Short-term hydropower production scheduling : feasibility and modeling

Youcef Sahraoui

NNT : 2016SACLX025

THÈSE DE DOCTORAT
DE
L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À
L'ÉCOLE POLYTECHNIQUE

ÉCOLE DOCTORALE N°580
Sciences et technologies de l'information et de la communication

Spécialité de doctorat : Mathématiques et Informatique

Par

Youcef SAHRAOUI

Planification de la production hydroélectrique au court terme : faisabilité et modélisation

Thèse présentée et soutenue à Palaiseau, le 9 juin 2016

Composition du Jury :

Michalis VAZIRGIANNIS	Professeur de l'École Polytechnique	Président du Jury
Michele MONACI	Professeur, Università degli Studi di Bologna	Rapporteur
Roberto WOLFLER CALVO	Professeur, Université Paris 13	Rapporteur
Pascale BENDOTTI	Ingénieur, EDF R&D	Examinateur
Delphine SINOQUET	Ingénieur, IFP Énergies Nouvelles	Examinateur
Claudia D'AMBROSIO	Chargé de Recherche, CNRS	Co-directeur de thèse
Leo LIBERTI	Directeur de Recherche, CNRS	Directeur de thèse

Contents

Acknowledgements

This thesis is the result of common efforts that I want to acknowledge.

I want to thank thesis reviewers Michele Monaci and Roberto Wolfler Calvo as well as defense examiners Delphine Sinoquet and Michalis Vazirgiannis for the attention they gave to my work and for their precious feedback to improve it.

I am grateful to my supervisors Claudia D'Ambrosio and Leo Liberti for introducing me to academic research in mathematical optimization and for continually transmitting their drive to obtain results.

This thesis was conducted under the *Industrial Agreement for Training through Research* (CIFRE) with EDF R&D. I am thankful to my EDF advisors Pascale Bendotti, Grace Doukopoulos, Arnaud Lenoir and Tomas Simovic – not to mention Ala Ben Abbes – for teaching me a great deal about electricity production management and for their patient mentoring.

I am indebted to all the fellow workers I have had: at Ecole Polytechnique's LIX lab, especially in the Sysmo team, and at EDF's OSIRIS department, especially in the "Optimization" team R36. All the small things we shared – from optimization insights to pataphysical small talk, including administrative tips and even and code debugging – add up to something meaningful: their company made my days. I feel lucky I had beside me Gustavo, Kostas and Panagiotis, as well as Jean-Christophe, Denis, Hugo and David.

Although they stayed backstage, friends and relatives played a major role in the completion of this work. I thank the main instigators Abd-El-Kader and Nacéra, all the Ould Kaddour's and Sahraoui's around the world, as well as Anne-Laure, Arthur, Aya, Camille, Cléo, Emmanuelle, Fanny, Faiza, Georgios, Guillaume S, Guillaume V, Hubert, Jérôme, Léa, Maria, Mathieu, Reda, Thibaut, Timothée, Yani, Younès and Zineb – to name a few. I also have a special thought for my late uncle Ahmed.

Synthèse

Planification de la production hydroélectrique au court terme : faisabilité et modélisation

Dans le secteur électrique, et en particulier chez le fournisseur et producteur EDF, l'optimisation mathématique est utilisée pour modéliser et résoudre des problèmes liés à la gestion de la production d'électricité. Citons quelques exemples d'applications : la planification des investissements en infrastructures, la modélisation des problèmes d'équilibre des marchés de l'électricité, la gestion des risques d'épuisement des barrages, la programmation des arrêts de tranches nucléaires. Plus particulièrement, l'hydroélectricté est une source d'énergie renouvelable, peu chère, flexible mais limitée. Exploiter l'hydroélectricité constitue donc un enjeu important pour la gestion de la production électrique.

Nous nous intéressons à des problèmes d'optimisation de Programmation Non Linéaire en Nombres Entiers (PNLNE). Ce sont des problèmes d'optimisation dont les variables de décision peuvent être continues ou discrètes et dont les fonctions exprimant l'objectif et les contraintes peuvent être linéaires ou non-linéaires. Les non-linéarités et la combinatoire induite par les variables entières rendent ces problèmes particulièrement difficiles à résoudre. En effet, les méthodes existantes n'arrivent pas toujours à résoudre les grands problèmes de PNLNE à l'optimalité avec des temps de calcul limités. En amont des performances de résolution, la faisabilité est une question préliminaire à aborder puisqu'il faut s'assurer que les PNLNE à résoudre admettent des solutions. En fait, la faisabilité peut s'avérer aussi difficile à prouver que l'optimalité. De plus, lorsque l'on rencontre des infaisabilités dans des modèles complexes, il est très utile mais très difficile d'analyser leurs origines. Par ailleurs, la résolution de PNLNE est rendue encore plus difficile si l'on requiert de certifier une précision exacte des résultats. En effet, les méthodes résolutions sont en général mises en oeuvre en arithmétique flottante, ce qui peut donner lieu à une précision approchée.

Dans cette thèse, nous abordons deux problèmes d'optimisation, un Programme Linéaire en Nombres Entiers (PLNE) et un Programme Non Linéaire (PNL), liés à la planification de la production hydroélectrique au court terme, *Hydro Unit-Commitment* (HUC) en Anglais. Etant

données des ressources d'eau finies dans les barrages, l'objet du HUC est de prescrire des programmes de production les plus rentables qui soient compatibles avec les spécifications techniques des usines hydroélectriques. Les volumes d'eau, les débits d'eau, et les puissances électriques sont representées par des variables continues tandis que l'activation des turbines est communément formulée avec des variables binaires. Les non-linéarités proviennent en général des fonctions qui modélisent la puissance générée en fonction du volume et du débit d'eau. Nous distinguons donc deux problèmes : un PLNE figurant des caractéristiques linéaires et discrètes, et un PNL figurant des caractéristiques non-linéaires et continues.

Dans le deuxième chapitre, nous traitons de la faisabilité d'un problème réel de planification de la production hydroélectrique formulé en PLNE. Comparé à un problème standard de planification, le modèle inclut deux spécifications supplémentaires : des points de fonctionnements discrets sur la courbe puissance-débit ainsi que des cibles à mi-parcours et de fin de parcours pour les niveaux d'eau des réservoirs. Les complications liées aux données réelles et au calcul numérique, associées aux caractéristiques spécifiques du modèle, rendent le problème plus difficile à résoudre et souvent infaisable. Etant données des instances réelles, nous reformulons le modèle pour rendre le problème faisable. Nous procédons par étapes pour identifier et traiter les sources d'infaisabilité une par une, à savoir les erreurs numériques et les infaisabilités de modélisation. Des résultats informatiques étayent l'efficacité de notre méthode sur un ensemble de test de 66 instances réelles qui contenait initialement de nombreuses infaisabilités.

Le troisième chapitre porte sur l'adaptation de l'algorithme du *Multiplicative Weights Update* (MWU) pour la PNLNE. Cette adaptation est fondée sur une reformulation paramétrée spécifique à chaque problème que l'on dénomme *pointwise*. Nous définissons des propriétés souhaitables pour obtenir de bonnes reformulations pointwise et nous fournissons des règles de haut niveau pour adapter l'algorithme étape par étape. Contrairement à la plupart des heuristiques, nous démontrons que la méthode MWU conserve une garantie d'approximation relative pour les PNLNE. Nous comparons notre méthode à la méthode *Multi-Start* (MS) pour résoudre un problème planification de la production hydroélectrique formulé en PNL. Les résultats informatiques penchent en faveur de la méthode MWU.

Mots-clefs: programmation mixte en nombres entiers, programmation non-linéaire, planification de la production hydroélectrique, infaisabilité, calculs numériques exacts, matheuristiques.

Summary

Short-term hydropower production scheduling: feasibility and modeling

In the electricity industry, and more specifically at the French utility company EDF, mathematical optimization is used to model and solve problems related to electricity production management. To name a few applications: planning for capacity investments, modeling equilibrium problems for electricity markets, managing depletion risk of hydro-reservoirs, scheduling outages and refueling for nuclear plants. More specifically, hydroelectricity is a renewable, cheap, flexible but limited source of energy. Harnessing hydroelectricity is thus critical for electricity production management.

We are interested in Mixed-Integer Non-Linear Programming (MINLP) optimization problems. They are optimization problems whose decision variables can be continuous or discrete and the functions to express their objective and constraints can be linear or non-linear. The non-linearities and the combinatorial aspect induced by the integer variables make these problems particularly difficult to solve. Indeed existing methods cannot always solve large MINLP problems to the optimum within limited computational timeframes. Prior to solution performance, feasibility is a preliminary challenge to tackle since we should ensure the MINLP problems to solve admit feasible solutions. Actually, feasibility can be as difficult to prove as optimality. In addition, when infeasibilities occur in complex models, it is useful but not trivial to analyze their causes. Also, certifying the exactness of the results compounds the difficulty of solving MINLP problems as solution methods are generally implemented in floating-point arithmetic, which may lead to approximate precision.

In this thesis, we work on two optimization problems – a Mixed-Integer Linear Program (MILP) and a Non-Linear Program (NLP) – related to Short-Term Hydropower production Scheduling (STHS). STHS is also commonly known as Hydro Unit-Commitment (HUC). Given finite resources of water in reservoirs, the purpose of HUC is to prescribe production schedules with largest payoffs that are compatible with technical specifications of the hydroelectric plants. While water volumes, water flows, and electric powers can be represented with continuous vari-

ables, commitment statuses of turbine units usually have to be formulated with binary variables. Non-linearities commonly originate from the Input/Output functions that model generated power according to water volume and water flow. We decide to focus on two distinguished problems: a MILP with linear discrete features, and an NLP with non-linear continuous features.

In the second chapter, we deal with feasibility issues of a real-world hydropower unit commitment MILP problem. Compared with a standard HUC problem, the model features two additional specifications: discrete operational points of the power-flow curve and mid-horizon and final strict targets for reservoir levels. Issues affecting real-world data and numerical computing, together with specific model features, make the problem harder to solve and often infeasible. Given real-world instances, we reformulate the model to make the problem feasible. We follow a step-by-step approach to systematically exhibit and cope with one source of infeasibility at a time, namely numerical errors and model infeasibilities. Computational results show the effectiveness of the approach on an original test set of 66 real-world instances that demonstrated a high occurrence of infeasibilities.

The third chapter is about the adaptation of the Multiplicative Weights Update (MWU) algorithm to solve MINLP. Such adaptation is based on a parametrized reformulation denoted *pointwise* that is specific to each problem. We define desirable properties for deriving pointwise reformulations and provide elaborate high-level guidelines to adapt the algorithm step-by-step. Unlike most heuristics, we show that the MWU method still retains a relative approximation guarantee in the NLP and MINLP settings. To deliver a proof of its applicability, we implement the method to solve a hard NLP HUC problem and benchmark it computationally against the Multi-Start (MS) method. We find the MWU compares favorably to the MS, which offers no approximation guarantee.

Keywords: mixed-integer programming, non-linear programming, hydro unit commitment, short-term hydropower production scheduling, infeasibility, exact computation, matheuristics.

Chapter 1

Introduction

Mathematical optimization is a subfield of Operations Research or Management Science and is sometimes synonymous to Mathematical Programming. The Operations Research discipline aims at using scientific tools to solve real-world problems such as programming and managing operations [39]. In the electricity industry, and more specifically at the French utility company EDF, mathematical optimization is used to model and solve problems related to electricity production management. To name a few applications: modeling equilibrium problems for electricity markets [31], managing depletion risk of hydro-reservoir depletion [4], scheduling outages and refueling for nuclear plants [60], planning mid-term production [47]. More specifically, hydroelectricity is a renewable, cheap, flexible but limited source of energy. Harnessing hydroelectricity – by resorting to mathematical optimization, when relevant – is thus critical for electricity production management.

We are interested in Mixed-Integer Non-Linear Programming (MINLP) optimization problems. They are optimization problems whose decision variables can be continuous or discrete and the functions to express their objective and constraints can be linear or non-linear. The non-linearities and the combinatorial aspect induced by the integer variables make these problems particularly difficult to solve [14, 23]. Indeed existing methods cannot always solve large MINLP problems to the optimum within limited computational timeframes. Prior to solution performance, feasibility is preliminary challenge to tackle since we should ensure the MINLP problems to solve admit feasible solutions [18]. Actually, feasibility can be has difficult to prove as optimality. In addition, when infeasibilities occur in complex models, it is useful but not trivial to analyze their causes. Also, certifying the exactness of the results compounds the difficulty of solving MINLP problems as solution methods are generally implemented in floating-point arithmetic, which may lead to approximate precision [71].

In this thesis, we work on two optimization problems – a Mixed-Integer Linear Program

(MILP) and a Non-Linear Program (NLP) – related to Short-Term Hydroelectricity production Scheduling (STHS) [17]. STHS is also commonly known as Hydro Unit-Commitment (HUC). Given finite resources of water in reservoirs, the purpose of HUC is to prescribe production schedules with largest payoffs that are compatible with technical specifications of the hydroelectric plants. While water volumes, water flows, and electric powers can be represented with continuous variables, commitment statuses of turbine units usually have to be formulated with binary variables. Non-linearities commonly originate from the Input/Output (I/O) functions that model generated power according to water volume and water flow. We decide to focus on two distinguished problems: a MILP with linear discrete features, and an NLP with non-linear continuous features.

In the remainder of Chapter 1, in Section 1.1, we state the two problems we later tackle (in Chapters 2 and 3) and summarize our contributions.

In Section 1.2, we introduce notions related to hydropower, processes, and structures that allow to harness hydropower to produce electricity, along with specificities of hydroelectricity compared to other technologies, especially when it comes to managing production.

In Section 1.3, we give an outline electricity production management in order to introduce one of its problems of interest: short-term hydroelectricity production scheduling.

In Section 1.4 we discuss of the use of optimization to tackle hydroelectricity management.

1.1 Problem statement and contributions

1.1.1 Case study 1: a real-world hydropower unit commitment problem

In Chapter 2, we deal with a real-world hydropower unit commitment MILP problem that arises in EDF's day-ahead scheduling process. More specifically, we focus on feasibility issues. First, the consistency of the real-world data we use needs to be checked. Secondly, the solution is obtained through numerical computations, that could alter the feasibility of the original problem. Last, some requirements are translated into constraints that could lead to infeasibility. Compared with a standard HUC problem, the model features two additional specifications: discrete operational points of the power-flow curve and mid-horizon and final strict targets for reservoir levels. Our aim is to find a feasible solution that deviates as little as possible from the given requirements.

For the set of real-world instances we work with, we provide a detailed analysis that distinguishes and explains interactions between:

- infeasibilities due to rounding errors;
- infeasibilities due to data errors, that is to say noisy values;

- infeasibilities due to data inconsistencies, that is to say irrelevant values;
- infeasibilities due to model inconsistencies.

Rounding errors are dealt with the scaling of volume-related variables and constraints, which we assume to be the most influential. Proper scaling is derived according to the *feasibility tolerance* of the floating-point-based solver.

Given an upper bound on the noise that affects data, data errors are dealt with a corrective relaxation of all constraints.

We only deal with the most frequent model inconsistency: target volume bounds. To do so, we propose a preliminary stage to recover feasibility while deviating as little as possible from the original target volume bounds.

The systematic approach we have followed improves the use of a generic MILP solver to effectively solve a real-world hydropower scheduling problem. It could be partly generalized into a methodology for practical optimization of real-world problems whose feasibility is questioned.

1.1.2 Case study 2: a heuristic for MINLP

In Chapter 3, we devise a new solution method for MINLP based on the adaptation of the Multiplicative Weights Update (MWU) algorithm and the derivation of a specific parametrized reformulation denoted *pointwise*.

We define desirable properties for deriving pointwise reformulations and provide elaborate high-level guidelines to adapt each step of the MWU. The method can be considered as a matheuristic – a heuristic based on Mathematical Programming – for global optimization of MINLP where the global phase is random-based and the local phase is performed by a generic solver. Unlike most heuristics, we show that the MWU method still retains a relative approximation guarantee in the NLP and MINLP settings.

To deliver a proof of its applicability, we implement the method to obtain sufficiently good solutions for a hard NLP HUC problem. Compared to another classical heuristic – the Multi-Start (MS) method – experimental results show good computational behavior under several aspects, while the MS method offers no approximation guarantee.

1.2 Hydropower and hydroelectricity

1.2.1 Basics of hydropower

Hydropower denotes any form of power related to water. Though they are commonly used for one another – it will be the case in this thesis – energy and power are two different physically

concepts: power is the instantaneous consumption or production of energy. Energy follows the conservation principle: it cannot be created nor lost but it is converted from one form to other forms or it is transferred from an object to other objects. The conservation principle relies on the abstraction of potential energy: it reflects the amount of tangible energy that can be released once forces have effect on the objects. Energy exists under several forms; to name a few: thermal, kinetic, electric, electromagnetic, chemical, nuclear.

In this thesis, we will focus on systems that transform kinetic energy of water *naturally* in motion into electricity: hydroelectricity. Other systems like waterwheels transform kinetic energy of water into another kind of kinetic energy. Steam engines within thermal plants also work with water in motion that has been previously heated; this process does not fall into the category of hydroelectricity.

The overall perpetual motion of water on earth is denoted water cycle. Its sketch is the following: water from clouds precipitates into rain or snow; rainfalls and snowmelt run off in streams or rivers; streams within drainage basins converge to lakes or larger water courses; water then flows into oceans; liquid water from all water bodies (oceans, lakes, etc.) evaporates into clouds. There is also motion of water in oceans: winds create waves, periodic gravitational forces of the moon cause tides, differences in temperature/pressure/salinity of water produces ebbs. Marine hydroelectricity relies on the use of such phenomena but we will not delve into the details of this technology.

