematical Programming, vol. 99, no. 2, pages 351–376, 2004. (Cited on pages 9, 132 and 133.)

[Ben-Tal *et al.* 2009] Aharon Ben-Tal, Laurent El Ghaoui and Arkadi Nemirovski. Robust optimization, volume 28. Princeton University Press, 2009. (Cited on pages 4, 6, 7 and 132.)

[Benders 1962] Jacques F Benders. *Partitioning procedures for solving mixed-variables programming problems.* Numerische mathematik, vol. 4, no. 1, pages 238–252, 1962. (Cited on pages 18 and 112.)

[Benkirane *et al.* 2020] Mohamed Benkirane, François Clautiaux, Jean Damay and Boris Detienne. *A Hypergraph Model for the Rolling Stock Rotation Planning and Train Selection.* Technical report, December 2020. (Cited on pages 37 and 94.)

[Bergner *et al.* 2015] Martin Bergner, Alberto Caprara, Alberto Ceselli, Fabio Furini, Marco E. Lübbecke, Enrico Malaguti and Emiliano Traversi. *Automatic Dantzig–Wolfe reformulation of mixed integer programs.* Mathematical Programming, vol. 149, no. 1, pages 391–424, February 2015. (Cited on page 18.)

[Bertsekas 2015] Dimitri P Bertsekas. Convex optimization algorithms. Athena Scientific Belmont, 2015. (Cited on page 73.)

[Bertsimas & Caramanis 2010] Dimitris Bertsimas and Constantine Caramanis. *Finite adaptability in multistage linear optimization.* IEEE Transactions on Automatic Control, vol. 55, no. 12, pages 2751–2766, 2010. (Cited on pages 9, 134 and 166.)

[Bertsimas & Dunning 2016] Dimitris Bertsimas and Iain Dunning. *Multistage robust mixed-integer optimization with adaptive partitions.* Operations Research, vol. 64, no. 4, pages 980–998, 2016. (Cited on page 135.)

[Bertsimas & Georghiou 2015] Dimitris Bertsimas and Angelos Georghiou. *Design of near optimal decision rules in multistage adaptive mixed-integer optimization.* Operations Research, vol. 63, no. 3, pages 610–627, 2015. (Cited on page 134.)

[Bertsimas & Georghiou 2018] Dimitris Bertsimas and Angelos Georghiou. *Binary decision rules for multistage adaptive mixed-integer optimization.* Mathematical Programming, vol. 167, no. 2, pages 395–433, 2018. (Cited on pages 38 and 134.)

[Bertsimas & Sim 2004] Dimitris Bertsimas and Melvyn Sim. *The price of robustness.* Operations research, vol. 52, no. 1, pages 35–53, 2004. (Cited on page 8.)

[Bertsimas *et al.* 2011] Dimitris Bertsimas, David B Brown and Constantine Caramanis. *Theory and applications of robust optimization.* SIAM review, vol. 53, no. 3, pages 464–501, 2011. (Cited on pages 6 and 132.)

[Bertsimas *et al.* 2013a] Dimitris Bertsimas, Eugene Litvinov, Xu Andy Sun, Jinye Zhao and Tongxin Zheng. *Adaptive robust optimization for the security constrained unit commitment problem.* IEEE Transactions on Power Systems, vol. 28, no. 1, pages 52–63, 2013. (Cited on page 135.)

[Bertsimas *et al.* 2013b] Dimitris Bertsimas, Ebrahim Nasrabadi and Sebastian Stiller. *Robust and Adaptive Network Flows.* Operations Research, vol. 61, no. 5, pages 1218–1242, October 2013. (Cited on pages 9 and 153.)

[Birge & Louveaux 2011] John R Birge and Francois Louveaux. Introduction to stochastic programming. Springer Science & Business Media, 2011. (Cited on pages 4, 6, 7, 8, 21 and 108.)

[Blanchot *et al.* Submitted] Xavier Blanchot, François Clautiaux, Boris Detienne, Aurélien Froger and Manuel Ruiz. *The Benders by batch algorithm: design and stabilization of an enhanced algorithm to solve multicut Benders reformulation of two-stage stochastic programs.* Submitted. (Cited on pages 21 and 166.)

[Boland *et al.* 2006] Natashia Boland, John Dethridge and Irina Dumitrescu. *Accelerated Label Setting Algorithms for the Elementary Resource Constrained*

Shortest Path Problem. Operations Research Letters, vol. 34, no. 1, pages 58–68, 2006. (Cited on page 35.)

[Bonsma *et al.* 2014] P.S. Bonsma, Jens Schulz and Andreas Wiese. *A Constant-Factor Approximation Algorithm for Unsplittable Flow on Paths.* SIAM journal on computing, vol. 43, no. 2, pages 767–799, 2014. (Cited on page 64.)

[Brandt 2010] Felix Brandt. Solving a large-scale energy management problem with varied constraints. Master's thesis, Karlsruhe Institute of Technology (KIT), Faculty of Informatics., Germany, 2010. (Cited on page 97.)

[Brucker 2004a] Peter Brucker. *Single Machine Scheduling Problems.* In Peter Brucker, editor, Scheduling Algorithms, pages 61–106. Springer, Berlin, Heidelberg, 2004. (Cited on page 13.)

[Brucker 2004b] Peter Brucker. *Some Problems in Combinatorial Optimization.* In Peter Brucker, editor, Scheduling Algorithms, pages 11–36. Springer, Berlin, Heidelberg, 2004. (Cited on page 13.)

[Bryant 1986] Bryant. *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Transactions on Computers, vol. C-35, no. 8, pages 677–691, August 1986. Conference Name: IEEE Transactions on Computers. (Cited on page 13.)