In this thesis, we will focus on dam hydroelectricity. Dam – or storage or reservoir – hydroelectricity relies on the possibility of retaining water in a reservoir thanks to a dam in order to release it to convert the kinetic energy of its discharge. Pump-storage hydroelectricity is a specific enhanced dam hydroelectric structure where the conversion is reversible: on top of the primary capacity of discharging water to produce electricity, we can use electricity to pump water from downstream to upstream; it is a transformation of electrical energy into gravitational potential energy. In addition to dam and pump-storage, another kind of hydroelectricity is denoted run-of-the river. It relies on the conversion the kinetic energy of water running in a river into electric energy without control of the flow. Run-of-the-river hydroelectricity is somewhat a degenerate dam structure. A notable advantage of dam hydroelectricity is the capacity to store water and to decide when to use it. Pump-storage structures offer the capacity to indirectly store electricity.

To give an insight of the electrical power that can be generated, let us introduce a basic formula for the conventional power characteristic – or I/O function – of a single dam hydroelectric structure by considering the energy balance with elementary linear variations under simplifying assumptions.

Let ΔV $[\mathrm{m}^3]$ be the volume of an elementary water amount discharge downstream from

the upstream reservoir; it is initially at a height h [m] from the turbine; its mass is $\rho\Delta V$ [kg], where density of water is ρ [kg.m^{-3}].

The weigh upon it is vertical, directed downwards and its norm is $\rho g\Delta V$.

The work of the weight from the upstream reservoir to turbine (or the loss in gravitational potential energy) is $\rho gh\Delta V$.

Not all the potential energy is transferred into electric energy, the power collected is dampened by an efficiency factor η to account for those losses, hence the hydroelectric energy produced is $\eta\rho gh\Delta V$.

If it takes ΔT [s] for that volume to go downstream, the average power transfer P is $\eta\rho gh\frac{\Delta V}{\Delta T}$ or

$$P = \eta\rho ghQ$$

where $Q = \frac{\Delta V}{\Delta T}$ can be seen as the average flow.

The power characteristic $P(Q, h)$ depends on the water flow Q and the relative water level h, which depends on the volume in the reservoir. Actually, the expression of the power characteristic is more complex; the interested reader can refer to [5] for a case of the Itaipú plant where variations in tailrace elevation, penstock head losses and turbine-generator efficiencies are considered in the power characteristic.

Commonly, several structures are built in a drainage basin to benefit from the several incoming streams and intermediate lakes; they are denoted multi-reservoir systems or *valleys*. The most elementary form is a series of cascaded reservoirs. More generically, multi-reservoir systems have a dendritic form. Within a multi-reservoir system, the several structures are connected by water courses or pipes; for each structure, incoming flows include the water discharge from the upstream structures on top of precipitations and natural streams, outgoing flows feed the reservoirs of downstream structures. A description of the generic equipment of hydroelectric structures is given in the Appendix A.1.

1.2.2 Electricity technology comparison

When we compare energy forms, it is actually conversions that we compare: production of the energies of interest from a given primary energy – *e.g.*, solar thermal energy versus solar electric energy –, consumption of the energies of interest for a given use, – *e.g.*, electric cars versus fuel-based cars –, storage of an energy into some forms of potential energy of interest – *e.g.*, electro-chemical batteries versus power-to-gas.

In order to highlight the specificities of hydroelectricity, let us compare it to a few other technologies that produce electricity: nuclear power plants, conventional thermal power plants (coal, gas or biomass fired), solar (photovoltaic) power plants, wind power plants. This comparison is not universal and must be qualified according to existing devices, practices and infrastructures that are context-dependent. Our comparison follows the enumeration of advantages and disadvantages presented in [77].

Hydroelectricity is a dispatchable, flexible, scarce, reversible energy. Electricity production is said to be dispatchable when it can be controlled (increased or decreased, switched on or off). Water, nuclear fuel, coal, gas, biomass are primary energy sources that can be stored. Control of water release for dam hydroelectricity (as described in Section 1.2) and of water steam for thermal plants (through the heat of nuclear reaction or fuel combustion) allows control of turbine spinning and electricity generation. Therefore, the productions of hydroelectric, nuclear, conventional thermal power plants are dispatchable. Conversely, windmills, solar plants and isolated run-of-the-river plants are not deemed controllable since their productions only depend on natural inputs over which there is no control. Such technologies are not totally uncontrollable as we can sometimes reduce their production outputs. Being able to dispatch electricity production is essential in order to match a varying demand, as we will see in Section 1.3. Also, operations of hydroelectric plants are quite flexible: they can be launched or shut down very quickly as technical specifications allow relatively fast variations. This property is not fully shared with thermal plants because of technical limitations on their operating domains for example. This property does not apply to solar and wind power plants as their operations is not even dispatchable. Flexibility can be regarded as a short-term dispatchability. However, water is a scarce resource. While thermal plants can be replenished on demand since fuel procurement is – theoretically – not limited, water levels in reservoirs depend on the inflows coming from seasonal precipitations. Scarcity is somewhat similar to long-run dispatchability. Additionally, pump-storage hydroelectricity allows to somewhat *reverse* electricity production by indirectly storing it to dispatch it later.

Hydroelectricity is a renewable, relatively clean and relatively cheap energy. Though there may be limited precipitations or droughts locally on the short run, there is no limited stock of flowing water on the long run as water keeps on cycling. Provided water cycles remain, hydroelectricity is renewable. In the future, climate change will however affect the regularity of water cycles – amounts and seasonality of precipitations and evaporations – as well as water needs – in case of floods, droughts or heatwaves. There is no greenhouse gas emission, waste nor residual pertaining to hydroelectricity. For this reason, water is sometimes denoted *white coal*. Water is a free fuel. Operations and maintenance costs of hydroelectric plants are somewhat low, while construction is much more capital-intensive [1]. The life of hydroelectric plants

spans from 50 up to more 100 years, which is comparable if not longer than the lifespan other kinds of power plants.

A lot is at stake when deciding to build a hydroelectric plant and when deciding how to operate it. First of all, the construction of a hydroelectric plant necessitates the flooding of a drainage basin and causes considerable changes of the ecosystem. Indeed hydroelectricity is locally bound to drainage basins and water courses. The location constraints are somewhat less restrictive for other technologies. For example, thermal plants require water course proximity for cooling as well as transport accessibility for fuel replenishment and waste disposal. Secondly, building hydroelectric plants may require substantial changes in existing human activities: population moves, preclusion of waterway navigation. Storing water and controlling its discharge allows uses that are sometimes concurrent with generation of electricity: water supply, flood control, irrigation, recreational activities, industrial uses, etc. The case of the Aswan dam across the Nile in Egypt and the renown ensuing relocation of the Abu Simbel archaeological sites (see `https://en.wikipedia.org/wiki/Aswan_Dam`) illustrate how intricate the stakes can be. The Three Gorges Dam over the Yangtze river in China is another example where effects on the environment and population relocation were significant. (see `https://en.wikipedia.org/wiki/Three_Gorges_Dam`) Thirdly, reliability of the structure is tantamount to ensure the safety of people living downstream; dam breaches are unfortunately not uncommon catastrophes, as we have witnessed in Brazil in November 2015 (see `https://en.wikipedia.org/wiki/Bento_Rodrigues_dam_disaster`). Very simplistically, fallouts from hydroelectric structures can be more critical than the ones related to thermal, wind and solar plants.

1.3 Electricity production management

Electricity production management aims at planning ahead which production levers to pull in order to satisfy demand in electricity as well as other requirements (as mentioned in 1.2.2: reliability, costs, scarce resources, etc.).

There is no unique framework to manage the production of electricity: levers and requirements vary considerably according to the setting considered. In this section, we will describe some aspects of the electricity production management at EDF that remain relevant for other generating companies.

1.3.1 Specificities of electricity

The electric system is basically composed of two intertwined layers: the physical exchanges of electricity and the related financial trades. At one end, producers – also commonly referred

to as generating companies (GenCos) – own or lease physical generating assets – that is to say power plants – to produce electricity, physically provide it on the grid and sell it to end users that consume it at the other end of the grid. For a generating company, the distribution of production assets according to their types is called the energy bundle – or energy mix. For the demand side, the aggregation of all electric consumptions at a given time is called the load. The grid is a network of nodes – sometimes denoted buses – connected by transmission lines. Buses correspond to injection points for generating companies or consumption points for end users. Safe and reliable operation of the electric system is ensured by Transmission System Operators (TSOs) and Distribution Network Operators (DNOs). The roles of and interactions between the major stakeholders involved in the electric system are presented in the Appendix A.2.

The electric system is demand-driven as the satisfaction of demand is essential: electricity is a good/service that is essential for human activity. It is interesting to note that electricity is an inevitable expense, thus its cheapness is also critical for the purchasing power of households and the bottom line of organizations. The satisfaction of the demand in electricity has to be instantaneous: end-users do not order it in advance nor are they willing to wait for a delayed delivery. The satisfaction of demand cannot be partial; for example, most electrical devices have technical specifications that forbid them to operate at half power. The satisfaction of demand is not flexible; a few end-users can curtail their consumption on demand, but, generally-speaking, there are less levers to control demand, compared to production.

The electric system is subject to global requirements. Not only satisfying demand is essential for its usage, it is also necessary for the reliability of the network. Indeed, the electrical network is somewhat fragile: a surge or a shortage of supply compared to demand can cause disturbances and even damages to production/transmission/end-users appliances. Sometimes the last resort is load shedding, that is temporary shutdowns for parts of the network. The equilibrium of supply of producers and demand of consumers is the primary requirement the TSO must pursue. Safety and reliability of the network is subject to more elaborate requirements (congestion and capacity of lines, power losses, frequency adjustment, etc.) that we will not explain further. On the grid, electricity a non-differentiable good, therefore the overall equilibrium is satisfied if each producer meets the demand of its own clients. Note also that, compared to the number of end-users, there are few power plants, mainly due to economies of scale. Those plants are owned/operated by few prominent players on the production side; their responsibilities in guaranteeing the equilibrium is therefore paramount. In case of mismatch in production and demand on the network, prominent players are bound to provide all the leeway they have on demand of the TSO in order to counter-balance the disequilibrium.

Production must be planned ahead in order to meet system requirements and satisfy a highly variable and uncertain demand. An electricity producer cannot rely on an inventory of past ex-

cess production because electricity is hardly storable at a large scale. Additionally, an electricity producer cannot resort to backlogs to match the demand of its consumers since consumption cannot be delayed. The usual stock equation, as derived in Lot-Sizing problems [76] for example, does not hold:

$$\forall t,\ \text{production}_t + \text{stock}_t + \text{backlog}_{t+1} = \text{demand}_t + \text{stock}_{t+1} + \text{backlog}_t.$$

Instead, we have:

$$\forall t,\ \text{production}_t = \text{demand}_t.$$

Where:

- production_t and demand_t denote respectively the electricity production and the demand in electricity at time t,
- stock_t and backlog_t denote respectively the cumulative past surplus production and the cumulative unsatisfied demand at time t.

Another related implicit distinction with the usual stock equation is that the equilibrium must be satisfied in real-time and not over time windows. An electricity producer must coordinate the productions of the several plants that are part of its energy bundle ($\text{production}_t, \forall t$). Due to the speed and magnitude of demand variations, production cannot track demand *ex post*. Indeed large real-time changes in the production of power plants are not most appropriate because of lack of technical flexibility, because they are economically prohibitive, and/or because they are restricted by the available resources. It is more sensible to anticipate by setting production levels ahead to meet forecast demand while making sure minor adjustments can be controlled on-the-go to meet real-time demand.

1.3.2 Different stages of production management

We will now focus on the production management of electricity. Though not mentioned explicitly previously, an electricity producer – as a company – aims at using efficiently its resources to make a profit while staying within a given threshold of risk. Those financial requirements are supplemented by specific requirements related to the electric system – supply-demand equilibrium and stability of the network more generally – and related to complexity of the production process.

The payoff of a generating company is derived as follows: revenues come from the sales to end users and on wholesale markets, costs come from investment, production and buying on wholesale markets.

A generating company must cope with several unknowns or risks: the regulatory context is likely to change over the lifetimes of facilities; markets prices and depths are uncertain; demand is uncertain; production is affected by uncertain factors such as technical failures, maintenance durations, and water precipitations into reservoirs of hydroelectric plants.

Decisions relative to production management can be decomposed according to the usual strategic/tactical/operational planning paradigm [2]. Though the demarcations are fuzzy, this paradigm coincides with the chain of long-term/mid-term/short-term planning horizons.

Long-term planning deals with equipment investment decisions to ensure the energy bundle is adapted to offer enough capacity and maneuvrability to satisfy future demand. The installed production facilities are then considered frozen for the mid-term and the short-term planning horizons.

Mid-term planning deals with the design of management policies such as deciding on maintenance campaigns for plants, stock policies – that are enforced either with guide-curves or through opportunity cost indicators – and hedging policies. Such policies then bind or at least influence the possible courses of actions for the short-term horizon.

Short-term planning deals with actual scheduling of power plants and is described in further details in Section 1.3.3.

In the very short term, adjusting the previously computed schedules to the realization of uncertainties is denoted rescheduling [62].

1.3.3 Short-term electricity production scheduling

Short-term scheduling deals with planning horizons ranging from one day to a couple of weeks and discretized in time periods themselves ranging from a few minutes to a few hours.

Scheduling is carried out for plants that are dispatchable: hydroelectric and thermal plants. When technically possible, non-dispatchable plants – windmills, solar plants and isolated run-of-the-river plants – can be disconnected from the grid which is most frequently economically irrelevant since their production is free. This kind of production is denoted inevitable and is not subject to scheduling. The contribution to the overall electricity provision can be estimated thanks to sunlight and wind forecasts.

At every time period, two kinds of decisions have be to made in a schedule of controllable plants: its commitment status – that is whether the plant is on or off – and its power level – that is how much power is produced. Scheduling commitment statuses of plants – or units within plants – is denoted unit commitment (UC). Scheduling power levels once commitment statuses are known is denoted economic dispatch (ED). Scheduling the two decisions together is more relevant and can also be denoted UC by extension.

Regarding power, we are only interested in the quantity produced and provided on the grid,

and we disregard other power characteristics that are essential for transmission planning such as voltage, frequency and phase; indeed we assume such characteristics satisfy by default system requirements and they can be adjusted by the TSO to ensure system stability.

Three settings can be distinguished for unit commitment: security-constrained [33], price-maker bidding [24, 54] and price-taker bidding [68] scheduling. We will be interested in the former two and a description of the three is given in the Appendix A.3. Security-constrained unit commitment (SCUC) is centralized: all the power plants must be scheduled jointly so that the aggregation of their production levels meets the demand, thus ensuring the security of the system. Given market prices, a price-taker producer self-schedules – that is to say regardless of an external security constraint – its production with a view to maximizing its payoff. The aim of a price-taker producer in the self-scheduling setting is to maximize its payoff. Since there is no requirement binding the different power plants, nor do schedules for one plant affect payoff of other plants, the scheduling can be carried independently for each plant, in a decentralized fashion. This kind of scheduling problem is sometimes referred to as price-based unit commitment (PBUC).

On top of the previous distinction, short-term scheduling problems may differ according to the following features:

- the plants considered: nuclear, conventional thermal, hydroelectric, etc.

- the kinds of costs considered: fixed/variable, setup/startup/shutdown, etc. Fixed costs are incurred independently of the production level; setup costs are fixed costs proportional to the duration of operation of a plant while startup/shutdown costs are fixed costs only incurred at the time a plant is started up or shut down. Variable costs refer to costs that vary as the production level varies.

- the representation of plants' operations: I/O curves, technical constraints, etc. I/O curves relate variables of interest (power, fuel stock, water flow, variable cost, etc.). For thermal plants for instance, such function gives the variable cost according to the generated power. Technical constraints include minimum up and down times, ramp rates, and other limitations on the modulation of operations.

- the consideration of uncertainties: demand forecasts, hydraulic inflow forecasts, technical breakdowns, electricity prices can be considered known or uncertain. When such parameters are considered uncertain, their representation may be vary: they may take a finite number of values based on scenarios, they may be random variables with known distribution, they may be uncertain within a known range.

- the consideration of other system/contextual/environmental requirements: plant availabilities, cap on pollution emission, etc.

We will give more details for the case of deterministic hydroelectric short-term scheduling.

1.3.4 Short-term hydroelectricity production scheduling

Short-term hydroelectricity production scheduling is also referred to as short-term hydro scheduling (STHS) or hydro unit commitment (HUC).

Water can be considered as a free fuel, but it is a limited resource – reservoirs have a finite capacity – and it cannot be ordered – inflows come from precipitations. From a financial perspective, the main question is to find the optimal time to use the water stored in reservoirs. Since water cycles are seasonal, this question is partly answered by the stock policies designed at the mid-term planning stage. The decision variables of the mid-term planning stage become set parameters at the short-term planning stage. Short-term planning deals with designing schedules that follow mid-term stock policies. Reservoir stock policies are twofold: there are guide curves and water (usage) costs. Guide curves set target levels of reservoirs at regular time milestones. Water usage costs – or water value – are dummy opportunity costs: they reflect the value of the optimal use of water in the future instead of using it now. For example, if market prices are lower than water costs, the best tradeoff is not to discharge water now because it will be more profitable to use that water later, in other words remuneration of a short-sighted present production (at market price) is lower than the anticipated remuneration of the same amount of production in the future (water cost).

Whether for the design of stock policies or for the scheduling of daily operations, purposes other than financial are considered: reliability of the materials, flood control for the safety of downstream populations, water supply, recreational activities, agricultural irrigation or industrial use.

While thermal plants somewhat operate independently from one another, hydropower plants are part of multi-reservoir systems – or hydro systems, or *valleys*, that is to say a network of interconnected reservoirs and hydroelectric plants. Water taken from an upstream reservoir is released through a plant and, after a time lapse, feeds a downstream reservoir. For this reason, dynamically interconnected plants require joint planning. Joint planning is more complicated because there are more schedules to consider at once and because we must make sure those schedules are compatible with respect to flows, inflows and reservoir capacities.

Let us describe further the payoff structure. Revenues are the remuneration of the power supplied at the market price. Dummy costs are incurred for using water. Sometimes, extra energy is required to launch or to stop a turbine, paying for this energy incurs startup or shutdown

costs. The abrupt changes related to startups and shutdowns cause degradation of the materials; therefore startup and shutdown costs can also reflect the amortization of the equipment. On a daily basis, for a plant that is running, costs for maintaining or setting up the plant have already been paid for and/or are paid independently of the schedule of operations: Such costs are sunk costs and are disregarded at this time scale although they are main drivers for mid-term planning.

Scheduling is done according to plants' operations. Plants' operations are represented with more or less approximations according to the required level of realism. Representations include I/O curves – or mappings or characteristics. Input variables are the the water flow released through a plant, the water level in the reservoir – or head; actually water flow can be separated in two types: *active* water flow that really produces power, and *spillage* – that is water release without production of power. There is a correspondence between head and stored water volume that depends on the shape of reservoir. Output is usually the power produced; it can also include operating reserve, that is to say the available increase/decrease in power that can be activated on short notice. On top of maximum flow and power capacities, operations may be restricted by forbidden zones in the I/O characteristic; they correspond to undesirable vibrations or damage of the equipment. To limit wear and tear of the equipment and ensure a sustainable production, abrupt changes in operations are also penalized or restricted. We have already mentioned penalties on startups and shutdowns, there are also ramp-up and ramp-down bounds when increasing or decreasing the water control. Besides, steady smooth operations are preferred to jagged ones.

1.4 Optimization for electricity production scheduling

Unit-commitment (UC) is a problem with explicit requirements: compute a schedule of operations that is optimal according to an economic criterion and that complies with specific requirements (such as demand or stock policies). In addition, notions of interest can be formalized with quantitative variables: operations – *e.g.*, how much power is produced? – the economic criterion – *e.g.*, what is the cost operations? – and other requirements – *e.g.*, what is the forecasted demand? Therefore, it is relevant to formalize UC as an optimization problem.

In the general setting, the UC problem can be cast into a large-scale non-convex mixed-integer non-linear programming problem. More specifically, binary variables allow to model logical patterns: ON/OFF states, startups and shutdowns [63], min-up/min-down constraints [46] disjunctive constraints in the operations of subsets of units (*e.g.*, exclusion of simultaneous use of pump and turbine as described in [34]). In addition, mathematical functions allow to model dependencies such as remuneration/cost of production, stock depletion, I/O characteristics. Those functions may be non-linear and non-convex; for example, I/O characteristics – relating flow and water level to power output – for hydroelectric plants are non-linear and non-

convex. The UC problem involves several time periods, and often several units are considered, which increases the dimension of the problem.

It is relevant to translate UC into an optimization problem and it is also relevant to use optimization methods when rules of thumbs or human reflection are limited with respect to the problem's complexity and size.

1.4.1 State of the art of unit commitment

In a broader perspective, UC has been subject to a large research activity due to its practical importance (see surveys [69, 56, 45, 72, 73]). However, it can still not be considered as a well-solved problem. Academic contributions to the subject are very diverse and differ in problem types, models, solution methods, solution assessment, size of test set (if any). To be more specific, we focus on UC in a deterministic setting, with a bias towards hydropower UC (HUC) problems, and we decline a non-comprehensive state of the art by solution method.

Dynamic programming (DP) (see [8] for example) is one of the classical approaches for solving UC. The major interest is to allow for non-linearity and non-convexity aspects. The major drawback is the scalability: time and memory requirements grow rapidly with the dimension of the problem; this phenomenon is often referred to as the curse of dimension. When dealing with states that depend on continuous quantities, a tradeoff needs to be found between the accuracy of the discretization mesh and the size of the discretized problem. To alleviate the curse of dimension, several approximations have been considered with moderate success (see [72] for references on the different techniques used). Although there have been attempts to solve UC problems with a DP scheme on a large scale, such a technique is more appropriate to solve problems on a smaller scale, like subproblems in a decomposition scheme. More specifically for HUC, a DP scheme is proposed in [5] for an optimal dispatch of turbine units of the Itaipú hydropower plant; the hydropower plant considered is composed of 18 parallel turbine units below a single reservoir fed by the Paraná river. The results emphasize the importance of considering individually each turbine unit and its startup/shutdown.

Network flow models [3] are fitted to represent the structure and dynamics of valleys. In addition, network flow algorithms, that are adapted for network structures by definition, are very efficient. However, only linear or convex piecewise linear dependencies related to flows and volumes can be modeled. Including other features, such as non-convex non-linearities, discontinuities, or general constraints, like demand or ramping requirements, while exploiting the network structure is non-trivial; such extensions usually necessitate approximations [27].

Mixed-Integer Linear Programming (MILP) is another well-accepted modeling and solving technique used for UC problems and also for HUC problems (see references in surveys [72, 73]), especially since MILP solvers have proven their efficiency. The nice features for using it are

that constraints can be easily added and that non-linearities can be approximated as piecewise linear functions. We refer the reader to [21, 10] for a detailed approximation of the head effect. Furthermore, the combinatorial aspects of the problem can be modeled using discrete variables. However, the inherent complexity of the solution technique provides no guarantee in terms of time it takes to reach optimality; similarly to DP, though to lesser extent, MILP solution methods are less and less performant as the size or the complexity of the model grows. Even worse, finding a feasible solution could be just as difficult and could even be impossible. As reported in [57], most existing MILP models fail when trying to solve directly a UC problem without an initial feasible solution. In Chapter 2, we consider a MILP approach to solve the HUC problem as proposed in [34]. Note that head effect is ignored and we consider a piecewise linear approximation of the univariate I/O characteristic: power only depends on the released water flow.

1.4.2 The unit commitment problem at EDF

At EDF, day-ahead scheduling is carried everyday for the two upcoming days and only the first day of the schedules is enforced; this process helps avoiding being blind to the future, and is complementary to mid-term planning policies.

EDF operates several thermal plants – nuclear and fossil-fired – and hydroelectric plants within valleys. Day-ahead scheduling is solved out as a Security-Constrained UC problem where net forecasted demand and other security requirements have to be met. The computed schedules are communicated to plant operators and to the TSO. For this large-scale UC problem (see [64] for more details), the solution method is a price decomposition based on a Lagrangian Relaxation (LR) [34].

Lagrangian Relaxation (LR) (see [29] for example) is certainly the most commonly used approach to solve large-scale UC problems, as stated in [72]. The major advantage of LR techniques is to solve a related problem – the dual – where binding constraints are relaxed thus allowing to solve subproblems independently and hopefully more efficiently than the whole original problem [16]. For EDF's UC, the main idea is to take advantage of its special structure, where all thermal plants and valleys are coupled through demand constraints. Such global constraints are in rather limited number – a few per time period – compared to the number of local constraints relative to the operations of plants and dynamics of valleys – at least proportional to the number of plants and reservoirs. Since subproblems are solved several times, the advantage of decomposition is maintained if they can be solved much more efficiently and if the algorithm that updates the Lagrange multipliers of the dual function converges fast enough. Each valley (respectively each thermal power plant) is considered as a PBUC subproblem and is solved using mixed integer programming (respectively dynamic programming), as presented in

[34]. Lagrangian multipliers are traditionally updated using a subgradient method [35].

A characteristic of LR techniques for solving non-convex programs is that the dual problem is a convex approximation, thus there are guarantees of convergence for that problem. However, the optimal solution of the dual problem may not be feasible for the primal problem when the dualized constraints are not satisfied. Different approaches have been used to recover a feasible solution either by appropriately modifying the objective function [25] or by a heuristic search phase [64, 67]. Besides, the Augmented Lagrangian [19] method as an extension to the standard LR approach is used as a second stage to recover feasibility for EDF's UC. The principle is to add a quadratic penalization of the relaxed constraints alongside the linear weight of standard LR to the objective function. Since the quadratic term compromises the decomposability property, a partial linearization is used.

1.4.3 Challenges

UC and HUC problems are difficult optimization problems. Regarding theoretical complexity, a kind of UC problem is proved to be NP-hard in [75]. The generic challenges in developing solution methods for optimization problems later introduced in Section B.5.4 in the Appendix B are also encountered for HUC problems. Besides, there are modeling challenges that are specific to the HUC application. As we will show, issues related with problem modeling and problem solving methods are very often intertwined.

Computational performance Essentially, short-term problems like UC problems need to be solved in the short term hence very quickly. For example, if the planning horizon is one day, a schedule must be computed every day; the closer we are to the time initiation of the next planning cycle, the more information we have – on plant statuses, demand forecasts, market prices, etc. – and the closer to reality the schedule will be. From a practical viewpoint, schedulers need to compute several schedules under different predicting hypotheses to anticipate uncertainties, or need to rerun the solution process as updates on unforeseen events come along the day. Also, HUC may be a subproblem of larger problems and must therefore be solved several times. For example, the Lagrangian decomposition for a SCUC problem requires the iterative solution of HUC subproblems. All in all, UC problems require fast solutions.

Accurate modeling By definition, a model is a simplified representation of reality, but, as a rule of thumb, the more elaborate a model is the least tractable the corresponding problem becomes.

Restricting the size by considering finite time horizons, or considering independently plants that are not, or aggregating units within a plant, is a basic kind of simplification. Given the

performance of available solution methods, such simplifications are sometimes necessary.

Even with a fixed size, optimization problems become harder to solve when the representation of the systems is very detailed because it usually implies more variables, more constraints, more difficult characteristics (discontinuities, non-convexities, non-linearities).

We must choose which features of the real system are meaningful to model, and how to model them given that different formulations lead to optimization problems with different properties, for which there are different classes of optimization algorithms, which have different computational performances.

Optimality As most schedules are evaluated with respect to a financial criterion (maximizing revenues, minimizing production costs), there may be a substantial difference in profits or savings between a good schedule and the best schedule; optimality is thus essential for every generating company.

Actually, optimality of solutions is essential for all stakeholders in a market environment. For example, when a company with market power overestimates its production costs in its bids because of suboptimality, the clearing price is shifted up, at the expense of buyers.

Optimality is related to performance and modeling. Solutions methods take longer to find the optimal solutions and to prove those solutions are actually optimal. The payoff prescribed by the optimal solution of an optimization model may be different from to the realized payoff *ex post* because of inherent model simplifications. However, solving a more refined model to optimality may not be easy. We must consequently strike a balance between optimal solutions of an approximate model and suboptimal solutions of a more realistic model.

Also, optimality is related to feasibility in LR decomposition schemes. Indeed, optimality of subproblems' solutions is necessary for the convergence to optimality of the dual problem which is necessary for the feasibility of the dualized constraints.

Feasibility When it is hard to obtain optimal solutions in a limited timeframe, we would like to settle at least for feasible ones. Indeed infeasible solutions may be completely irrelevant. Infeasible solutions could for example produce schedules where reservoirs have negative water stocks and would not be enforced. As a matter of fact, the workshop of practitioners organized by the authors of [37] deemed "dynamic feasibility" as a "very important" feature models should capture. However, mathematical programs for unit commitment involve multiple constraints with intricate variables, therefore it is not always trivial to know whether a problem is feasible or infeasible *a priori*. Proving or disproving the feasibility of problems can require a lot computational efforts, problems can remain undetermined when solution methods fail to terminate. Feasibility is really a performance issue: BB search strategies or the invocation of heuristics are

quite different whether we want to prove or disprove infeasibility.

For example in [79], the authors derive a necessary and sufficient condition that allows to check feasibility of a hydro unit commitment problem; checking this condition allows to avoid unnecessary computations in case of infeasibility and also to build a feasible schedule in case of feasibility. Several units, each with several operating states, fed by a single reservoir are considered. The condition is twofold but relates only to joint feasibility of water balance and reservoir capacity: given an intermediate commitment schedule, past feasibility is checked and we can know if the completion of the schedule will necessarily be infeasible. This condition is used within a branch-and-bound scheme to implicitly enumerate feasible schedules or to prove no feasible one exists. The method cannot however be directly extended to the problem we study in Chapter 2 as dynamic constraints such as ramping rates and multi-reservoir interacting flows are not taken into account.

Beyond the computational burden required to obtain feasible solutions, we must sometimes deal with problems that are declared infeasible. When a problem is infeasible, it is even less trivial though very useful to detect the sources of infeasibilities. For a given infeasible problem, an Irreducible Infeasible Subset of constraints (IIS) is a set of constraints of the main problem which is infeasible, but for which any subset is feasible [32]. According to Lodi in [50], isolating IIS for an infeasible Linear Programming (LP) is NP-hard on its own; consequently the MILP case is at least as hard. Analyzing the sources of infeasibility of an infeasible program are useful to *repair* a model when we are interested in solving a problem supposed to be feasible. Physical phenomenon happen, therefore real-world problems, if properly modeled, should be feasible. When infeasibility occurs, we must identify its causes in order to lift it. Causes of infeasibilities include: computing errors, data errors and model inconsistencies.

Model inconsistencies may be due to inaccurate modeling in the sense that the behavior of system under study is not accurately modeled or that the prescriptions on that system are not accurately modeled. For example, ensuring smooth operations to preserve the equipment or following mid-term policies are important requirements but are sometimes translated into overly restrictive constraints that happen to make problems infeasible though they should not be satisfied at the expense of feasibility. Ideally, satisfaction of such requirements can be quantified and considered as secondary goals. In that case, a multi-objective approach could allow to compute several schedules with different tradeoffs between short-term, mid-term and reliability issues in order to support schedulers in their decision process [15].

Numerical issues, numerical inaccuracies Firstly, we are interested in solving numerical-valued problems, not just theoretical ones. Secondly, mathematical optimization often does not provide analytical solutions but solutions as outputs of several calculations performed as

numerical computations. Thirdly, numerical computations often rely on finite-precision floating-point arithmetic. Computational performance is about time and also about precision of results. When compared to infinite-precision rational computations, finite-precision implementations are much faster, require less memory, often result in near-feasible near-optimal solutions without guarantee of correctness [26].

Correctness (of feasibility, or of optimality) is not certified as numerical solvers include tolerances to cope with the round-off errors inherent to finite-precision computing. Indeed approximate results are often satisfactory. Also we are not necessarily interested in the exact result when input data is itself already inaccurate. Indeed, solutions to UC problems are computed in large decision-support systems that are subject to data corruption implying that solution methods must be robust to input data errors [13].

Yet, the authors of [42] warn us that "even when numerical instability does not prevent the optimizer from solving the model to optimality, it may slow down performance significantly". Exact results are sometimes necessary "to establish fundamental theoretical results and in subroutines for the construction of provably accurate cutting planes" as pointed out by the authors of [22]. When dealing with infeasible models, imprecise data and approximate solvers, exact results enable to detect sources of infeasibilities to calibrate data precision according to tolerances and to reformulate the model.

To illustrate the relatively recent interest in numerical issues, the library MIPLIB 2010 [44] of MILP instances that are challenging to solve for several benchmark general-purpose solvers contains an *UNSTABLE* category of instances which exhibit bad numerical properties. In addition, the developments of the exact LP solver QSOPT-EX [26] and the exact MILP solver SCIP-EX [22] are also quite recent. The treatment of numerical issues remains a promising direction of investigation as Lodi wishes for "a tool for detecting minimal sources of numerical instability" [50] among the desirable future advances in MILP computations.

Chapter 2

Real-world hydropower unit commitment: data and model pre-processing for infeasibilities

In this chapter, our objective is to effectively solve real-world instances of the Hydropower Unit-Commitment (HUC) problem – that is to say obtain feasible instances. The problem comes from EDF's day-ahead production scheduling process of multi-reservoir systems. Mathematically speaking, the problem is formulated as a Mixed-Integer Linear Programming (MILP) model. In practice, infeasibility situations often occur when solving this HUC problem. Infeasibilities of real-world problems indicate that consistency of model and data needs to be reviewed so that the problem admits a solution. Our aim is thus to analyze more precisely the sources of infeasibilities in order to propose data and model pre-processing.

Issues affecting real-world data and numerical computing, together with specific model features, make our problem harder to solve and often infeasible. The first concerns arise when determining whether an instance is feasible or infeasible because computations could be affected by numerical errors. Indeed, the floating-point representation of large (volumes) and small (flows) variables introduces rounding errors and the data from the real-world instances could be inaccurate. Because reservoirs can be regarded as integrators, such errors may be amplified and eventually alter a computed feasibility result. Compared with a standard HUC problem, we have essentially two additional model specifications that are induced by discrete operational constraints and by water management policies. The first specification requires that the schedule for each plant should operate over a discrete set of operational points. The second specification requires that each reservoir level should meet target volumes.

We propose a step-by-step approach: the idea is to systematically isolate practical difficulties

– rounding errors, data errors, data inconsistency, model infeasibilities – to cope with them one at a time. First, we present a complete model, which incorporates all operational specifications, and illustrate through experiments a few of the aforementioned practical difficulties. Secondly, we derive a simple model to analyze numerical errors; the simple model preserves the basic continuous characteristics of a standard HUC problem, *e.g.*, conservation of water and bounds on reservoirs, but removes the discrete operational requirements. As we focus on feasibility issues, the key idea is to design the simple model so that its feasible set contains the feasible set of the complete model. An exact solver is used to check consistency of the results obtained using a floating-point solver and to detect rounding errors. The effects of data errors on feasibility is mitigated by the introduction of marginal corrective slacks in the model. Once numerical issues are dealt with, consistency of model and data is checked with respect to both additional specifications – discrete operations and target volumes. The derivation of several relaxations of the complete model allows us to analyze and classify infeasibilities. Finally, while data inconsistencies are discarded, model infeasibilities due to target volumes are eliminated using a 2-stage method. In the first stage, a minimal deviation from target volumes is computed to make the problem feasible. In the second stage, the original HUC problem is solved with a possible deviation from the target volumes as defined in the first stage.