[Bucarey *et al.* 2022] Víctor Bucarey, Bernard Fortz, Natividad González-Blanco, Martine Labbé and Juan A. Mesa. *Benders decomposition for network design covering problems.* Computers & Operations Research, vol. 137, page 105417, January 2022. (Cited on page 21.)

[Buchheim & Kurtz 2017] Christoph Buchheim and Jannis Kurtz. *Min–max–min robust combinatorial optimization.* Mathematical Programming, vol. 163, no. 1-2, pages 1–23, 2017. (Cited on pages 134 and 169.)

[Bulhões *et al.* 2020] Teobaldo Bulhões, Ruslan Sadykov, Anand Subramanian and Eduardo Uchoa. *On the exact solution of a large class of parallel machine scheduling problems.* Journal of Scheduling, vol. 23, no. 4, pages 411–429, August 2020. (Cited on page 18.)

[Calinescu *et al.* 2002] G. Calinescu, A. Chakrabarti, H.J. Karloff and Y. Rabani. *Improved approximation algorithms for resource allocation.* In Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2002, page 401–414. Springer-Verlag, 2002. (Cited on page 64.)

[Caprara *et al.* 2013] A. Caprara, F. Furini and E. Malaguti. *Uncommon Dantzig-Wolfe Reformulation for the Temporal Knapsack Problem.* INFORMS Journal on Computing, vol. 25, no. 3, pages 560–571, 2013. (Cited on pages 64, 65, 66, 67, 86 and 92.)

[Caprara *et al.* 2016] A. Caprara, F. Furini, E. Malaguti and E. Traversi. *Solving the Temporal Knapsack Problem via Recursive Dantzig–Wolfe Reformulation.* Information Processing Letters, vol. 116, no. 5, pages 379 – 386, 2016. (Cited on pages 65, 66 and 87.)

[Carlier & Pinson 1989] J. Carlier and E. Pinson. *An Algorithm for Solving the Job-Shop Problem.* Management Science, vol. 35, no. 2, pages 164–176, 1989. Publisher: INFORMS. (Cited on page 34.)

[Carlier & Pinson 1994] J. Carlier and E. Pinson. *Adjustment of heads and tails for the job-shop problem.* European Journal of Operational Research, vol. 78, no. 2, pages 146–161, October 1994. (Cited on page 27.)

[Carlier *et al.* 2004] Jacques Carlier, Laurent Péridy, E Pinson and David Rivreau. *Elimination rules for job-shop scheduling problem: overview and extensions.* In Handbook of Scheduling: Algorithms, models, and performance analysis. Chapman & Hall/CRC, 2004. (Cited on page 27.)

[Charnes & Cooper 1959] A. Charnes and W. W. Cooper. *Chance-Constrained Programming.* Management Science, vol. 6, no. 1, pages 73–79, October 1959. Publisher: INFORMS. (Cited on page 6.)

[Chassein *et al.* 2019] André Chassein, Marc Goerigk, Jannis Kurtz and Michael Poss. *Faster Algorithms for Min-max-min Robustness for Combinatorial Problems with Budgeted Uncertainty.* European Journal of Operational Research, 2019. (Cited on page 135.)

[Chen & Zhang 2009] Xin Chen and Yuhan Zhang. *Uncertain linear programs: Extended affinely adjustable robust counterparts.* Operations Research, vol. 57, no. 6, pages 1469–1482, 2009. (Cited on page 133.)

[Chen *et al.* 2002] B. Chen, Refael Hassin and Michal Tzur. *Allocation of bandwidth and storage.* IIE Transactions, vol. 34, no. 5, pages 501–507, 2002. (Cited on pages 64, 67 and 68.)

[Chen *et al.* 2008] Xin Chen, Melvyn Sim, Peng Sun and Jiawei Zhang. *A linear decision-based approximation approach to stochastic programming.* Operations Research, vol. 56, no. 2, pages 344–357, 2008. (Cited on page 133.)

[Christofides *et al.* 1981] Nicos Christofides, A. Mingozzi and P. Toth. *State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems.* Networks, vol. 11, no. 2, pages 145–164, 1981. (Cited on pages 25, 26 and 35.)

[Clautiaux *et al.* 2017] François Clautiaux, Saïd Hanafi, Rita Macedo, Marie-Émilie Voge and Cláudio Alves. *Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints.* European Journal of Operational Research, vol. 258, no. 2, pages 467–477, 2017. (Cited on page 153.)

[Clautiaux *et al.* 2018] François Clautiaux, Ruslan Sadykov, François Vanderbeck and Quentin Viaud. *Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem.* Discrete Optimization, vol. 29, pages 18–44, August 2018. (Cited on page 14.)

[Clautiaux *et al.* 2021] F. Clautiaux, B. Detienne and G. Guillot. *An iterative dynamic programming approach for the temporal knapsack problem.* European Journal of Operational Research, vol. 293, no. 2, pages 442–456, September 2021. (Cited on pages 16, 28, 37, 63, 64 and 170.)

[Conforti *et al.* 2010] Michele Conforti, Gérard Cornuéjols and Giacomo Zambelli. *Polyhedral Approaches to Mixed Integer Linear Programming.* In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi and Laurence A. Wolsey, editors, 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, pages 343–385. Springer, Berlin, Heidelberg, 2010. (Cited on page 34.)

[Conway *et al.* 1967] R. W. Conway, W. L. Maxwell and L. W. Miller. Theory of scheduling. Addison-Wesley, Reading, MA, 1967. (Cited on page 42.)