This chapter is organized as follows. Section 2.1 is devoted to the description of the complete model for the considered HUC problem. In Section 2.2, first computational tests are performed using a MILP solver as a black box on a test set of real-world instances, thus illustrating a high occurrence of infeasibilities. The simple model is derived in Section 2.3. Numerical errors are introduced in Section 2.4 and analyzed by resorting to an exact solver in Sections 2.5 and 2.6. An analysis of the different sources of infeasibilities is carried out in Section 2.8 and a 2-stage method with feasibility recovery is proposed in Section 2.9; final experimental results show its effectiveness on the original test set considered. In Section 2.10, some concluding remarks and future work directions are drawn.

2.1 Model description

We focus on the hydropower Unit-Commitment (HUC) problem as it appears for the scheduling of EDF's multi-reservoir hydropower systems in the short term. The interested reader can refer to [78, 45, 73] for surveys on the HUC problem. As for the problem we deal with, we consider a price-taker scheduler in a deterministic setting. Plant availabilities, power prices, water values and external reservoirs inflows are given parameters. The planning horizon is set to two days and it is uniformly discretized in 30-minutes time periods. The objective of this Profit-Based UC (PBUC) problem is to maximize power revenues and value of saved water. Power output

only depends on water flow and does not depend on reservoirs' water levels, *i.e.*, *head effect* is ignored; power-flow curves feature discrete points and affine segments. In addition, several plant operational constraints must be satisfied and water levels in reservoirs must lie within bounds given by their capacities.

We now present in details the *complete* mathematical model starting by introducing notation. It is denoted *complete* as opposed to the *simple* one introduced in Section 2.3.

2.1.1 Notation

In this section, we present sets and indices, parameters and variables. For proprietary reason, we introduce units derived from International System (IS) base units. From a practical point of view, we used decimal multiples and submultiples for the parameters and variables values to be within an appropriate range for numerical precision (see Section 2.5 for more details).

Sets and indices

- $t \in T = \{1, \ldots, \bar{t}\}$: time period index and set. For simplicity, we assume there is an even number of time periods $\bar{t}$.

- $r \in R$: reservoir index and set.

- R^C, R^T: set of reservoirs without storage capacity, set of reservoirs managed with target values and water usage value.$\{R^C, R^T\}$ is a partition of R. Reservoirs in R^C represent structures that contain water but that cannot store it properly like very small basins or canals; such reservoirs are found upstream of run-of-the-river plants where all incoming inflow must be discharged; such structures are still referred to as reservoirs for the sake of simplicity though it is somewhat improper with respect to actual reservoirs in R^T.

- $i \in I$: plant index and set.

- I^C, I^D, I^G, I^P: set of plants with continuous operations, set of plants with discrete operations, set of generating plants, set of pumping plants.
  $\{I^C, I^D\}$ is a partition on I, and $\{I^G, I^P\}$ is also a partition of I. Note that the generating plants release water to produce power while pumping plants consume power to pump water. The convention chosen here is to model power with algebraic values (positive or negative) and flow with absolute values along with a sign that depends on the direction of the flow.

- $I^R \subset I^G \times I^P$: set pump-storage stations. These stations represent a couple of plants, say (i', i''), with a generating plant i' and a pumping plant i'' that are in between the same upstream reservoir and the same downstream reservoir.

- $j \in J_{ti}$: operational point index and set of operational points of plant i at time period t ($\forall t \in T, \forall i \in I$). We assume w.l.o.g. that $|J_{ti}| \geq 1$. The set of possible operational points may vary in time as operators may plan to change the settings of the operations prior to scheduling for the planning horizon.

- $R_i^+, R_i^- \in R$: upstream reservoir and downstream reservoir of plant i ($\forall i \in I$).

- $I_r^+, I_r^- \subset I$: set of upstream plants and set of downstream plants of reservoir r ($\forall r \in R$).

Parameters

- Π [s]: period duration.

- Ω_i[expressed in number of time periods]: time lapse for water to flow through plant $i \in I^G$ to downstream reservoir R_i^- once water is released from upstream reservoir R_i^+ or to be pumped up through plant $i \in I^P$ from downstream reservoir R_i^- to upstream reservoir R_i^+ ($\forall i \in I$).

- $\overline{\Psi}_r, \underline{\Psi}_r$ [m^3]: maximum and minimum volume capacities in reservoir r ($\forall r \in R^T$).

- $\overline{A}_r, \overline{B}_r, \underline{A}_r, \underline{B}_r$ [m^3]: mid-horizon maximum, final maximum, mid-horizon minimum and final minimum target volumes in reservoir r ($\forall r \in R^T$).

- Γ_{tr} [m^3/s]: external water inflow in reservoir r during period t, that is to say available at end of period t ($\forall t \in T, \forall r \in R$); inflows can be positive for precipitations or runoff water; inflows are negative for evaporation or water extraction.

- $\overline{\Upsilon}_i, \underline{\Upsilon}_i$ [m^3/s]: maximum water flow ramp-up and maximum water flow ramp-down between two consecutive time periods at plant i, respectively ($\forall i \in I$).

- Φ_{tij} [m^3/s]: water flow increment released at time period t, through plant i, corresponding to operational point j ($\forall t \in T, \forall i \in I, \forall j \in J_{ti}, \Phi_{tij} > 0$).

- $\overline{\Sigma}_{ti}$ [m^3/s]: maximum spillage allowed during time period t at plant i ($\forall t \in T, \forall i \in I$). To have a consistent notation, spillage is artificially introduced for pumps and $\forall t \in T, \forall i \in I^P, \overline{\Sigma}_{ti} = 0$.

- Λ_{tij} [W]: power increment generated at time period t by plant i corresponding to operational point j ($\forall t \in T, \forall i \in I, \forall j \in J_{ti}$). Head effect is assumed negligible as water level in reservoirs is almost invariant over the planning horizon.
- ω_r [currency/m^3]: water value of reservoir r ($\forall r \in R^T$). Similarly, water value is assumed constant with respect to volume.
- λ_t [currency/W per time period]: price for power at time period t ($\forall t \in T$).
- $\overline{Q}_{ti} = \sum_{j \in J_{ti}} \Phi_{tij}$ [m^3/s]: maximum possible water flow released at time period t, through plant i ($\forall t \in T, \forall i \in I$).
- $\underline{Q}_{ti} = \begin{cases} \Phi_{ti1} & \text{when plant must operate and release a non-zero flow} \\ 0 & \text{otherwise} \end{cases}$ [m^3/s]: minimum possible water flow released at time period t, through plant i ($\forall t \in T, \forall i \in I$).
- $\overline{P}_{ti} = \max_{k \in J_{ti}} \sum_{j=1}^{k} \Lambda_{tij}$ [W]: maximum possible power generated during time period t by generating plant i ($\forall t \in T, \forall i \in I^G$).
- $\underline{P}_{ti} = \min_{k \in J_{ti}} \sum_{j=1}^{k} \Lambda_{tij}$ [W]: its absolute value correspond to maximum power consumed during time period t by pumping plant i ($\forall t \in T, \forall i \in I^P$).

2.1.2 MILP formulation of the complete model

This section is devoted to the description of the complete mathematical model. We start with bounds on variables, continue with the constraints, and finally introduce the objective function.

By convention, sums on empty sets are equal to zero.

Variables

- v_{tr} [m^3]: water volume in reservoir r at end of time period t ($\forall t \in T, \forall r \in R$).
- q_{ti} [m^3/s]: water flow released through plant i during time period t ($\forall t \in T, \forall i \in I$).
- s_{ti} [m^3/s]: water flow spilled through plant i during time period t ($\forall t \in T, \forall i \in I$).
- p_{ti} [W]: power generated by plant i during time period t ($\forall t \in T, \forall i \in I$).
- x_{tij}: binary activation status of discrete operational point j of plant i during time period t ($\forall t \in T, \forall i \in I, \forall j \in J_{ti}$).
- y_{tij}: continuous operational point j of continuous-operating plant i during time period t ($\forall t \in T, \forall i \in I^C, \forall j \in J_{ti}$).

- z_{ti}: binary variable allowing spillage at plant i in time period t ($\forall t \in T, \forall i \in I$).

To simplify notation, variables are not defined for initial values (for $t = 0$ or -1) but they may be introduced when needed with specific values.

Variable bounds

Now we list the simple bounds on variables introduced in the previous section.

$$\underline{\Psi}_r \leq v_{tr} \leq \overline{\Psi}_r \qquad \forall t \in T, \forall r \in R \tag{2.1.1}$$

$$0 \leq \underline{Q}_{ti} \leq q_{ti} \leq \overline{Q}_{ti} \qquad \forall t \in T, \forall i \in I \tag{2.1.2}$$

$$0 \leq s_{ti} \leq \overline{\Sigma}_{ti} \qquad \forall t \in T, \forall i \in I \tag{2.1.3}$$

$$0 \leq p_{ti} \leq \overline{P}_{ti} \qquad \forall t \in T, \forall i \in I^G \tag{2.1.4}$$

$$\underline{P}_{ti} \leq p_{ti} \leq 0 \qquad \forall t \in T, \forall i \in I^P \tag{2.1.5}$$

$$x_{tij} \in \{0, 1\} \qquad \forall t \in T, \forall i \in I, \forall j \in J_{ti} \tag{2.1.6}$$

$$y_{tij} \in [0, 1] \qquad \forall t \in T, \forall i \in I^C, \forall j \in J_{ti} \tag{2.1.7}$$

$$z_{ti} \in \{0, 1\} \qquad \forall t \in T, \forall i \in I \tag{2.1.8}$$

Constraints

Water conservation We start by presenting the constraints modeling the conservation of water at time period t and reservoir r ($\forall t \in T, \forall r \in R_T$):

$$\begin{aligned} v_{tr} = v_{(t-1)r} &+ \Pi\Gamma_{tr} \\ &+ \Pi \sum_{i \in I_r^+ \cap I^G} (q_{(t-\Omega_i)i} + s_{(t-\Omega_i)i}) \\ &+ \Pi \sum_{i \in I_r^- \cap I^P} (q_{(t-\Omega_i)i} + s_{(t-\Omega_i)i}) \\ &- \Pi \sum_{i \in I_r^- \cap I^G} (q_{ti} + s_{ti}) \\ &- \Pi \sum_{i \in I_r^+ \cap I^P} (q_{ti} + s_{ti}). \end{aligned} \tag{2.1.9}$$

v_{0r} and $(q_{ti}, s_{ti})_{t \leq 0}$ are replaced with their initial values.

The water volume in the reservoir is equal to the water volume at the previous time period plus the external forecasted inflows, the water released from the upstream plants (first summation), and pumped from the downstream plants (second summation), minus the outflows released

by the downstream plants (third summation) and pumped by the upstream plants (fourth summation). The constraints take into account the time lapse needed for water to reservoir r.

For each reservoirs r without storage capacity ($\forall r \in R^C$), the water volume conservation constraint at each time period t ($\forall t \in T$) is slightly different:

$$\begin{aligned}\Gamma_{tr} + &\sum_{i \in I_r^+ \cap I^G} (q_{(t-\Omega_i)i} + s_{(t-\Omega_i)i}) \\ &+ \sum_{i \in I_r^- \cap I^P} (q_{(t-\Omega_i)i} + s_{(t-\Omega_i)i}) \\ &- \sum_{i \in I_r^- \cap I^G} (q_{ti} + s_{ti}) \\ &- \sum_{i \in I_r^+ \cap I^P} (q_{ti} + s_{ti}) = 0. \end{aligned} \tag{2.1.10}$$

Like above without the volume variables, water is conserved: outgoing flows balance incoming flows, except that water cannot be stored.

Target volumes On top of the reservoir capacity (2.1.1), mid-horizon and final volumes are subject to target bounds:

$$\underline{A}_r \leq v_{\frac{\bar{t}}{2}r} \leq \overline{A}_r \qquad \forall r \in R^T \tag{2.1.11}$$

$$\underline{B}_r \leq v_{\bar{t}r} \leq \overline{B}_r \qquad \forall r \in R^T. \tag{2.1.12}$$

Power-flow characteristics We now introduce a system of constraints to express power-flow curves.

$$q_{ti} = \sum_{j \in J_{ti}} y_{tij} \Phi_{tij} \qquad \forall t \in T, \forall i \in I^C \tag{2.1.13}$$

$$p_{ti} = \sum_{j \in J_{ti}} y_{tij} \Lambda_{tij} \qquad \forall t \in T, \forall i \in I^C \tag{2.1.14}$$

$$x_{tij} \leq y_{tij} \qquad \forall t \in T, \forall i \in I^C, \forall j \in J_{ti} \tag{2.1.15}$$

$$y_{ti(j+1)} \leq x_{tij} \qquad \forall t \in T, \forall i \in I^C, \forall j \in \{1, \ldots, |J_{ti}| - 1\}. \tag{2.1.16}$$

For plants with continuous operations, constraints (2.1.13), (2.1.14), (2.1.15) and (2.1.16) describe piecewise linear power-flow curves where the breakpoints are the operational points; the

piecewise linear curves are expressed with the *incremental formulation*, see [55].

$$q_{ti} = \sum_{j \in J_{ti}} x_{tij} \Phi_{tij} \qquad \forall t \in T, \forall i \in I^D \tag{2.1.17}$$

$$p_{ti} = \sum_{j \in J_{ti}} x_{tij} \Lambda_{tij} \qquad \forall t \in T, \forall i \in I^D \tag{2.1.18}$$

$$x_{ti(j+1)} \leq x_{tij} \qquad \forall t \in T, \forall i \in I^D, \forall j \in \{1, \ldots, |J_{ti}| - 1\} \tag{2.1.19}$$

For the discrete-operating plants, flow and power can only take a set of discrete values as modeled by constraints (2.1.17), (2.1.18) and (2.1.19); the formulation is a simplified restriction of the incremental formulation in order to express variables that only take a set of discrete values.

Spillage We next introduce the set of constraints related to spillage:

$$s_{ti} \leq z_{ti} \overline{\Sigma}_{ti} \qquad \forall t \in T, \forall i \in I^G \tag{2.1.20}$$

$$z_{ti} \leq x_{ti(|J_{ti}|)} \qquad \forall t \in T, \forall i \in I^G. \tag{2.1.21}$$

Constraints (2.1.20) imply that spillage through plant i during time period t can be non-zero only if variable $z_{ti} = 1$. According to constraints (2.1.21), this means that spillage is allowed only if the last operational point is reached, *i.e.*, $x_{ti(|J_{ti}|)} = 1$.

Monotonicity, pump-turbine exclusion Next, we have monotonicity constraints alongside with simultaneous pumping and generation exclusion constraints:

$$x_{(t-1)ij} \geq x_{(t-2)ij^-} + x_{tij^+} - 1 \qquad \forall t \in T, \forall i \in I^D, \forall j \in J_{ti} \tag{2.1.22}$$

$$x_{(t-1)ij} \leq x_{(t-2)ij^-} + x_{tij^+} \qquad \forall t \in T, \forall i \in I^D, \forall j \in J_{ti} \tag{2.1.23}$$

$$x_{ti'1} + x_{ti''1} \leq 1 \qquad \forall t \in T, \forall (i', i'') \in I^R \tag{2.1.24}$$

$$x_{ti'1} + x_{(t-1)i''1} \leq 1 \qquad \forall t \in T, \forall (i', i'') \in I^R \tag{2.1.25}$$

$$x_{(t-1)i'1} + x_{ti''1} \leq 1 \qquad \forall t \in T, \forall (i', i'') \in I^R. \tag{2.1.26}$$

where $(j^-, j, j^+) \in J_{(t-2)i} \times J_{(t-1)i} \times J_{ti}$ are the indices pointing at the same operational point at three consecutive time periods $(t-2, t-1, t)$, *i.e.*, $\sum_{k=1}^{j^-} \Phi_{(t-2)ik} = \sum_{k=1}^{j} \Phi_{(t-1)ik} = \sum_{k=1}^{j^+} \Phi_{tik}$. Also, $x_{0ij}, x_{(-1)ij}$ are replaced with their initial values.

Since turbines within a plant are aggregated, constraints (2.1.22)-(2.1.23) ensure smooth operations to maintain a sustainable level of stress and wear for the turbines within a plant. Assuming that one operational point corresponds to one turbine, the monotonicity constraints on operational points reduce to 2-period minimum uptime and minimum downtime for turbine;

min-up/min-down in the general case for thermal units have been studied in [46, 63]. Constraints (2.1.24)-(2.1.26) impose that the pump and turbine of a reversible station cannot operate at the same time and that consecutive operations of pump and turbine require a one-period transition.

Ramp-up, ramp-down Finally, we present the ramp-up and ramp-down constraints:

$$(q_{ti} + s_{ti}) - (q_{(t-1)i} + s_{(t-1)i}) \leq \overline{\Upsilon}_i \qquad \forall t \in T, \forall i \in I \tag{2.1.27}$$

$$(q_{ti} + s_{ti}) - (q_{(t-1)i} + s_{(t-1)i}) \leq -\underline{\Upsilon}_i \qquad \forall t \in T, \forall i \in I \tag{2.1.28}$$

where q_{0i}, s_{0i} are replaced with their initial values.