[Croce *et al.* 1996] F. Della Croce, V. Narayan and R. Tadei. *The two-machine total completion time flow shop problem.* European Journal of Operational Research, vol. 90, no. 2, pages 227 – 237, 1996. (Cited on pages 45 and 48.)

[Crowder *et al.* 1983] Harlan Crowder, Ellis L. Johnson and Manfred Padberg. *Solving Large-Scale Zero-One Linear Programming Problems.* Operations Research, vol. 31, no. 5, pages 803–834, 1983. (Cited on page 27.)

[Dantzig & Wolfe 1960] George B. Dantzig and Philip Wolfe. *Decomposition Principle for Linear Programs.* Operations Research, vol. 8, no. 1, pages 101–111, February 1960. Publisher: INFORMS. (Cited on pages 16 and 17.)

[Dantzig *et al.* 1954] G. Dantzig, R. Fulkerson and S. Johnson. *Solution of a Large-Scale Traveling-Salesman Problem.* Journal of the Operations Research Society of America, vol. 2, no. 4, pages 393–410, 1954. Publisher: INFORMS. (Cited on page 27.)

[Dantzig *et al.* 1955] George B Dantzig, Alex Orden and Philip Wolfe. *The generalized simplex method for minimizing a linear form under linear inequality restraints.* Pacific Journal of Mathematics, vol. 5, no. 2, pages 183–195, 1955. (Cited on page 17.)

[Dantzig 1955] George B. Dantzig. *Linear Programming under Uncertainty.* Management Science, vol. 1, no. 3-4, pages 197–206, April 1955. Publisher: INFORMS. (Cited on pages 3 and 8.)

[Deineko & Woeginger 2010] Vladimir G. Deineko and Gerhard J. Woeginger. *Pinpointing the complexity of the interval min–max regret knapsack problem.* Discrete Optimization, vol. 7, no. 4, pages 191–196, November 2010. (Cited on page 153.)

[Demassey *et al.* 2006] Sophie Demassey, Gilles Pesant and Louis-Martin Rousseau. *A Cost-Regular Based Hybrid Column Generation Approach.* Constraints, vol. 11, no. 4, pages 315–333, November 2006. (Cited on page 27.)

[Dempe 2002] Stephan Dempe. Foundations of bilevel programming. Springer Science & Business Media, 2002. (Cited on page 3.)

[Desaulniers *et al.* 2008] Guy Desaulniers, François Lessard and Ahmed Hadjar. *Tabu Search, Partial Elementarity, and Generalized Fc-Path Inequalities for the Vehicle Routing Problem with Time Windows.* Transportation Science, vol. 42, no. 3, pages 387–404, 2008. (Cited on page 35.)

[Desrochers *et al.* 1992] Martin Desrochers, Jacques Desrosiers and Marius Solomon. *A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows.* Operations Research, vol. 40, no. 2, pages 342–354, April 1992. Publisher: INFORMS. (Cited on page 50.)

[Desrosiers & Lübbecke 2011] Jacques Desrosiers and Marco E Lübbecke. *Branch-price-and-cut algorithms.* Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Chichester, pages 109–131, 2011. (Cited on page 117.)

[Detienne *et al.* 2009] Boris Detienne, Laurent Péridy, Éric Pinson and David Rivreau. *Cut generation for an employee timetabling problem.* European Journal of Operational Research, vol. 197, no. 3, pages 1178–1184, September 2009. (Cited on pages 21 and 28.)

[Detienne *et al.* 2010] Boris Detienne, Éric Pinson and David Rivreau. *Lagrangian domain reductions for the single machine earliness–tardiness problem with release dates.* European Journal of Operational Research, vol. 201, no. 1, pages 45–54, February 2010. (Cited on pages 28 and 33.)

[Detienne *et al.* 2011] Boris Detienne, Stéphane Dauzère-Pérès and Claude Yugma. *Scheduling jobs on parallel machines to minimize a regular step total cost function.* Journal of Scheduling, vol. 14, no. 6, pages 523–538, December 2011. (Cited on page 35.)

[Detienne *et al.* 2012] Boris Detienne, Stéphane Dauzère-Pérès and Claude Yugma. *An exact approach for scheduling jobs with regular step cost functions on a single machine.* Computers & Operations Research, vol. 39, no. 5, pages 1033–1043, May 2012. (Cited on pages 16, 28, 37, 54 and 93.)

[Detienne *et al.* 2016] Boris Detienne, Ruslan Sadykov and Shunji Tanaka. *The two-machine flowshop total completion time problem: Branch-and-bound algorithms based on network-flow formulation.* European Journal of Operational Research, vol. 252, no. 3, pages 750–760, August 2016. (Cited on pages 16, 28, 33, 39, 41, 42 and 171.)

[Detienne *et al.* Submitted] Boris Detienne, Henri Lefebvre, Enrico Malaguti and Michele Monaci. *Adaptive robust optimization with objective uncertainty.* Submitted. (Cited on pages 32 and 165.)

[Detienne 2014] Boris Detienne. *A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints.* European Journal of Operational Research, vol. 235, no. 3, pages 540–552, June 2014. (Cited on pages 35, 39 and 94.)

[Dowsland & Dowsland 1992] Kathryn A. Dowsland and William B. Dowsland. *Packing problems.* European Journal of Operational Research, vol. 56, no. 1, pages 2–14, January 1992. (Cited on page 13.)

[Edwin & Curtius 1990] K.W. Edwin and F. Curtius. *New maintenance-scheduling method with production cost minimization via integer linear programming.* International journal of Electrical Power & Energy Systems, vol. 12, no. 3, pages 165–170, 1990. (Cited on page 97.)

[Elhallaoui *et al.* 2005] Issmail Elhallaoui, Daniel Villeneuve, François Soumis and Guy Desaulniers. *Dynamic Aggregation of Set-Partitioning Constraints in Column Generation.* Operations Research, vol. 53, no. 4, pages 632–645, August 2005. Publisher: INFORMS. (Cited on page 18.)

[Everett 1963] Hugh Everett. *Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources.* Operations Research, vol. 11, no. 3, pages 399–417, June 1963. Publisher: INFORMS. (Cited on page 21.)

[Fischetti *et al.* 2010] Matteo Fischetti, Domenico Salvagnin and Arrigo Zanette. *A note on the selection of Benders' cuts.* Mathematical Programming, vol. 124, no. 1-2, pages 175–182, July 2010. (Cited on page 19.)

[Fischetti *et al.* 2017] Matteo Fischetti, Ivana Ljubić and Markus Sinnl. *Redesigning Benders Decomposition for Large-Scale Facility Location.* Management Science, vol. 63, no. 7, pages 2146–2162, July 2017. Publisher: INFORMS. (Cited on page 21.)

[Fisher 1973] Marshall L. Fisher. *Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I.* Operations Research, vol. 21, no. 5, pages 1114–1127, October 1973. Publisher: INFORMS: Institute for Operations Research. (Cited on page 21.)

[Focacci *et al.* 1999] F. Focacci, A. Lodi and M. Milano. *Cost-Based Domain Filtering.* In Joxan Jaffar, editor, Principles and Practice of Constraint Programming – CP'99, Lecture Notes in Computer Science, pages 189–203, Berlin, Heidelberg, 1999. Springer. (Cited on page 27.)

[Fortz & Poss 2009] B. Fortz and M. Poss. *An improved Benders decomposition applied to a multi-layer network design problem.* Operations Research Letters, vol. 37, no. 5, pages 359–364, September 2009. (Cited on pages 21, 34 and 118.)

[Fourcade *et al.* 1997] Fabrice Fourcade, Ellis Johnson, Mourad Bara and Philippe Cortey-Dumont. *Optimizing nuclear power plant refueling with mixed-integer programming.* European journal of operational research, vol. 97, no. 2, pages 269–280, 1997. (Cited on page 97.)

[Frangioni *et al.* 2017] Antonio Frangioni, Bernard Gendron and Enrico Gorgone. *On the computational efficiency of subgradient methods: a case study with Lagrangian bounds.* Mathematical Programming Computation, vol. 9, no. 4, pages 573–604, December 2017. (Cited on page 24.)

[Gabrel *et al.* 2014] Virginie Gabrel, Cécile Murat and Aurélie Thiele. *Recent advances in robust optimization: An overview.* European journal of operational research, vol. 235, no. 3, pages 471–483, 2014. (Cited on pages 6 and 132.)

[Garey *et al.* 1976] M. R. Garey, D. S. Johnson and R. Sethi. *The complexity of flowshop and jobshop scheduling.* Mathematics of Operations Research, vol. 1, no. 2, pages 117–129, 1976. (Cited on page 42.)

[Gavranović & Buljubašić 2013] Haris Gavranović and Mirsad Buljubašić. *A Hybrid Approach Combining Local Search and Constraint Programming for a Large Scale Energy Management Problem.* RAIRO - Operations Research, vol. 47, no. 4, pages 481–500, 2013. (Cited on page 97.)

[Geoffrion 1971] A. M. Geoffrion. *Duality in Nonlinear Programming: A Simplified Applications-Oriented Development.* SIAM Review, vol. 13, no. 1, pages 1–37, 1971. Publisher: Society for Industrial and Applied Mathematics. (Cited on page 23.)

[Geoffrion 1974] A. M. Geoffrion. *Lagrangean relaxation for integer programming.* In M. L. Balinski, editor, Approaches to Integer Programming, Mathematical Programming Studies, pages 82–114. Springer, Berlin, Heidelberg, 1974. (Cited on pages 21 and 23.)

[Georghiou *et al.* 2015] Angelos Georghiou, Wolfram Wiesemann and Daniel Kuhn. *Generalized decision rule approximations for stochastic programming via liftings.* Mathematical Programming, vol. 152, no. 1-2, pages 301–338, 2015. (Cited on page 134.)

[Ghaoui *et al.* 2011] Laurent El Ghaoui, Vivian Viallon and Tarek Rabbani. *Safe Feature Elimination for the LASSO and Sparse Supervised Learning Problems.* arXiv:1009.4219 [cs, math], May 2011. arXiv: 1009.4219. (Cited on page 27.)

[Gharbi *et al.* 2013] Anis Gharbi, Talel Ladhari, Mohamed Kais Msakni and Mehdi Serairi. *The two-machine flowshop scheduling problem with sequence-independent setup times: New lower bounding strategies.* European Journal of Operational Research, vol. 231, no. 1, pages 69–78, November 2013. (Cited on pages 43, 44, 49 and 62.)

[Godskesen *et al.* 2013] Steffen Godskesen, Thomas Sejr Jensen, Niels Kjeldsen and Rune Larsen. *Solving a real-life, large-scale energy management problem.* Journal of Scheduling, vol. 16, no. 6, pages 567–583, 2013. (Cited on page 97.)

[Goh & Sim 2010] Joel Goh and Melvyn Sim. *Distributionally robust optimization and its tractable approximations.* Operations research, vol. 58, no. 4-part-1, pages 902–917, 2010. (Cited on pages 8, 132 and 133.)

[Gorissen *et al.* 2015] Bram L Gorissen, İhsan Yanıkoğlu and Dick den Hertog. *A practical guide to robust optimization.* Omega, vol. 53, pages 124–137, 2015. (Cited on page 134.)