Objective function

Our aim is to maximize the revenues given as:

$$\varrho = \max \sum_{t \in T} \sum_{i \in I} \lambda_t p_{ti} + \sum_{r \in R^T} \omega_r (v_{\bar{t}r} - \Psi_{0r} - \Pi \sum_{t \in T} \Gamma_{tr}). \tag{2.1.29}$$

The first summation gives the profit/cost due to power generation/consumption during the planning horizon (recall that $\boldsymbol{\lambda}$ is the electricity price vector), while the second summation gives the net value of the water volume remaining in the reservoir at the end of the planning horizon.

2.2 First computational tests

In this section, we present characteristics of the test set and first computational results with the complete model presented in Section 2.1.2.

2.2.1 Test set and configuration

Our test set is composed of 66 real-world instances corresponding to all combinations of 6 valleys at 11 dates. For all instances, $\bar{t} = 96$ time periods are considered. Table 2.1 summarizes instance characteristics per valley. First column gives valley name and other entries are:

- $|R|$: number of reservoirs
- $|I|$: number of plants
- $|I^R|$: number of pump-storage stations
- $|I^C|$: number of continuously-operating plants

- #bin vars: average number of binary variables per valley over the 11 dates and standard deviation in parenthesis
- #vars: average number of variables per valley over the 11 dates and standard deviation in parenthesis.

Valley	$\|R\|$	$\|I\|$	$\|I^R\|$	$\|I^C\|$	#bin vars		#vars	
v-a	1	1	0	0	136	(75)	524	(75)
v-b	2	3	1	0	953	(149)	2017	(149)
v-c	5	6	0	1	1707	(670)	3835	(670)
v-d	5	6	1	1	3296	(806)	6006	(1732)
v-e	5	9	1	0	3974	(582)	7066	(582)
v-f	10	16	2	4	4453	(406)	10343	(453)

Table 2.1: Test set characteristics per valley

Valley topographies – order and structure of reservoirs and plants along streams – remain unchanged across instances of each valley. From one date to another, data such as initial values, volume bounds, water inflows, water values, electricity prices vary; the sets of operational points also change, thus changing the number of variables as well. The variations in valley topography and date for our test set were chosen to be representative of possible configurations. Since we have a compact formulation where binary variables correspond directly to operational points, the average number of binary variables is somewhat an indicator of the combinatorial difficulty of the instances.

Tests were executed on 64-bits Intel Xeon CPU E5504 running at 2.00 GHz x8 cores with Linux and 11.7 GB of RAM. The mathematical programming modeling language AMPL Version 20121017 was used to run the MILP solver IBM ILOG CPLEX 12.4.0.0 [38]. For each instance, a time limit of 1200 seconds was imposed.

2.2.2 Test results

Table 2.2 summarizes the solution by CPLEX of the 66 instances presented in Section 2.2.1 with the complete model described in Section 2.1.2. CPLEX solution status of a MILP instance can be *integer infeasible* when the instance is proved infeasible, *feasible* when a feasible solution is found, or *intractable* when no integer solution is found within time limit, leaving feasibility of the instance an open question. First column shows solution status while second column gives the number of instances #inst whose solution falls into the aforementioned categories.

Solution status	#inst
Integer infeasible	28
Feasible	14
Intractable	24
Total result	66

Table 2.2: Number of instances according to solution status of the complete model

In addition to the numerous infeasible and intractable cases, several messages from the solver indicate numerical problems: bad average condition numbers, marginally violated constraints and rounded integer variables. Numerical problems may indicate inaccurate solutions.

2.2.3 Problem statement

When working with this real-world optimization problem, we run into specific practical difficulties, namely numerical problems and infeasibilities.

First we must ensure numerically-computed results are affected neither by the errors inherent to floating-point computations nor by noisy data from real-world instances.

Then, since we are considering a real problem that we want to model so as to be feasible, we must update our model and adjust our data so as to recover feasibility.

2.3 LP formulation of a simple model

This section is devoted to the description of a simple version of the model presented in Section 2.1.2 to work with a tractable model and avoid undecidable situations where no solution is found within satisfactory time.

Indeed, several instances remain unsolved even with the state-of-the-art solver CPLEX, as shown in Table 2.2. Actually, most standard MILP solvers – like CPLEX – are based on floating-point arithmetic; such arithmetic allows relatively fast computations but inherently incurs rounding errors. The use of an exact solver based on rational arithmetic should guarantee the exactness of our solution (see Section 2.5.2 below for more details). Unfortunately, exact solvers are slower than their approximate counterparts. To get satisfactory results within time limits for our large-scale instances with both types of solvers, we work with a simpler model.

Assuming the solution process is slow mainly because of the combinatorial difficulties in the complete model, no binary variable is considered in the simple model. Instead of considering a standard linear relaxation of the complete model, we prefer to remove all constraints involving

binary variables, including the exclusion of simultaneous pumping and generation.

Following same notation as Section 2.1.1, the mathematical formulation of the simple model:

- bounds

$$(2.1.1)$$

$$\underline{Q}_{ti} \leq q_{ti} \leq \overline{Q}_{ti} + \overline{\Sigma}_{ti} \qquad \forall t \in T, \forall i \in I \tag{2.3.1}$$

$$s_{ti} = 0 \qquad \forall t \in T, \forall i \in I \tag{2.3.2}$$

$$(2.1.4) - (2.1.5)$$

- constraints

$$(2.1.9) - (2.1.12)$$

$$p_{ti} = \frac{\sum_{j \in J_{ij}} \Lambda_{tij}}{\overline{Q}_{ti} + \overline{\Sigma}_{ti}} q_{ti} \qquad \forall t \in T, \forall i \in I \tag{2.3.3}$$

$$(2.1.27) - (2.1.28)$$

- objective

$$(2.1.29)$$

In the simple model, we artificially get rid of spillage (see (2.3.2)) and aggregate it with the water flow whose domain is extended accordingly, see (2.3.1).

For a given instance, the feasibility set of the simple model strictly includes the feasibility set of the complete model since constraints (2.1.17)-(2.1.26) involving binary variables are removed. Therefore less infeasibilities will happen with the simple model. As for the objective function, the simple model is not strictly speaking a relaxation of the complete one because the expression of the power-flow curve (2.1.18)-(2.1.14) is only approximated by constraint (2.3.3). Note that the power variables p_{ti} are dependent variables which appear only in the objective function.

All in all, the simple model enables us to get rid of intractabilities, and a few infeasibilities, to focus on numerical errors.

2.4 Numerical errors

A numerical error is observed at the end of a computation when a numerically-computed result is different from the exact solution.

For an optimization problem, a major kind of error we can run into is when feasibility is altered. Two cases may happen: either computations find a problem infeasible while exact feasible solutions exist, or computations provide a solution said to be feasible while the exact feasible set is empty. This kind of errors is denoted as *feasibility inconsistency*. In this work, given the large occurrence of infeasibilities, as shown in Table 2.2, we will focus on identifying and mitigating feasibility inconsistencies.

Rounding errors inherent to floating-point arithmetic may cause feasibility inconsistency. Feasibility inconsistency must be checked with respect to the exact result. That is why we invoke a solver based on rational arithmetic said to be *exact* as it is free from rounding errors.

Once feasibility consistency is dealt with: either an instance is known to be exactly infeasible or an instance is known to be exactly feasible. In the former case, infeasibility becomes a cause for modeling concern since we are dealing with a real-world problem that should be modeled so as to be feasible (see subsequent Section 2.9). In the latter case, numerical errors related to the feasibility (optimality) certificate of the computed feasible (optimal) solution must be checked. For linear programs, they respectively correspond to satisfaction of linear constraints and variable bounds (non-negativity of reduced costs – when minimizing). For mixed-integer programs, errors on feasibility certificate can be observed by checking integrality of integer variables on top of satisfaction of linear constraints; errors on optimality of the computed optimal solution are less trivial to track because they may occur at any node of the branch-and-bound tree. These kinds of errors are not studied in this work.

2.5 Scaling to deal with floating-point errors

As defined in the IEEE Standard for Floating-Point Arithmetic (IEEE 754), the binary double-precision format is a floating-point system in base 2 with a 52-bit precision. It is a commonly-used format for numerical computations. The set of possible representations in that format is noted $\mathbb{F}$. A number $a \in \mathbb{R}^*$ is represented by $fl(a) \in \mathbb{F}$ such that:

$$fl(a) = (-1)^{s(a)} \cdot m(a) \cdot 2^{e(a)-53} \tag{2.5.1}$$

where sign $s(a) \in \{0, 1\}$ is expressed with 1 bit, significand $m(a) \in \{2^{52}, \ldots, 2^{53} - 1\}$ is expressed with 52 bits, exponent $e(a) \in \{-1021, \ldots, 1024\}$ is expressed with 11 bits.

Floating-point representations are not exact, $\mathbb{F} \subsetneq \mathbb{R}$ and it may happen that $fl(a) \neq a$. For any real a such that $|a| \in [2^{-1022}, 2^{1023}]$, unit roundoff $u = 2^{-53} \approx 10^{-16}$ is an upper bound on the relative error $E_r(fl(a))$ of the approximation of a by $fl(a)$:

$$E_r(fl(a)) = \frac{|a - fl(a)|}{|a|} \leq 2^{-53} \approx 10^{-16}. \tag{2.5.2}$$

Also, absolute error $E_a(fl(a))$ on the floating-point representation is defined as and bounded by:

$$E_a(fl(a)) = |a - fl(a)| \lesssim u|fl(a)|. \tag{2.5.3}$$

The first rounding error happens when representing a real number in a floating-point format. Other rounding errors occur with calculations. Indeed, $\mathbb{F}$ is not closed under the basic arithmetic operations: even if the operands lie in $\mathbb{F}$, the result of such operations may need to be rounded to lie in $\mathbb{F}$. Relative rounding error on the result of a single operation is also bounded by u. The composition of several operations may propagate small rounding errors into larger errors. The study of such errors occurring in implementations of algorithms is an important subject of numerical analysis; [36] is a classic reference of this discipline.

2.5.1 Tolerances, scaling, and floating-point-based solvers

As stated in [44], solvers based on floating-point representations, such as CPLEX, introduce tolerances in order to deal with the rounding errors that are inherent to such representations.

To avoid detrimental errors on the feasibility set, linear constraints are marginally relaxed with the introduction of an absolute *feasibility tolerance* ϵ_f. For instance, for a variable x and a right-hand-side (RHS) parameter b: while we model a constraint as $x \geq b$, solvers see $x + \epsilon_f \geq fl(b)$.

Indeed provided $E_a(fl(b)) \leq \epsilon_f$ is satisfied, when only considering rounding error on the RHS term:

$$\begin{cases} E_a(fl(b)) \leq \epsilon_f \\ fl(b) \leq b + E_a(fl(b)) \end{cases} \Rightarrow b \geq fl(b) - \epsilon_f.$$

Solvers ensure that the floating-point representation of the constraint is marginally less restrictive than the exact one: it is better to slightly extend the feasible set than to arbitrarily restrict it because of rounding errors.

When providing an instance to a floating-point-based solver, we should make sure that the following condition holds for all constraints' RHS:

$$E_a(fl(b)) \lesssim u|fl(b)| \leq \epsilon_f. \tag{2.5.4}$$

If we scale down the unit of the left-hand-side (LHS) term, for example:

$$x := x/10 \text{ then } x \geq b \text{ becomes } x \geq b/10$$

and the rounding error on the RHS:

$$E_a(fl(b/10)) \approx E_a(fl(b))/10.$$

Usually, the solver's feasibility tolerance is uniquely set across constraints; $\epsilon_f = 10^{-6}$ by default for CPLEX. When only considering the effects of errors on RHS terms, orders of magnitude should be scaled down to 10^{10} at most so that condition (2.5.4) holds, which could be rewritten as:

$$|fl(b)| \leq \epsilon_f / u. \tag{2.5.5}$$

In our problem, volumes and related bounds are the numbers with the greatest magnitude. In addition, recursive constraints (2.1.9) show that volumes result from several intermediary operations, which often lead to error propagation. Volumes are therefore most likely to be subject to substantial rounding errors. For this reason, we decide to study the effect of scaling of volume-related parameters and variables.

Note that tuning directly the solver's feasibility tolerance could be an option if all constraints' RHS terms were of the same magnitude. As this is not the case, we rather rescale each variable individually so that the constraints in which they are involved are not disturbed by rounding error.

2.5.2 Computational tests

Standard floating-point solvers feature scaling routines by default, and these are kept activated for the computational tests. Moreover, we find it useful to scale variables and constraints known to be sensitive, thus rescaling the instance before providing it to the solver.

Computational protocol For volume-related parameters and variables, we test 5 scalings – denoted with letters from A to E – which correspond to increasing volume units we do not express here for proprietary reasons. Given volumes' orders of magnitude, the feasibility set, as represented by CPLEX, is expected to be altered when using scalings A and B, because condition (2.5.5) is not satisfied.

In order to identify situations where the feasibility status computed by CPLEX is incorrect, CPLEX's results are compared with results of an exact solver. Namely, we use SCIP-EX, a MILP solver based on rational arithmetic, instead of floating-point arithmetic [22]. It was run using MPS files generated through AMPL. Since no discrete variable appears in the simple model, SCIP-EX is used as an LP solver.

Feasibility consistency according to scaling Table 2.3 summarizes feasibility consistency for the solution by CPLEX of the 66 instances presented in Section 2.2.1 with the simple model described in Section 2.3 according to scaling. With regards to feasibility, the solution returned by CPLEX for problem formulated with different volume scalings may be:

- *consistentF* if solution processes with all scalings reach a feasible solution and agree with SCIP-EX's exact result
- *consistentI* if solutions with all scalings are found infeasible and agree with SCIP-EX's exact result
- *inconsistentF* if at least a solution with one scaling is found infeasible though the instance is feasible according to SCIP-EX's exact result
- *inconsistentI* if at least a solution process with one scaling reaches a feasible solution though the instance is infeasible according to SCIP-EX's exact result.

Note that the solution obtained using with a rational-based solver is left unchanged by scaling as there is no feasibility tolerance. Therefore, SCIP-EX's results do not depend on scaling.

feasibility consistency	SCIP-EX status	CPLEX status					#inst
		scaling A	scaling B	scaling C	scaling D	scaling E	
consistentF	optimal	optimal	optimal	optimal	optimal	optimal	12
consistentI	infeasible	infeasible	infeasible	infeasible	infeasible	infeasible	18
inconsistentF	optimal	infeasible	optimal	optimal	optimal	optimal	32
inconsistentI	infeasible	optimal	optimal	optimal	optimal	optimal	1
	infeasible	infeasible	optimal	optimal	optimal	optimal	1
	infeasible	infeasible	infeasible	infeasible	infeasible	optimal	2
Total Result							66

Table 2.3: Number of instances whose solutions are (in)consistent according to scaling, for the simple model

Because condition (2.5.5) with scaling A is not satisfied, CPLEX's solution is wrongfully found infeasible for the 32 *inconsistentF* instances.

Error on objective according to scaling Let us now consider the 44 (=12 *consistentF* + 32 *inconsistentF*) instances that are rightfully found feasible and solved to optimality by CPLEX for scalings B to E. For each instance, the relative objective error $\%\Delta$ is computed as the relative difference between the objective value found by both solvers. Table 2.4 summarizes statistics on relative objective error according to scaling. Maximum (max), average (avg), and standard deviation (std) of the relative objective error $\%\Delta$ are computed over the 44-instance set for each scaling.

Even though condition (2.5.5) with scaling B is not satisfied, there is hardly no feasibility inconsistency observed for the considered test set. However, the errors on the computed objective value for scaling B appear to be much greater than those obtained with scalings C to E.

	%Δ			
	scaling B	scaling C	scaling D	scaling E
max	5.7E-03	8.1E-09	1.7E-08	1.6E-08
avg	2.1E-04	1.9E-10	4.4E-10	4.1E-10
std	1.0E-03	1.2E-09	2.5E-09	2.4E-09

Table 2.4: Statistics of relative objective error according to scaling, for the simple model

This indicates that rounding errors are more likely to occur for scaling B. It is not in the immediate scope of our investigation but it affects subsequent results: exact optimal solutions for LP problems are critical to obtain bounds and valid inequalities to solve MILP problems with a branch-and-cut scheme.

Scaling and data errors To return to the 4 (=1+1+2) *inconsistentI* instances of Table 2.3, the results illustrate there is a limit for the use of scalings with CPLEX. Indeed, when a relatively large scaling is used, significant digits of handled variables and parameters are neglected as they become lower than the absolute feasibility tolerance. Therefore, the relaxation introduced by the standard feasibility tolerance for rounding errors becomes too loose and exactly infeasible instances (as seen by SCIP-EX) are seen as feasible.

Actually, in our case, as we want to formulate problems to be feasible, if a slight perturbation of the problem helps recovering feasibility, it is a hint that there are data errors. An approximate computation with inaccurate data might recover feasibility by chance, whereas an exact computation with inaccurate data cannot. However, the combination of large scalings and feasibility tolerance for an approximate computation might over-relax the problem. For this reason, scaling E, being too large, is discarded for future computations with CPLEX, while data errors are dealt with in the subsequent section.

2.6 Corrective relaxation to deal with data errors

Beside rounding errors, *data errors* are another kind of numerical errors. Data errors are observed when a numerically-computed result is different from the exact solution because of inaccurate input data. As pointed above in Section 2.5.2, even an exact computation may lead to an incorrect result if input data is inaccurate.

If input value $\hat{a}$ is read for parameter a, the absolute error on data $\hat{a}$ is:

$$E_a(\hat{a}) = |\hat{a} - a|.$$

Since only $\hat{a}$ is given, upper bounds on relative data error $\overline{E_r}$ are derived for all parameters:

- $\Psi_0, \overline{\Psi}, \underline{\Psi}, \overline{A}, \overline{B}, \underline{A}, \underline{B}, \Gamma$ regarding reservoirs
- $\overline{\Upsilon}, \underline{\Upsilon}, \overline{\Sigma}, \Phi, \Lambda$ regarding plants.