[Griset *et al.* 2021] Rodolphe Griset, Pascale Bendotti, Boris Detienne, Marc Porcheron, Halil Şen and François Vanderbeck. *Combining Dantzig-Wolfe and Benders decompositions to solve a large-scale nuclear outage planning problem.* European Journal of Operational Research, July 2021. (Cited on pages 16, 18, 21, 34, 35 and 95.)

[Grötschel *et al.* 1981] M. Grötschel, L. Lovász and A. Schrijver. *The ellipsoid method and its consequences in combinatorial optimization.* Combinatorica, vol. 1, no. 2, pages 169–197, June 1981. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 2 Publisher: Springer-Verlag. (Cited on page 11.)

[Gschwind & Irnich 2017] Timo Gschwind and Stefan Irnich. *Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities.* OR Spectrum, vol. 39, no. 2, pages 541–556, 2017. (Cited on pages 65, 86 and 90.)

[Guignard & Kim 1987] Monique Guignard and Siwhan Kim. *Lagrangean decomposition: A model yielding stronger Lagrangean bounds.* Mathematical programming, vol. 39, no. 2, pages 215–228, 1987. (Cited on page 67.)

[Guillot & Stauffer 2020] Matthieu Guillot and Gautier Stauffer. *The Stochastic Shortest Path Problem: A polyhedral combinatorics perspective.* European Journal of Operational Research, vol. 285, no. 1, pages 148–158, August 2020. (Cited on page 14.)

[Hanasusanto *et al.* 2015] Grani A Hanasusanto, Daniel Kuhn and Wolfram Wiesemann. *K-adaptability in two-stage robust binary programming.* Operations Research, vol. 63, no. 4, pages 877–891, 2015. (Cited on pages 9, 38, 134, 155, 156, 158, 163 and 169.)

[Haouari & Kharbeche 2013] Mohamed Haouari and Mohamed Kharbeche. *An assignment-based lower bound for a class of two-machine flow shop problems.* Computers and Operations Research, vol. 40, no. 7, pages 1693 – 1699, 2013. (Cited on pages 60 and 61.)

[Held & Karp 1970] Michael Held and Richard M. Karp. *The Traveling-Salesman Problem and Minimum Spanning Trees.* Operations Research, vol. 18, no. 6, pages 1138–1162, 1970. Publisher: INFORMS. (Cited on page 21.)

[Hoogeveen *et al.* 2006] Han Hoogeveen, Linda van Norden and Steef van de Velde. *Lower bounds for minimizing total completion time in a two-machine flow shop.* Journal of Scheduling, vol. 9, no. 6, pages 559–568, 2006. (Cited on page 43.)

[Hooker 1994] J. N. Hooker. *Logic-based methods for optimization.* In Alan Borning, editor, Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, pages 336–349, Berlin, Heidelberg, 1994. Springer. (Cited on page 144.)

[Hooker 2013] John N. Hooker. *Decision Diagrams and Dynamic Programming.* In Carla Gomes and Meinolf Sellmann, editors, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science, pages 94–110, Berlin, Heidelberg, 2013. Springer. (Cited on page 13.)

[Ibaraki & Nakamura 1994] Toshihide Ibaraki and Yuichi Nakamura. *A Dynamic Programming Method for Single Machine Scheduling.* European Journal of Operational Research, vol. 76, no. 1, pages 72–82, 1994. (Cited on pages 27, 35, 50 and 74.)

[Ibaraki 1987] Toshihide Ibaraki. *Chapter 7 Successive sublimation methods for dynamic programming computation.* Annals of Operations Research, vol. 11, no. 1, pages 397–439, December 1987. (Cited on pages 13, 27, 35, 36, 74 and 75.)

[Ibaraki 1988] T Ibaraki. *Enumerative approaches to combinatorial optimization.* Annals of Operations Research, vol. 10-11, 1988. (Cited on page 35.)

[Ignall & Schrage 1965] Edward Ignall and Linus Schrage. *Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems.* Operations Research, vol. 13, no. 3, pages 400–412, 1965. (Cited on page 43.)

[Irnich *et al.* 2010] Stefan Irnich, Guy Desaulniers, Jacques Desrosiers and Ahmed Hadjar. *Path-Reduced Costs for Eliminating Arcs in Routing and Scheduling.* INFORMS Journal on Computing, vol. 22, no. 2, pages 297–313, May 2010. Publisher: INFORMS. (Cited on page 28.)

[Jiang *et al.* 2014] Ruiwei Jiang, Muhong Zhang, Guang Li and Yongpei Guan. *Two-stage network constrained robust unit commitment problem.* European Journal of Operational Research, vol. 234, no. 3, pages 751–762, 2014. (Cited on page 135.)

[Joncour 2011] Cédric Joncour. *2D-orthogonal packing and scheduling problems: modelling by graph theory and mathematical programming approaches.* PhD thesis, Université Sciences et Technologies - Bordeaux I, 2011. (Cited on pages 98 and 102.)

[Jost & Savourey 2013] Vincent Jost and David Savourey. *A 0–1 integer linear programming approach to schedule outages of nuclear power plants.* Journal of Scheduling, vol. 16, no. 6, pages 551–566, 2013. (Cited on page 97.)

[Kall & Wallace 1994] Peter Kall and Stein W. Wallace. Stochastic Programming. John Wiley & Sons, Chichester, second edition édition, 1994. (Cited on page 7.)

[Kämmerling & Kurtz 2020] Nicolas Kämmerling and Jannis Kurtz. *Oracle-based algorithms for binary two-stage robust optimization.* Computational Optimization and Applications, vol. 77, no. 2, pages 539–569, 2020. (Cited on pages 136 and 139.)

[Kao *et al.* 2008] Gio K. Kao, Edward C. Sewell and Sheldon H. Jacobson. *A branch, bound, and remember algorithm for the* $1|r_i|\sum T_i$ *scheduling problem.* Journal of Scheduling, vol. 12, no. 2, pages 163–175, August 2008. (Cited on page 56.)

[Karp & Held 1967] Richard M. Karp and Michael Held. *Finite-State Processes and Dynamic Programming.* SIAM Journal on Applied Mathematics, vol. 15, no. 3, pages 693–718, May 1967. Publisher: Society for Industrial and Applied Mathematics. (Cited on page 13.)

[Kuhn *et al.* 2011] Daniel Kuhn, Wolfram Wiesemann and Angelos Georghiou. *Primal and dual linear decision rules in stochastic and robust optimization.* Mathematical Programming, vol. 130, no. 1, pages 177–209, 2011. (Cited on page 133.)

[Land & Doig 1960] A. H. Land and A. G. Doig. *An Automatic Method of Solving Discrete Programming Problems.* Econometrica, vol. 28, no. 3, pages 497–520, 1960. Publisher: [Wiley, Econometric Society]. (Cited on page 32.)

[Land & Doig 2010] Ailsa H. Land and Alison G. Doig. *An Automatic Method for Solving Discrete Programming Problems.* In Michael Jünger, Thomas M.

Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi and Laurence A. Wolsey, editors, 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, pages 105–132. Springer, Berlin, Heidelberg, 2010. (Cited on page 32.)

[Lefebvre *et al.* Under revision] Henri Lefebvre, François Clautiaux and Boris Detienne. *A two-stage robust approach for the weighted number of tardy jobs with objective uncertainty.* Under revision. (Cited on pages 166 and 169.)

[Lemaréchal 2001] Claude Lemaréchal. *Lagrangian Relaxation.* In Michael Jünger and Denis Naddef, editors, Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions, Lecture Notes in Computer Science, pages 112–156. Springer, Berlin, Heidelberg, 2001. (Cited on pages 21 and 24.)

[Liebchen *et al.* 2009] Christian Liebchen, Marco Lübbecke, Rolf Möhring and Sebastian Stiller. *The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications.* In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Ravindra K. Ahuja, Rolf H. Möhring and Christos D. Zaroliagis, editors, Robust and Online Large-Scale Optimization, volume 5868, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. (Cited on page 9.)

[Lusby *et al.* 2010] Richard Martin Lusby, Laurent Flindt Muller and Bjørn Petersen. *A solution approach to the ROADEF/EURO 2010 challenge based on Benders' Decomposition.* Technical report, 2010. (Cited on page 97.)

[Lübbecke & Desrosiers 2005] Marco E. Lübbecke and Jacques Desrosiers. *Selected Topics in Column Generation.* Operations Research, vol. 53, no. 6, pages 1007–1023, December 2005. (Cited on page 16.)

[Martin & Shmoys 1996] Paul Martin and David B. Shmoys. *A new approach to computing optimal schedules for the job-shop scheduling problem.* In William H. Cunningham, S. Thomas McCormick and Maurice Queyranne, editors, Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, pages 389–403, Berlin, Heidelberg, 1996. Springer. (Cited on page 27.)

[Martin *et al.* 1990] R. Kipp Martin, Ronald L. Rardin and Brian A. Campbell. *Polyhedral Characterization of Discrete Dynamic Programming.* Operations Research, vol. 38, no. 1, pages 127–138, February 1990. Publisher: INFORMS. (Cited on pages 14 and 152.)

[Meyer 1974] R. R. Meyer. *On the existence of optimal solutions to integer and mixed-integer programming problems.* Mathematical Programming, vol. 7, no. 1, pages 223–235, December 1974. (Cited on page 11.)

[Mukerji *et al.* 1991] Rana Mukerji, Hyde M Merrill, Bruce W Erickson, JH Parker and RE Friedman. *Power plant maintenance scheduling: optimizing economics and reliability.* IEEE Transactions on Power Systems, vol. 6, no. 2, pages 476–483, 1991. (Cited on page 97.)

[Nemhauser & Ullmann 1968] George L. Nemhauser and Zev Ullmann. *A Note on the Generalized Lagrange Multiplier Solution to an Integer Programming Problem.* Operations Research, vol. 16, no. 2, pages 450–453, 1968. Publisher: INFORMS. (Cited on page 21.)

[Neumann 1928] J v Neumann. *Zur theorie der gesellschaftsspiele.* Mathematische annalen, vol. 100, no. 1, pages 295–320, 1928. (Cited on page 136.)

[Perchet & Vigeral 2015] Vianney Perchet and Guillaume Vigeral. *A Minmax Theorem for Concave-Convex Mappings with no Regularity Assumptions.* Journal of Convex Analysis, vol. 22, 01 2015. (Cited on page 24.)

[Peridy *et al.* 2003] Laurent Peridy, Eric Pinson and David Rivreau. *Using short-term memory to minimize the weighted number of late jobs on a single machine.* European Journal of Operational Research, vol. 148, no. 0, pages 591–603, 2003. (Cited on pages 50 and 53.)

[Pessoa *et al.* 2018a] A. Pessoa, R. Sadykov, E. Uchoa and F. Vanderbeck. *Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation.* INFORMS Journal on Computing, vol. 30, no. 2, pages 339–360, May 2018. Publisher: INFORMS. (Cited on page 18.)