Given upper bound on relative data error $\overline{E_r}(\hat{a})$ of input value $\hat{a}$, we have:

$$E_a(\hat{a}) \leq \overline{E_a}(\hat{a}) \approx \overline{E_r}(\hat{a}) \cdot |\hat{a}|.$$

2.6.1 Corrective relaxations

To recover feasibility, we must correct data to relax the feasible region given initially. More specifically, within each constraint we correct parameters by adding or subtracting the upper bound on the error in the direction that relaxes the constraint. Assuming w.l.o.g. $a \geq 0$, $x \geq 0$, the *exact* constraint:

$$ax \geq b, \text{ where } a \in [\hat{a} - \overline{E_a}(\hat{a}), \hat{a} + \overline{E_a}(\hat{a})], b \in [\hat{b} - \overline{E_a}(\hat{b}), \hat{b} + \overline{E_a}(\hat{b})]$$

is no longer represented by the *inexact* constraint:

$$\hat{a}x \geq \hat{b}$$

but is relaxed into the following corrected constraint:

$$(\hat{a} + \overline{E_a}(\hat{a}))x \geq \hat{b} - \overline{E_a}(\hat{b}).$$

This *data correction* is carried out for all inequality constraints; equality constraints being considered as two opposite inequalities. The exact feasible region is extended because we are bound to consider the worst-case errors for all parameters.

Note that our primary goal is to deal with infeasibilities, that is why we focus only on constraints. As the feasibility region is enlarged and the objective coefficients are not adjusted accordingly, it is possible that a better objective value is found with data correction.

Note also that we propose to use scalings to deal with rounding errors (as described in Section 2.5.1) and data correction to handle data errors. It seems like we could have used scaling for both. However, using scalings would not work with an exact solver because no feasibility tolerance is used. Moreover, feasibility tolerance can handle only cumulative error on all terms of a constraint, while the proposed data correction handles each term individually.

2.6.2 Computational tests

Computational protocol We consider the simple model presented in Section 2.3. We choose to keep scaling C for volumes for all reservoirs of all instances as it was shown to be a suitable scaling.

Feasibility consistency according to data correction We sort the instances based on the following case disjunction:

- case 1 if SCIP-EX and CPLEX reach a feasible solution without data correction;
- case 2 if SCIP-EX and CPLEX solutions are still found infeasible with data correction;
- case 3 if SCIP-EX and CPLEX solutions are found infeasible without data correction, but both solvers reach a feasible solution with it;
- case 4 if SCIP-EX solution is found infeasible without data correction, but CPLEX reaches a feasible solution without it, and both solvers reach a feasible solution with data correction.

Table 2.5 summarizes feasibility consistency of the solutions for the 66 instances presented in Section 2.2.1 by CPLEX and SCIP-EX without data correction (*noDC*) and with data correction (*wDC*).

case#	*noDC*		*wDC*		#inst
	SCIP-EX	CPLEX	SCIP-EX	CPLEX	
case 1	optimal	optimal	optimal	optimal	44
case 2	infeasible	infeasible	infeasible	infeasible	11
case 3	infeasible	infeasible	optimal	optimal	9
case 4	infeasible	optimal	optimal	optimal	2
Total Result					66

Table 2.5: Number of instances whose solutions are (in)consistent according to data correction, for the simple model with scaling C

First of all, comparing SCIP-EX results without or with data correction (second and fourth columns) shows that 11 (= 9+2) instances were infeasible because of data errors and become feasible with data correction. Second, SCIP-EX and CPLEX solutions disagree without data correction for the 2 instances (case 4), whereas they now agree with it. In the end, the proposed data correction is shown to be effective as feasibility consistency between solvers is recovered.

Error on objective according to data correction Let us compare relative objective error between solvers for the instances that are rightfully found feasible and solved to optimality: the 44 feasible instances without data correction, relative to case 1, and the 55 feasible (= 44 + 9 + 2) instances with data correction, relative to cases 1, 3, and 4. Table 2.6 summarizes

statistics on relative objective error according to data correction. The relative objective error $\%\Delta$ is computed as explained previously in Section 2.5.2. For the two sets of instances that are solved to optimality, we restate the number of instances and we present the same statistics as in Table 2.4. Luckily, rounding errors are mitigated with data correction as CPLEX objective value

	$\%\Delta$	
	noDC	*wDC*
max	8.1E-09	9.7E-11
avg	1.9E-10	4.9E-12
std	1.2E-09	1.8E-11
#inst	44	55

Table 2.6: Statistics of relative objective error according to data correction, for the simple model with scaling C

is relatively more accurate.

2.7 Extensions of results to the complete model

Ideally, we would like to check that the computational results obtained by scaling and data correction for the simple linear model can be extended to the complete mixed-integer model. Unfortunately, SCIP-EX solution times are too long for the complete model.

Therefore, we do not refer to SCIP-EX's results to check feasibility consistency. We keep on using scaling C for volumes and data correction on the complete model. Table 2.7 summarizes the solution by CPLEX for the 66 instances presented in Section 2.2.1 with the complete model described in Section 2.1.2 without and with data correction. First (Second, respectively) column shows solution status *noDC* (*wDC*, respectively). Third column gives the number of instances #inst for each combination of the first and second column entries. Note that results *noDC* correspond to the ones presented in Section 2.2.2 (Table 2.2). Out of 66 instances, we are now confident that 46 (=14+11+21) are exactly feasible and 13 exactly infeasible; nothing can be said about the feasibility of the remaining 7 (=4+3) instances.

2.8 Infeasibility classification

Without numerical errors in the data and in the solution process, we expect our problem to be modeled so as to be feasible. As infeasibilities still occur, we would like to understand

noDC	*wDC*	#inst
feasible	feasible	14
infeasible		11
intractable		21
infeasible	infeasible	13
infeasible	intractable	4
intractable		3
Total Result		66

Table 2.7: Number of instances according to CPLEX solution status, without data correction *noDC* and with it *wDC*, for the complete model with scaling C

their sources and update the model accordingly. In this section, infeasibilities are first classified according to the two main specifications: discrete operations and water-management policies.

2.8.1 Approach

CPLEX's *conflict refiner* routine [38] offers to identify a "set of mutually contradictory constraints". It is not guaranteed to be minimal, that is to say it may not be an Irreducible Infeasible Subset (as defined in [32]). Even so we cannot systematically call the routine because it is very time consuming given the size of our instances. In addition, this routine considers all constraints individually while we are interested on insights by groups of constraints. Therefore, we mimic the conflict refiner routine by manually relaxing sets of constraints corresponding to the two specifications of the complete model – namely discrete operations and water-management policies – and by checking feasibility status. We still use it occasionally when individual refinement is needed.

According to the combination of relaxations, we consider 4 models:

- the simple model (from Section 2.3) without target volumes $\tilde{s}$
- the simple model s: $\mathcal{F}(s) \subset \mathcal{F}(\tilde{s})$
- the complete model (from Section 2.1.2) without target volumes $\tilde{c}$: $\mathcal{F}(\tilde{c}) \subset \mathcal{F}(\tilde{s})$
- the complete model c: $\mathcal{F}(c) \subset (\mathcal{F}(\tilde{c}) \cap \mathcal{F}(s))$

where $\mathcal{F}(m)$ is the feasible set of model m.

Indeed, the specification of discrete operations was relaxed so as to obtain the simple model s from the complete model c. We now also consider relaxations obtained by removing target

volume bounds which are a hard expression of water-management policies. Ideally we wish to satisfy target volumes but we relax them to detect if they are sources of infeasibility and therefore obtain models $\tilde{s}$ and $\tilde{c}$. Target volume bounds will be referred to as TV.

2.8.2 Computational results and interpretation

In Table 2.8, we refer to results obtained considering the full test set with the complete model, scaling C, and data correction. Namely, first column shows CPLEX solution status, second column shows (when applicable) infeasibility classification (as defined in details below), and third column gives, for each combination of status/class, the number of relative instances #inst. Note that results of the first column are the ones obtained for column *wDC* of Table 2.7.

solution status	infeasibility classification	#inst
feasible	–	46
intractable	–	7
infeasible	data inconsistent	5
	unattainable TV	6
	impossible discrete operations	0
	unattainable TV and impossible discrete operations	0
	incompatible TV and discrete operations	2
Total Result		66

Table 2.8: Number of instances according to solution status and infeasibility classification, for the complete model with scaling C and data correction

Infeasibilities for the complete model, *i.e.*, $\mathcal{F}(c) = \emptyset$, can be classified according to the feasibility status of the considered relaxations:

- *data inconsistent* when $\mathcal{F}(\tilde{s}) = \emptyset$: it may happen, for example, when initial volume is out of bounds $\Psi_r 0 > \overline{\Psi}_r$ for some reservoir r. There exist several other cases we identified thanks to CPLEX's conflict refiner; often interpretation of an operator is required to recover feasibility. For this reason we decide to discard this kind of infeasibilities within the scope of this work.

- *unattainable TV* when $\mathcal{F}(s) = \emptyset$ and $\mathcal{F}(\tilde{c}) \neq \emptyset$: it happens when the target volumes are unattainable even when discrete plant operational domains are relaxed. This kind of infeasibilities will be dealt with in Section 2.9.

- *impossible discrete operations* when $\mathcal{F}(\tilde{c}) = \emptyset$ and $\mathcal{F}(s) \neq \emptyset$: such infeasibilities happen when the sets of discrete operational points together with ramp-up/down, monotonicity and exclusion constraints forbid values of water flow required to remain within reservoir capacities. Because such requirements are critical for the operations of plants, we must rely on the know-how of an operator to find a tradeoff. For the same reasons as data inconsistencies, we discard this kind of infeasibilities.
- *unattainable TV and impossible discrete operations* when $\mathcal{F}(s) = \emptyset$ and $\mathcal{F}(\tilde{c}) = \emptyset$: when the two previous kinds of infeasibilities concomitantly happen. As we discard *impossible discrete operations*, we also discard this kind of infeasibilities.
- *incompatible TV and discrete operations* when $\mathcal{F}(s) \neq \emptyset, \mathcal{F}(\tilde{c}) \neq \emptyset$: it happens when the target volumes are incompatible with complete discrete plant operational domains, though target volumes are attainable with relaxed plant operational domains, and discrete operations are feasible without target volumes. Like the cases where target volumes are unattainable, this kind of infeasibilities will be dealt with in Section 2.9.

Within the tested set, there was no occurrence of *impossible discrete operations* and *unattainable TV and impossible discrete operations*. We are left with cases relative to *unattainable TV* and *incompatible TV and discrete operations*. We must therefore deal with too tight target volumes, which are responsible for the infeasibilities of at least 8 (=6+2) instances of our tests.

For the 7 instances whose solutions are intractable for the complete model, solving the relaxed models reached feasible solutions: $\mathcal{F}(s) \neq \emptyset, \mathcal{F}(\tilde{c}) \neq \emptyset$. They can either be feasible or feature an incompatibility between target volumes and discrete operations.

2.9 Target volume reformulation for feasibility recovery

Because mid-term planning policies cannot always be properly reflected through the valuation of water or to satisfy punctual requirements for reservoir operations, daily water level trajectories are controlled with the introduction of target volumes.

Introducing them as hard constraints often results in infeasibility as seen above in Section 2.8 and also as mentioned in, for example, [58].

To recover feasibility, we propose a 2-stage method that reformulates water-management policies.

2.9.1 Minimization of target volume deviations

In the first stage we complete and marginally modify the model formulated in Section 2.1.2 to recover feasibility. Slack variables are introduced, namely:

- $\overline{\alpha}_r, \overline{\beta}_r, \underline{\alpha}_r, \underline{\beta}_r$ [m^3]: deviations to mid-horizon maximum target volume, final maximum target volume, mid-horizon minimum target volume and final minimum target volume of reservoir r ($\forall r \in R^T$).

The bounds on these new variables are defined as follows ($\forall r \in R^T$):

$$
\begin{aligned}
0 \leq \overline{\alpha}_r &\leq \max(\overline{A}_r - \overline{\Psi}_r, 0) \\
0 \leq \overline{\beta}_r &\leq \max(\overline{B}_r - \overline{\Psi}_r, 0) \\
0 \leq \underline{\alpha}_r &\leq \max(\underline{\Psi}_r - \underline{A}_r, 0) \\
0 \leq \underline{\beta}_r &\leq \max(\underline{\Psi}_r - \underline{B}_r, 0)
\end{aligned}
$$

so that volumes deviate from target bounds while satisfying reservoir capacity bounds.

Relaxing constraints (2.1.11)-(2.1.12) as follows is expected to guarantee feasibility:

$$\underline{A}_r - \underline{\alpha}_r \leq v_{\overline{t}/2,r} \leq \overline{A}_r + \overline{\alpha}_r \qquad \forall r \in R^T \tag{2.9.1}$$

$$\underline{B}_r - \underline{\beta}_r \leq v_{\overline{t},r} \leq \overline{B}_r + \overline{\beta}_r \qquad \forall r \in R^T. \tag{2.9.2}$$

Yet we want to relax the constraints only when necessary. For instance, if target volume bounds are attainable, no slack should be introduced. To deviate as little as possible from the initial target volume bounds, we compute the minimal amount of slacks that is necessary and sufficient to recover feasibility. To do this, objective function (2.1.29) is replaced by another objective, *i.e.*, the minimization of the sum of deviations, so as to recover feasibility:

$$\varepsilon = \min \sum_{r \in R^T} (\overline{\alpha}_r + \overline{\beta}_r + \underline{\alpha}_r + \underline{\beta}_r). \tag{2.9.3}$$

The first stage terminates with a value ε. ε can be considered as the 1-norm of the deviation vector $(\overline{\alpha}_r - \underline{\alpha}_r, \overline{\beta}_r - \underline{\beta}_r)_{r \in R^T}$.

2.9.2 Optimization of the original problem within deviations

In the second stage, we still consider the slack variables and related constraints (2.9.1)-(2.9.2) but we limit the sum of deviations as follows:

$$\sum_{r \in R^T} (\overline{\alpha}_r + \overline{\beta}_r + \underline{\alpha}_r + \underline{\beta}_r) \leq \varepsilon \tag{2.9.4}$$

where ε corresponds to the result of the first stage. Either the initial feasible set was empty and it is now enlarged within the computed deviation to contain at least one feasible solution, or the initial feasible set was already non-empty and it remains unchanged when the deviation is zero.

We can then optimize the original objective function (2.1.29) over a feasible set now guaranteed to be non-empty.

Note that the proposed method could be interpreted as a lexicographic method to solve a bi-objective MILP where the first objective is minimizing the deviation with respect to the target volumes (2.9.3) and the second objective is the original objective function (2.1.29). For details on lexicographic methods for multi-objective optimization problems, the reader is referred to [65], for example.

2.9.3 Computational tests

Feasibility recovery Table 2.9 expands Table 2.8 above to present results of the feasibility recovery stage. The first (respectively second) column shows the solution status of the original problem *orig_opt* (respectively the feasibility recovery stage *feas_opt*). Column 3 provides infeasibility classification and column 4 shows if the sum of deviation ε is equal to 0 or not. Finally, column 5 gives the number of relative instances #inst corresponding to each combination of the previous columns.

solution status		infeasibility classification	ε	#inst
orig_opt	feas_opt			
feasible	feasible	–	0	46
intractable	feasible	-	0	1
			>0	6
infeasible	infeasible	data inconsistent	–	5
	feasible	unattainable TV	>0	6
	feasible	incompatible TV and discrete operations	>0	2
Total Result				66

Table 2.9: Number of instances according to solution status, infeasibility classification and feasibility recovery results, for the complete model with scaling C and data correction

The 46 instances that were originally feasible remain unchanged as the computed deviation is zero. Feasibility recovery stage converges to 0 for 1 instance originally intractable, thus proving original feasibility. As for the other 6 originally intractable instances, no zero integer solution was found in the recovery stage within time limit, thus not proving original feasibility, but the lower bound at termination was still zero, thus not proving original infeasibility either; at least the non-zero integer solution found in the first stage allows to consider a non-empty restricted feasible set for the second stage. The 5 instances that were originally data inconsistent remain infeasible. For the 8 (=6+2) instances that were originally infeasible because of target volumes,

a non-zero deviation is computed to recover feasibility. All in all, feasibility is recovered for 61 instances out of 66.

Revenues optimization For those 61 instances, Table 2.10 summarizes solution statistics of the second stage. Table entries are :

- valley: valley name
- #inst: number of feasible instances
- $\|\varrho\|$: average normalized revenues and standard deviation
- $\|\varepsilon\|$: average normalized sum of deviations used for constraints (2.9.4) and standard deviation
- %gap: average relative gap (between best integer solution and upper bound) and standard deviation
- #nodes: average number of branch-and-bound nodes and standard deviation in thousands
- time: average wall-clock solution time and standard deviation in seconds.

valley	#inst	$\|\varrho\|$		$\|\varepsilon\|$		%gap		#nodes		time (s)	
		avg	std	avg	std	avg	std	avg	std	avg	std
v-a	7	22%	39%	14%	38%	0.0%	0.0%	0	0	0	0
v-b	11	43%	22%	0%	0%	0.1%	0.1%	2,359	2,953	412	512
v-c	11	33%	37%	23%	33%	490%	1,085%	2,216	1,725	771	543
v-d	10	56%	21%	10%	32%	0.1%	0.0%	565	1,334	250	501
v-e	11	44%	24%	13%	30%	1.3%	2.4%	1,054	239	1,201	0
v-f	11	54%	23%	0%	0%	0.1%	0.1%	464	470	630	558

Table 2.10: Solution statistics per valley of second stage for the complete model with scaling C and data correction

For proprietary reasons, we do not show explicitly values of revenues and deviations. For indication, we show values normalized per valley, *i.e.*, assuming $\overline{\varrho} > \underline{\varrho}$

$$\|\varrho\| = \frac{\varrho - \underline{\varrho}}{\overline{\varrho} - \underline{\varrho}}$$

and $\|\varrho\| = 0$ if $\overline{\varrho} = \underline{\varrho}$, where ϱ is the revenues given by the best solution found for a specific couple (valley, date) while $\overline{\varrho}$ and $\underline{\varrho}$ are the maximum and minimum revenues found as best solution value for the same valley across dates. The normalized value $\|\varepsilon\|$ for deviations was computed in similar fashion.