[Pessoa *et al.* 2018b] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa and François Vanderbeck. *Automation and combination of linear-programming based stabilization techniques in column generation.* INFORMS Journal on Computing, vol. 30, no. 2, pages 339–360, 2018. (Cited on pages 120 and 156.)

[Pessoa *et al.* 2020] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa and François Vanderbeck. *A generic exact solver for vehicle routing and related problems.* Mathematical Programming, vol. 183, no. 1, pages 483–523, September 2020. (Cited on page 18.)

[Pflug 2000] Georg Ch Pflug. *Some remarks on the value-at-risk and the conditional value-at-risk.* In Probabilistic constrained optimization, pages 272–281. Springer, 2000. (Cited on page 6.)

[Pinedo 2012] Michael L. Pinedo. *Advanced Single Machine Models (Deterministic).* In Michael L. Pinedo, editor, Scheduling: Theory, Algorithms, and Systems, pages 69–109. Springer US, Boston, MA, 2012. (Cited on page 13.)

[Pisinger 2005] David Pisinger. *Where are the hard knapsack problems?* Computers & Operations Research, vol. 32, no. 9, pages 2271–2284, 2005. (Cited on pages 158 and 161.)

[Pochet & Wolsey 2006] Yves Pochet and Laurence A Wolsey. Production planning by mixed integer programming. Springer Science & Business Media, 2006. (Cited on page 13.)

[Polyak 1969] B. T. Polyak. *Minimization of unsmooth functionals*. USSR Computational Mathematics and Mathematical Physics, vol. 9, no. 3, pages 14–29, January 1969. (Cited on page 25.)

[Porcheron *et al.* 2010] Marc Porcheron, Agnès Gorge, Olivier Juan, Thomas Simovic and Guillaume Dereu. *Challenge ROADEF/EURO 2010: A large-scale energy management problem with varied constraints*, 2010. (Cited on page 97.)

[Postek & Hertog 2016] Krzysztof Postek and Dick den Hertog. *Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set*. INFORMS Journal on Computing, vol. 28, no. 3, pages 553–574, 2016. (Cited on page 135.)

[Rahmaniani *et al.* 2017] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau and Walter Rei. *The Benders decomposition algorithm: A literature review*. European Journal of Operational Research, vol. 259, no. 3, pages 801–817, June 2017. (Cited on page 21.)

[Righini & Salani 2008] Giovanni Righini and Matteo Salani. *New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem*. Networks, vol. 51, no. 3, pages 155–170, 2008. (Cited on page 35.)

[Rockafellar & Uryasev 2000] R Tyrrell Rockafellar and Stanislav Uryasev. *Optimization of conditional value-at-risk*. Journal of risk, vol. 2, pages 21–42, 2000. (Cited on page 6.)

[Rozenknop *et al.* 2013] Antoine Rozenknop, Roberto Wolfler Calvo, Laurent Alfandari, Daniel Chemla and Lucas Létocart. *Solving the electricity production planning problem by a column generation based heuristic*. journal of Scheduling, vol. 16, no. 6, pages 585–604, 2013. (Cited on pages 97 and 102.)

[Sadykov & Vanderbeck 2013] R. Sadykov and F. Vanderbeck. *Bin Packing with Conflicts: a generic Branch-and-Price algorithm*. INFORMS Journal on Computing, vol. 25, no. 2, pages 244–255, 2013. (Cited on page 84.)

[Sadykov *et al.* 2019] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri and Eduardo Uchoa. *Primal Heuristics for Branch and Price: The Assets of Diving Methods*. INFORMS Journal on Computing, vol. 31, no. 2, pages 251–267, 2019. (Cited on pages 18, 117, 118 and 156.)

[Schrijver 1986] Alexander Schrijver. Theory of linear and integer programming. John Wiley & Sons, Inc., USA, 1986. (Cited on page 2.)

[Sellmann 2004] Meinolf Sellmann. *Theoretical Foundations of CP-Based Lagrangian Relaxation.* In CP, pages 634–647, 2004. (Cited on pages 27 and 28.)

[Shapiro *et al.* 2014] Alexander Shapiro, Darinka Dentcheva and Andrzej Ruszczyński. Lectures on stochastic programming: modeling and theory. SIAM, 2014. (Cited on pages 4, 5, 7, 8 and 19.)

[Shapiro 2021] Alexander Shapiro. *Tutorial on risk neutral, distributionally robust and risk averse multistage stochastic programming.* European Journal of Operational Research, vol. 288, no. 1, pages 1–13, January 2021. (Cited on page 4.)

[Sherali & Lim 2007] Hanif D. Sherali and Churlzu Lim. *Enhancing Lagrangian Dual Optimization for Linear Programs by Obviating Nondifferentiability.* INFORMS J. on Computing, vol. 19, no. 1, pages 3–13, January 2007. (Cited on page 50.)

[Sherali & Ulular 1989] Hanif D. Sherali and Osman Ulular. *A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems.* Applied Mathematics and Optimization, vol. 20, no. 1, pages 193–221, July 1989. (Cited on page 50.)

[Slyke & Wets 1969] R. M. Van Slyke and Roger Wets. *L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming.* SIAM Journal on Applied Mathematics, vol. 17, no. 4, pages 638–663, 1969. (Cited on page 112.)

[Sourd 2009] Francis Sourd. *New Exact Algorithms for One-Machine Earliness-Tardiness Scheduling.* INFORMS Journal on Computing, vol. 21, no. 1, pages 167–175, February 2009. Publisher: INFORMS. (Cited on page 27.)

[Soyster 1973] A. L. Soyster. *Technical Note—Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming.* Operations Research, vol. 21, no. 5, pages 1154–1157, October 1973. Publisher: INFORMS. (Cited on page 6.)

[Subramanyam *et al.* 2020] Anirudh Subramanyam, Chrysanthos E. Gounaris and Wolfram Wiesemann. *K-adaptability in two-stage mixed-integer robust optimization.* Mathematical Programming Computation, vol. 12, no. 2, pages 193–224, June 2020. (Cited on pages 38, 134, 155, 156 and 158.)

[Tanaka & Araki 2013] Shunji Tanaka and Mituhiko Araki. *An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times.* Computers & Operations Research, vol. 40, no. 1, pages 344–352, January 2013. (Cited on page 64.)

[Tanaka & Fujikuma 2012] Shunji Tanaka and Shuji Fujikuma. *A dynamic-programming-based exact algorithm for general single-machine scheduling*

with machine idle time. Journal of Scheduling, vol. 15, no. 3, pages 347–361, June 2012. (Cited on pages 36, 64, 84 and 171.)

[Tanaka *et al.* 2009] Shunji Tanaka, Shuji Fujikuma and Mituhiko Araki. *An exact algorithm for single-machine scheduling without machine idle time.* Journal of Scheduling, vol. 12, no. 6, pages 575–593, December 2009. (Cited on pages 36, 50, 58 and 64.)

[Tanaka *et al.* 2015] Shunji Tanaka, Ruslan Sadykov and Boris Detienne. *A new Lagrangian bound for the min-sum job-shop scheduling.* July 2015. (Cited on pages 16 and 28.)

[Tanaka 2011] Shunji Tanaka. *Extension of the Dynasearch to the Two-Machine Permutation Flowshop Scheduling Problem (in Japanese).* Transactions of the Institute of Systems, Control and Information Engineers, vol. 24, no. 2, pages 23–30, 2011. (Cited on pages 48, 49, 51 and 56.)

[Thiele *et al.* 2009] Aurélie Thiele, Tara Terry and Marina Epelman. *Robust linear optimization with recourse.* Rapport technique, pages 4–37, 2009. (Cited on pages 9 and 135.)

[T'kindt *et al.* 2004] V. T'kindt, F. Della Croce and C. Esswein. *Revisiting Branch and Bound Search Strategies for Machine Scheduling Problems.* Journal of Scheduling, vol. 7, no. 6, pages 429–440, November 2004. (Cited on page 56.)

[van Ackooij *et al.* 2018] Wim van Ackooij, Jérôme De Boeck, Boris Detienne, Stefania Pan and Michael Poss. *Optimizing power generation in the presence of micro-grids.* European Journal of Operational Research, vol. 271, no. 2, pages 450–461, December 2018. (Cited on page 167.)

[Van Slyke & Wets 1969] R. M. Van Slyke and Roger Wets. *L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming.* SIAM Journal on Applied Mathematics, vol. 17, no. 4, pages 638–663, July 1969. Publisher: Society for Industrial and Applied Mathematics. (Cited on page 19.)

[Vanderbeck & Wolsey 2010] François Vanderbeck and Laurence A. Wolsey. *Reformulation and Decomposition of Integer Programs.* In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi and Laurence A. Wolsey, editors, 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, pages 431–502. Springer, Berlin, Heidelberg, 2010. (Cited on pages 13 and 24.)

[Vanderbeck 2000] François Vanderbeck. *On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm.* Operations Research, vol. 48, no. 1, pages 111–128, 2000. (Cited on pages 16 and 34.)

[Vanderbeck 2011] François Vanderbeck. *Branching in branch-and-price: a generic scheme.* Mathematical Programming, vol. 130, no. 2, pages 249–294, 2011. (Cited on page 120.)

[Vanderbei 2020] Robert J Vanderbei. Linear programming: foundations and extensions, volume 285. Springer Nature, 2020. (Cited on page 10.)

[Vayanos *et al.* 2011] Phebe Vayanos, Daniel Kuhn and Berç Rustem. *Decision rules for information discovery in multi-stage stochastic programming.* In Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on, pages 7368–7373. IEEE, 2011. (Cited on pages 38 and 134.)

[Velde 1990] Steef van de Velde. *Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation.* Annals of Operations Research, pages 257–268, 1990. (Cited on pages 43 and 45.)

[Wolsey & Nemhauser 1999] Laurence A. Wolsey and George L. Nemhauser. Integer and Combinatorial Optimization. John Wiley & Sons, July 1999. (Cited on pages 2, 16, 32 and 141.)

[Wolsey 2020] Laurence A Wolsey. Integer programming. John Wiley & Sons, 2020. (Cited on pages 2 and 10.)

[Zhao & Zeng 2012a] Long Zhao and Bo Zeng. *An exact algorithm for two-stage robust optimization with mixed integer recourse problems.* submitted, available on Optimization-Online.org, 2012. (Cited on pages 38 and 135.)

[Zhao & Zeng 2012b] Long Zhao and Bo Zeng. *Robust unit commitment problem with demand response and wind energy.* In Power and Energy Society General Meeting, 2012 IEEE, pages 1–8. IEEE, 2012. (Cited on page 135.)

[Zhao *et al.* 2013] Chaoyue Zhao, Jianhui Wang, Jean-Paul Watson and Yongpei Guan. *Multi-stage robust unit commitment considering wind and demand response uncertainties.* IEEE Transactions on Power Systems, vol. 28, no. 3, pages 2708–2717, 2013. (Cited on pages 135 and 147.)

[Zhen *et al.* 2018] Jianzhe Zhen, Dick Den Hertog and Melvyn Sim. *Adjustable robust optimization via Fourier–Motzkin elimination.* Operations Research, vol. 66, no. 4, pages 1086–1100, 2018. (Cited on page 135.)