2.10 Conclusion

In this chapter, we dealt with feasibility issues of a real-world hydropower unit commitment. The problem was formulated as a mixed-integer linear programming model. Given real-world instances, we reformulated the model to make the problem feasible. Compared with a standard HUC problem, the model featured two additional specifications: discrete operational points of the power-flow curve and mid-horizon and final strict targets for reservoir levels. In practice, solving this HUC problem often incurred infeasibility situations. We used a step-by-step approach to systematically exhibit and cope with one source of infeasibility at a time, namely numerical errors and model infeasibilities.

Numerical errors were analyzed by resorting to an exact solver; rounding errors were dealt with proper scaling while data errors were mitigated with corrective relaxations. We derived a preliminary processing on the model and instances used by solvers to eliminate the detrimental effects of numerical errors. We showed that the remaining infeasibilities were originated by conflicting model specifications between target volumes and discrete operational points.

We proposed a 2-stage method to lift such infeasibilities. In the first stage, a minimal deviation from target volumes is estimated to make the problem feasible. In the second stage, the original HUC problem is solved with a possible deviation from the target volumes as defined in the first stage.

Computational results showed the effectiveness of the data and model pre-processing – rescaling, correction and 2-stage method with feasibility recovery – on an original test set of 66 real-world instances that demonstrated a high occurrence of infeasibilities.

In a wider perspective, the mathematical optimization issues we have grappled with, and the many remaining, can be considered from a system engineering viewpoint, as suggested in [66]. Indeed, the mathematical model together with the implementation of a solution method to solve data instances is a decision-support system for short-term hydropower scheduling, Then, two relevant systems engineering questions are product verification and product validation. Basic definitions of these notions are given in [40] as: “Simply put, the Product Verification Process answers the critical question – Was the end product realized right? The Product Validation Process addresses the equally critical question – Was the right end product realized?”

The first computational tests (Section 2.2) can be regarded as a verification of the initial product. Its results show that the initial product did not provide correct solutions within a limited amount of time. Looking further, the analysis and scaling we carried out on floating-point errors for the feasibility tolerance of volume-related constraints could be extended and partly automated to deal with other constraints and integer feasibility tolerance. The data corrections

and the two-stage method for target volumes were the next steps to recover feasibility to comply with verification. The rationale behind these treatments was to introduce necessary and sufficient slacks from the initial product to recover feasibility. Several other paths could have been followed (see [18]).

A further step should be to validate the initial product along with the amendments we crafted. To do so, insightful directives are presented in [12], especially with a view to better collaborate with end-users/clients. One way to validate such a system would be to integrate it in a larger simulation test bed. For example, with a month-long data set of guide curves for reservoir, we could run the suggested optimization model consecutively to appraise the terminal deviation, if any. Also, our belief is that, despite the simplicity of genericity, a one-model-fits-all approach is too narrow to represent the various valleys under various conditions of operations.

Returning to product verification, providing correct solutions within a limited amount of time requires further investigation. Analyzing the performance of the current approach – direct MILP solver invocation – and *diagnosing* its difficulties could be done following the methodology suggested in [43]. There are ongoing works to design other solution approaches for example by resorting to path-reformulation (see [74]). Along the same lines, we have found promising to investigate the underlying analogy between commitment decisions and items of the well-known knapsack problem [41] in order to exploit the structural properties of the problem.

Chapter 3

A multiplicative weights update heuristic for mixed-integer non-linear programming – an application to a hydropower unit commitment problem

In this chapter, we discuss a new heuristic for Mixed-Integer Non-Linear programming based on the adaptation of the well-known Multiplicative Weights Update (MWU) method (see [7]). The MWU was applied in the past to derive approximation algorithms for Linear Programming (LP) [59] and Semi-Definite Programming (SDP) [6]. Although we do not derive approximation algorithms, we demonstrate the applicability of the MWU as a heuristic for MINLP. To adapt the MWU to MINLP, we have devised an *ad hoc* reformulation – that we have denoted *pointwise* – which could also be used in a framework different from the MWU. Unlike most heuristics, we show that the MWU still retains a relative approximation guarantee in the MINLP setting. We present a further adaptation of our general method to a non-linear continuous HUC problem. We find that the MWU is competitive with respect to the classical Multi-Start (MS) algorithm. Applications to another NLP problem and to a MINLP problem can be found in [53].

This chapter is organized as follows. First, we give the generic description of the methodology:

- in Section 3.1, we remind the original MWU method and its theoretical guarantee in order to motivate its derivation into a (meta)heuristic;
- in Section 3.2, we introduce the notion of pointwise reformulations;

- in Section 3.3, we provide guidelines for the adaptation of the MWU method for pointwise reformulated MINLP.

Then, we present the application for an NLP HUC problem:

- in Section 3.4, we introduce its mathematical model, derive an adapted pointwise reformulation and describe an adapted variant of the MWU algorithm;
- in Section 3.5, we present computational results.

3.1 The MWU framework

3.1.1 Original MWU framework

The (original) Multiplicative Weights Update (MWU) algorithm (see [7]) is a stochastic algorithm with a relative performance guarantee on a weighted average of the errors. We rephrase here the didactic intrepretation described in [7]. Over T rounds, an investor has to choose which advisor to follow among a set of q advisors. His goal is to maximize his expected cumulative payoff. At a round $t \leq T$, the investor for example chooses to follow advisor i without knowing beforehand the payoff associated with that decision, then the market unfolds and the payoffs associated with all the decisions are revealed. Ideally, the investor wishes to "guess" who is the best advisor, that is to say the advisor whose decisions have yielded the largest cumulative payoff. Unfortunately, it cannot be done without hindsight. The intuition is to use the past performance of advisors to predict their future performance. The cumulative performance of advisors is graded by weights ω that denote the trust granted to them, those weights are updated at each round after payoffs associated with decisions are revealed. Instead of choosing completely at random and instead of choosing only the best advisor so far, the intuition is to choose at random according to a distribution that depends on the weights.

Algorithm 1 gives the pseudocode of the original MWU.

Given a positive constant $\eta \leq \frac{1}{2}$ and a number of iterations $T \in \mathbb{N}$:

Before the first round, all advisors are trusted equally $\forall i \leq q, \omega_i^1 = 1$. At each round $t \leq T$, the MWU performs the following tasks:

- it samples which advisor to follow according to a probability distribution $(p_i^t)_{i \leq q}$ proportional to weights $(\omega_i^t)_{i \leq q}$;
- it observes the scaled cost/gain $\psi_i^t \in [-1, 1]$ based on the decision of each advisor $i \leq q$;
- it updates weights $(\omega_i^{t+1})_{i \leq q}$ based on the advisors' performance, in order the compute the new distribution $(p_i^{t+1})_{i \leq q}$.

Algorithm 1 Multiplicative Weights Update algorithm

1: initiate: $\forall i \leq q, \omega_i^1 \leftarrow 1$
2: **for** $t \in \{1, \ldots, T\}$ **do**
3: sample i in $\{1, \ldots, q\}$ according to $(p_j^t)_{j \leq q}$, where $p_j^t = \frac{\omega_j^t}{\sum_{\ell \leq q} \omega_\ell^t}, \forall j \leq q$
4: observe costs incurred $(\psi_i^t)_{i \leq q}$
5: update weights according to costs $\forall i \leq q, \omega_i^{t+1} \leftarrow \omega_i^t(1 - \eta\psi_i^t)$
6: **end for**

Here we have used costs instead of payoffs. Costs are scaled $\psi^t \in [-1, 1]^q$; positive costs reflect bad outcomes. The objective of the investor is to follow the advisor whose decisions incur the minimum overall cost. Note that the realization of the costs ψ can be given by an oracle.

The idea of the algorithm – and its name – is based on Step 5 of Algorithm 1.

The MWU yields an approximation guarantee on the weighted average costs/gains relative to the performance of the best advisor.

3.1.2 The MWU approximation guarantee

We recall here the precise statement of the MWU approximation guarantee.

Given $q, \eta, T, \omega, \psi, p$ as above, it can be shown that [7]:

$$E_{\mathsf{MWU}} \triangleq \sum_{t \leq T} \sum_i \psi_i^t \, p_i^t \quad \leq \quad \min_{i \leq q} \left(\sum_{t \leq T} \psi_i^t + \eta \sum_{t \leq T} |\psi_i^t| \right) + \frac{\ln q}{\eta}. \tag{3.1.1}$$

We remark that the guarantee (3.1.1) is actually an immediate consequence of the more general bound:

$$\forall i \leq q \qquad E_{\mathsf{MWU}} \quad \leq \quad \sum_{t \leq T} \psi_i^t + \eta \sum_{t \leq T} |\psi_i^t| + \frac{\ln q}{\eta}.$$

If the decision maker picks the advisor to follow at each round according to $(p^t)_{t \leq T}$, the expected cumulative cost E_{MWU} is bounded by a piecewise function of the accumulated cost of any advisor.

Also, different weight update rules yield slightly different bounds. For example, when the exponential update rule is used in Step 5 of Algorithm 1:

$$\omega_i^{t+1} \leftarrow \omega_i^t e^{-\eta\psi_i^t},$$

the derived bound is:

$$\forall i \leq q \qquad E_{\mathsf{MWU}} \quad \leq \quad \sum_{t \leq T} \psi_i^t + \eta \sum_{t \leq T} (\psi_i^t)^2 \, p_i^t + \frac{\ln q}{\eta},$$

where $(\psi^t)^2$ is the component-wise square of the vector ψ^t. Taking the minimum over $i \leq q$ on both sides of the inequality yields the exponential MWU bound corresponding to the inequality (3.1.1).

3.1.3 The intuition of the MWU for MINLP

The original MWU algorithm (Algorithm 1) samples some values from an iteratively updated distribution in order to optimize a given loss criterion. We adapt this framework to MINLP by introducing auxiliary problems, which can be solved more efficiently than the original formulation. Hence they can be solved repeatedly at a relatively low computational cost. The auxiliary problems are obtained through the pointwise reformulation which refers to a family of parametrized approximations where terms of the original formulation containing decision variables are replaced with parameters denoted θ. The adaptation of the MWU then relies on two analogies: first, advisors predict the values of parameters θ; secondly, costs associated with those parameters are computed with respect to the optimality of the solution of the auxiliary pointwise problem.

The sketch of the MWU for MINLP (see Algorithm 2) is very similar to the original one (see Algorithm 1). Given positive constants $\eta \leq \frac{1}{2}$ and a number of iterations $T \in \mathbb{N}$, the MWU maintains a list of weights $\omega = (\omega_i \mid i \leq) \in [0,1]^q$ which are used to randomly update the values of a vector $\theta \in \Theta \subseteq \mathbb{R}^q$ of parameters incurring an associated vector $\psi \in [-1,1]^q$ of costs; the costs are then used to update the weights ω according to

$$\forall i \leq q \quad \omega_i \leftarrow \omega_i(1 - \eta\psi_i). \tag{3.1.2}$$

Algorithm 2 Sketch of MWU algorithm for MINLP

1: **while** termination condition is not met **do**
2: sample θ from distribution depending on ω
3: solve pointwise auxiliary problem parametrized by θ
4: compute the cost vector ψ associated to θ
5: update ω using ψ as in (3.1.2)
6: **end while**

The idea is that the weights ω iteratively adapt θ to being a good solution of an optimization problem which aims to minimize the costs ψ. As the i-th cost ψ_i gets smaller (and perhaps negative, becoming a gain), the associated weight ω_i increases (because of update (3.1.2)), which should hopefully yield an even smaller cost at the next iteration.

It takes a nontrivial amount of work to adapt the MWU to the NLP and MINLP settings. We provide guidelines to relate θ to a given (MI)NLP in Section 3.2 and to compute ψ in Section 3.3 for the general case. We suggest specific implementations for the HUC application in Sections 3.4.2 and 3.4.3.

3.1.4 The MWU as a metaheuristic for MINLP

In [70], the following definition of metaheuristic is given:

> A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms.

For example, the Multi-Start (MS) metaheuristic is described in Algorithm 3.

Algorithm 3 Multi-Start metaheuristic

1: **while** termination condition is not met **do**
2: sample a starting point x'
3: perform a local descent from x', yielding $\tilde{x}$
4: if $\tilde{x}$ improves the best optimum x^* so far, update x^* with $\tilde{x}$
5: **end while**

It is evident that, given any optimization problem, a nontrivial amount of work is necessary to determine the best distribution to sample from, what kind of local descent algorithm to employ, and which termination condition is appropriate. For heuristics, the answer mainly comes from computational results (see Section 3.5 for comparative tests). To have a fair performance benchmark between MS and MWU without over-tuning, we arbitrarily decide on simple settings to run our experiments. Specific variations of the settings allow us to draw insights on the empirical behavior of the two heuristics for the HUC problem we deal with.

Also, we feel we should give a word of caution about the reasons which lead us to investigate yet another metaheuristic, given their abundance. Moreover, the word "novel" associated with "metaheuristic" has already attracted some well argued criticism [70]. Our main motivation for singling out the MWU is that, unlike the vast majority of metaheuristics, it comes with a relative approximation guarantee on the cumulative (over all iterations $t \leq T$) mean costs, which turn out to be bounded above by a piecewise linear function of the cumulative *lowest* cost over $i \leq q$. Although, in general, such a guarantee is rather weak, it is surprising that it should exist at all, considering that it essentially only depends on the weights update in (3.1.2). For cases when an upper bound can be provided for the lowest cost over $i \leq q$, the MWU readily turns into a proper approximation algorithm, as shown in [7]. Although we do not derive such approximations in

this work, we mean to study this issue in further works, and hope that this work will spark interest in the matter.

We remark that our algorithmic approach combines a metaheuristic framework and mathematical programming - to derive and solve pointwise reformulations. Therefore the MWU heuristic belongs to the category of *matheuristics* [51]. Several other hybrid algorithms are presented in the survey [61].

3.2 Pointwise reformulations

A Mixed-Integer Nonlinear Programming (MINLP) problem is usually cast in the general form:

$$\left.\begin{array}{rrcl} \min\limits_{x\in\mathbb{R}^n} & f(x) & & \\ \forall \ell \leq m & g_\ell(x) & \leq & 0 \\ \forall j \in Z & x_j & \in & \mathbb{Z}, \end{array}\right\} [P] \tag{3.2.1}$$

where $Z \subseteq \{1, \ldots, n\}$ is given. If $Z = \emptyset$, the problem is called a Nonlinear Programming (NLP) problem.

3.2.1 Concept and definition

We show how to relate the parameters θ sampled in the MWU framework to a general MINLP formulation P as in problem (3.2.1). Broadly speaking, we reformulate P by replacing r *problematic* terms (*e.g.*, the non-convex terms) with simpler terms parametrized by the θ parameters. This yields a simplified formulation R in the original decision variables x, which varies in function of θ. We shall then iteratively solve R with θ fixed to values determined by the MWU framework; each iterative solution of R being used to observe costs ψ.

The adaptation of the algorithm – sampling θ and *observing* ψ – is addressed in further details in Section 3.3. Definitions and properties related to the pointwise reformulation are introduced in the current section.

Notationally, for a mathematical programming formulation P, we write $\mathsf{val}(P)$ to denote the objective function value of a global optimum of P, and $\mathsf{feas}(P)$ to denote the feasible set of P.

Definition 3.2.1. *Given a MINLP P as in problem* (3.2.1)*, a pointwise reformulation $R^\theta = \underset{\mathsf{t}\leftarrow\mathsf{t}'(\theta)}{\mathsf{ptw}}(P)$ is a family of MINLP formulations, parametrized by $\theta = (\theta_s \mid s \leq r)$, which are obtained by replacing given occurrences $\mathsf{t}_1, \ldots, \mathsf{t}_r$ of terms appearing in P by corresponding parametrized terms $\mathsf{t}'_s(\theta_s)$ (for $s \leq r$).* □

For every replaced term t_s (for $s \leq r$) in Definition 3.2.1, let D_s be the range of $\mathsf{t}_s(x)$, where the term is interpreted as a function of the decision variables x of P ranging in the respective domains. For every replacement term $\mathsf{t'}_s$ (for $s \leq r$), let $D'_s(\theta_s)$ be the range of $[\mathsf{t'}_s(\theta_s)](x)$ when the term is interpreted as a function of the decision variables x of P ranging in the respective domains. For $s \leq r$, let Θ_s be the range of the corresponding parameter θ_s, and let $\Theta = (\Theta_s \mid s \leq r)$.

Given a parameter vector $\theta \in \Theta$ and a function ϕ to reformulate, we denote by ϕ^θ the function obtained by replacing the terms t by the terms $\mathsf{t'}(\theta)$. Thus, for example, the objective and constraints of R are denoted as f^θ, g^θ, respectively.

We can therefore write a pointwise reformulation R^θ of P as follows:

$$\left.\begin{array}{lrcl} \min\limits_{x \in \mathbb{R}^{n'}} & f^\theta(x) & & \\ \forall \ell \leq m & g^\theta_\ell(x) & \leq & 0 \\ \forall j \in Z' & x_j & \in & \mathbb{Z}, \end{array}\right\} [R^\theta] \tag{3.2.2}$$

where $Z' \subseteq \{1, \ldots, n'\}$.

Note that problem (3.2.2) is actually a family of formulations, parametrized by θ. Note also that, whereas the number of variables n' may be different from n, on account of replacing variable terms with parameter symbols, we enforce the same number of constraints m in both P and R^θ: replacing terms with parameters may yield trivial constraints, which we stipulate to be formally part of problem (3.2.2) for technical reasons. In practice, when solving pointwise reformulations, trivially satisfied constraints may be dropped, of course.

3.2.2 Properties

Definition 3.2.2. *Given a MINLP P and $R^\theta = \underset{\mathsf{t} \leftarrow \mathsf{t'}(\theta)}{\mathsf{ptw}}(P)$, both defined on a vector x of decision variables in $\mathbb{R}^n$:*

(a) R^θ is spanning if, for any $x \in \mathbb{R}^n$, there are values of θ such that evaluating the functions of P and of R^θ at x yields the same results – more precisely, $\exists \overline{\theta} \in \Theta_s$ such that

$$\forall s \leq r \qquad D_s \subset \bigcup_{\theta_s \in \Theta_s} D'(\theta_s) \quad \wedge \quad [\mathsf{t'}_s(\overline{\theta_s})](x) = \mathsf{t}_s(x);$$

(b) R^θ is exact if, for each globally optimal solution x^* of P, there is at least one vector $\theta' \in \Theta$ such that x^* is also an optimal solution of $R^{\theta'}$;

(c) R^θ is efficient if there is a polynomial-time algorithm for approximately solving R^θ (for $\theta \in \Theta$) to within a given $\varepsilon > 0$ approximation factor. □

Note that the exactness property of a pointwise reformulation is only meaningful for feasible problems. We stipulate that any pointwise reformulation of a class of infeasible instances is exact, by definition.

More informally, we say that a pointwise reformulation is *good* if there is an established, practically efficient technology for solving R^θ either optimally or approximately. For example, if R^θ turns out to be a Linear Program (LP) or convex NLP (cNLP), then R^θ is efficient; if it turns out to be a Mixed Integer Linear Program (MILP), R^θ it is good. Obviously, every efficient pointwise reformulation is also good.

Example 3.2.3. *Consider the following formulation P:*

$$\min x^2(1-y)+y \tag{3.2.3}$$
$$x+2y \geq 2 \tag{3.2.4}$$
$$y \in \{0,1\}. \tag{3.2.5}$$

If we set $y=0$, constraint (3.2.4) *and the objective function direction force $x=2$, whereas if $y=1$ we can let $x=0$; therefore the global optimum is $(x^*,y^*)=(0,1)$. We replace the term x^2 in the objective function by the term consisting of the scalar parameter θ, obtaining a pointwise reformulation R^θ:*

$$\begin{aligned} \min (1-\theta)y+\theta & \\ x+2y &\geq 2 \\ y &\in \{0,1\}. \end{aligned}$$

It is easy to see that R^θ is spanning whenever $\theta \in \mathbb{R}_+$. If we set $y=0$ we obtain $x \geq 2$, whereas $y=1$ yields no constraints on x. The objective function value if $y=0$ is θ; $y=1$ yields $1-\theta+\theta=1$. So for $\theta<1$ the set of global optima is $[2,\infty)\times\{0\}$; $\theta>1$ yields the global optimal set $\mathbb{R}_+\times\{1\}$, and if $\theta=1$ every feasible solution is optimal, with optimal objective function value equal to 1. Hence $(x,y)=(x^*,y^*)=(0,1)$ yields an optimum as long as $\theta \geq 1$, which means that this pointwise reformulation is exact. This pointwise reformulation is not efficient, since it is a MILP, but it is good. □

Lemma 3.2.4. *Given P and a spanning reformulation $R^\theta = \underset{\mathsf{t}\leftarrow\mathsf{t}'}{\mathsf{ptw}}(P)$, we have:*

$$\mathsf{feas}(P) \subseteq \bigcup_{\theta\in\Theta} \mathsf{feas}(R^\theta). \tag{3.2.6}$$

Proof. Let $x' \in \mathsf{feas}(P)$. Since R^θ is spanning, there is $\xi \in \Theta$ such that $\mathsf{t}_s(x') = [\mathsf{t}_s(\xi_s)](x')$ for each $s \leq r$, which implies $g_\ell^\xi(x^*) = g_\ell(x^*)$ for all $\ell \leq m$. Since $x' \in \mathsf{feas}(P)$, $g_\ell(x') \leq 0$

for all $\ell \leq m$, hence x' is also feasible in R^ξ. Since $\mathsf{feas}(R^\xi)$ is a subset of the right hand side of (3.2.6) for each possible $\xi \in \Theta$, the result follows. □

The following example shows that the inclusion (3.2.6) cannot be tightened to an equality.

Example 3.2.5. *Consider the pure feasibility NLP formulation F:*

$$x \geq \frac{1}{2} \tag{3.2.7}$$
$$x^2 = x, \tag{3.2.8}$$

where x is a continuous decision variable. Constraint (3.2.8) *is equivalent to $x \in \{0, 1\}$, so constraint* (3.2.7) *forces $x = 1$, hence $\mathsf{feas}(F) = \{1\}$. We rewrite $x^2 = x\,x$ and replace the first occurrence of x by θ, which yields the pointwise reformulation $\underset{\mathsf{t}\leftarrow\mathsf{t'}}{\mathsf{ptw}}(F) = R^\theta$:*

$$x \geq \frac{1}{2} \tag{3.2.9}$$
$$\theta\, x = x. \tag{3.2.10}$$

Any value of $\theta \neq 1$ requires $x = 0$ in order for constraint (3.2.10) *to hold, but $x = 0$ is infeasible by constraing* (3.2.9)*, i.e., $\mathsf{feas}(R^{\theta\neq 1}) = \emptyset$. Setting $\theta = 1$ yields $\mathsf{feas}(R^1) = [\frac{1}{2}, \infty)$. In particular, we have*

$$\{1\} = \mathsf{feas}(F) \subsetneq \bigcup_{\theta\in\mathbb{R}} \mathsf{feas}(R^\theta) = \mathsf{feas}(R^1) = [\tfrac{1}{2}, \infty). \tag{3.2.11}$$

Since F and R^θ have no objective function, every feasible solution is optimal by definition. Verifying exactness reduces to checking that, for every feasible solution of F, there are values of θ such that the same solution is feasible in R^θ, which is established by (3.2.11)*. So this reformulation is exact. Since F is a non-convex NLP but R^θ is an LP, this reformulation is efficient.* □

A relaxation of a MP formulation Q provides a guaranteed bound (in the optimization direction) at every feasible point of Q. Since relaxations must be efficiently solvable, and since one usually looks for a bound to the *optimal* objective function value of Q, rather than to any objective function value achieved by a feasible point in Q, it makes sense to generalize a relaxation so it is about optimal rather than feasible points. A *bounding reformulation* of Q as a reformulation which, when solved to optimality, provides a bound in the optimization direction to the optimal objective function value of Q (and, moreover, its feasible set contains the feasible set of Q). Obviously, all relaxations are bounding reformulations.

Lemma 3.2.6. *For any formulation P and spanning pointwise reformulation R^θ, there exists $\xi \in \Theta$ such that R^ξ is a bounding reformulation of P.*

Proof. The proof of Lemma 3.2.4 implies that, if $\xi \in \Theta$ is such that $\mathsf{t}_s(x^*) = [\mathsf{t}_s(\xi_s)](x^*)$ for all $s \leq r$, $x^* \in \mathsf{feas}(R^\xi)$. Similarly, we show $f^\xi(x^*) = f(x^*)$, and therefore:

$$\mathsf{val}(R^{\theta^*}) \leq f^{\theta^*}(x^*) = f(x^*) = \mathsf{val}(P),$$

which establishes the result. □

We remark that the bounding reformulation guaranteed by Lemma 3.2.6 need not be a relaxation in the traditional sense.

Remark 3.2.7. *Often, the replacement process* $\mathsf{t} \leftarrow \mathsf{t}'(\theta)$ *raises a cardinality issue: we replace* r *terms of the original formulation, and we have to use its solution to compute* q *costs, where* r *might in general be different from* q*. We shall discuss this issue in Section 3.3.1 below.* □

Last but not least, note that pointwise reformulations can be used in more general settings than just the MWU algorithm. Although they have been devised with the MWU in mind, what they really achieve is a general mechanism for automatically decomposing the solution process of a MINLP into two phases: one for deciding values of θ, and the other for solving the corresponding pointwise reformulation.

3.3 MWU adaptation for pointwise reformulated MINLP

Once we have derived a pointwise reformulation, we adapt the MWU loop to iteratively: set values for the θ parameters, solve the resulting auxiliary pointwise problem, observe associated costs ψ, update weights ω to update the distribution used to sample values for the θ parameters.

The pseudocode of the MWU algorithm for MINLP is shown in Algorithm 4. It takes a MINLP formulation P and a pointwise reformulation $\underset{\mathsf{t}\leftarrow\mathsf{t}'(\theta)}{\mathsf{ptw}}(P)$ as inputs, and produces a hopefully good solution as output.

Note that p^t is still necessary for the expression of the weighted cumulative error E_{MWU} (see bound (3.1.1)) though it is not used directly in the algorithm.

Next, we shall discuss Steps 6, 8-9 and 11 of Algorithm 4 in more detail.

3.3.1 Sampling parameters

As mentioned in Remark 3.2.7, the dimension r of θ and the dimension q of ω and ψ might be different. Although it is hided with the vector notation in Algorithm 4, the question is how to apply Step 6, since it implicitly assumes that $r = q$. We deal with this issue by defining θ using aggregations (if $r > q$) or disaggregations (if $q > r$) of the values $\tilde{\omega}$ sampled from $[0, \omega]$. Aggregations and disaggregations are obtained by applying any number of operators, such as

Algorithm 4 MWU(P)

initiate: weights $\omega^1 \leftarrow \mathbf{1}$
initiate: parameters $\theta^0 \leftarrow \tilde{\theta}$ (arbitrarily in Θ)
initiate: incumbent $x^* \leftarrow \infty$
for $t \in \{1, \ldots, T\}$ **do**
  normalize distribution $p_i^t \leftarrow \frac{\omega_i^t}{\sum_{j \leq q} \omega_j^t}), \forall i \leq q$
  sample $\tilde{\omega}^t \leftarrow$ Uniform$([0, \omega^t])$
  assign $\theta^t \leftarrow \theta^{t-1} \tilde{\omega}^t$
  solve $\underset{\text{t}\leftarrow\text{t'}(\theta^t)}{\text{ptw}}(P)$, get solution x^t
  optionally refine x^t (*e.g.*, using local descent on P)
  if x^t is better than the incumbent, replace $x^* \leftarrow x^t$
  compute costs $\psi^t \in [-1, 1]^q$ from x^t
  update weights $\omega_i^{t+1} \leftarrow \omega_i^t (1 - \eta \psi_i^t), \forall i \leq q$
end for

products or sums, to $\tilde{\omega}$. Since there are many ways to split products and sums in a prescribed number of different parts, the precise details of this step are heuristic in nature. We note that the MWU performance guarantee on E_{MWU} (see bound (3.1.1)) depends on q but not on r, and so it is not impacted by such details.

3.3.2 Solution and refinement

Solving pointwise reformulations

If P has no integer variable and $\underset{\text{t}\leftarrow\text{t'}}{\text{ptw}}(P)$ is efficient, it is likely to be an LP or a cNLP, both of which can be either solved or accurately approximated in polynomial time. If the pointwise reformulation is not efficient but at least good, it may be a type of non-convex NLP for which we have a practically fast solver which scales reasonably well.

If P has some integer variables and $\underset{\text{t}\leftarrow\text{t'}}{\text{ptw}}(P)$ is good, P might be a MILP or a convex MINLP (cMINLP). This complicates matter, since solving a MILP or a cMINLP to optimality at each iteration is usually computationally costly, even if good solver technologies exist. Note, however, that all the MWU algorithm needs in order for its guarantee to hold is simply an error vector ψ^t at each iteration $t \leq T$. So in fact we can run *any heuristic we like* on the pointwise reformulation.

Refining the pointwise optimum

The refinement step is optional in theory, but computational experience shows it is necessary in practice for the MWU to perform well. If x' is the solution of the pointwise reformulation, any solver which is designed to improve x' with respect to the *original* formulation P (at least locally), can be used to refine x'.

3.3.3 Computing the MWU costs

This is the most critical step of the MWU algorithm, since it influences the performance guarantee. It has two requirements:

(a) $\psi^t \in [-1, 1]^q$ for all $t \leq T$;

(b) for any $t \leq T$, ψ^t measures the error of the current local solution x^t of the pointwise reformulation with respect to the objective and constraints of the original formulation.

We need (a) to prove the MWU relative approximation guarantee, and (b) in order to relate E_{MWU} to the solution quality of the incumbent x^*. This occurs since E_{MWU} depends on p and ψ, p depends on ω, which is updated using ψ, and ψ depends on the local solution x of the pointwise reformulation, which replaces the incumbent x^* whenever it improves it. Note also that x depends on θ through the pointwise reformulation, and that θ is randomly chosen from the discrete distribution p, proportional to ω.

Based on (a) and (b), we compute a scalar α^t related to optimality, and a set $\{\beta^t_\ell \mid \ell \leq m\}$ of vectors related to feasibility. Since we penalize infeasibilities but we generally do not award *better feasibility*, the components of β^t_ℓ are usually required to be in $[0, 1]$ rather than $[-1, 1]$.

More specifically, let $R^{\theta^t} = \underset{\mathsf{t}\leftarrow\mathsf{t'}(\theta^t)}{\mathsf{ptw}}\ (P)$ be the pointwise reformulation at iteration $t \leq T$, let $f(x), g_\ell(x) \leq 0$ (for $\ell \leq m$) be the objective function and constraints of P as per problem (3.2.1), and let $f^{\theta^t}(x), g^{\theta^t}_\ell(x) \leq 0$ be those of R^t (for $\ell \leq m$). After Steps 8-9 of Algorithm 4, we can evaluate the current solution x^t in the pointwise reformulation by computing $f^{\theta^t}(x^t)$ and $g^{\theta^t}_\ell(x^t)$ for each $\ell \leq m$. We define arrays of values, α^t, β^t at each iteration $t \leq T$:

- let α^t be proportional to $f^{\theta^t}(x^t) - f(x^t)$, so as to favor a pointwise reformulation with a lower objective value (for a MINLP in the general minimization form (3.2.1); if considering a maximization problem, α^t should be replaced by $-\alpha^t$);
- for all $\ell \leq m$, let β^t_ℓ be proportional to $\max(g^{\theta^t}_\ell(x^t), 0)$, so as to penalize a pointwise reformulation which makes a feasible solution infeasible.

The arrays α, β can be scaled in any way which makes them satisfy requirement (a) above. We assume this scaling is application-dependent. We can now define ψ^t in a very simple way as

the concatenation of α^t and β^t_ℓ for all $\ell \leq m$, which also fixes $q = 1 + m$. Other application-dependent ways of defining ψ by means of α, β are also possible.

3.4 MWU for an NLP HUC

We present here an adaptation of the MWU framework for a non-linear continuous hydropower unit commitment problem.

The common work [53] also features applications of the MWU to two other problems: the Distance Geometry Problem (DGP) [48], which arises in the positioning of mobile sensors and in protein conformation for example; and a variant of Markowitz's Mean-Variance Portfolio Selection problem (MVPS) [52].

This section is organized as follows. We describe the model of the HUC problem we deal with in 3.4.1. We introduce an adapted pointwise reformulation for that model and check its properties in 3.4.2. We design an adapted variant of the MWU algorithm in 3.4.3. Computational results showing better performance of the designed MWU method over a simple MS method are presented in 3.5.

3.4.1 Model description

As introduced previously, HUC is a problem that arises for short-term scheduling of power plants. Here, we consider a simplification of the the profit-based price-taker problem presented in [11]: the instances feature it is single-reservoir system with one generating unit whose power-flow curve is non-convex and depends on the water level in the reservoir.

Over a uniformly discretized finite time horizon (ranging from one day to a week), given an initial water volume in the reservoir, the goal is to find the optimal schedule of released water flow which maximizes the revenues obtained by providing the generated power to the grid, such that the final volume of water remaining in the reservoir reaches a desired target.

Let us present parameters of the model:

- time horizon $H = \{1, \cdots, \bar{h}\}$
- time period duration τ [h]
- initial water volume V_0 [m^3]
- final target water volume $V_{\bar{h}}$ [m^3]
- volume bounds $\underline{V}$ and $\overline{V}$ [m^3] on the volume v_h, for each $h \in H$
- maximum flow bound $\overline{Q}$ [m^3/s] on the released flow x_h, for each $h \in H$

- maximum ramp-down Q^- and ramp-up Q^+ $[\mathrm{m}^3/\mathrm{s/h}]$ for the flow x_h, for each $h \in H$
- forecasted inflows I_h $[\mathrm{m}^3/\mathrm{s}]$, for each $h \in H$
- forecasted proportional power selling prices Π_h [currency$/\mathrm{MWh}$], for each $h \in H$
- parameters and coefficients $K_1, \ldots, K_6$, $L_1, \ldots, L_6$, $\underline{L}$ and R_0 of a polynomial function which models the generated power y_h, for each $h \in H$.

Let us present variables of the model:

- released water flow $(x_h \mid h \in H)$ $[\mathrm{m}^3/\mathrm{s}]$
- generated power $(y_h \mid h \in H)$ $[\mathrm{MW}]$
- volume of water $(y_h \mid h \in H)$ $[\mathrm{m}^3]$.

The model can be formulated as follows:

$$\max_{x,y,v} \sum_{h \in H} \tau \Pi_h y_h \tag{3.4.1a}$$

$$v_0 = V_0 \tag{3.4.1b}$$

$$v_{\overline{h}} = V_{\overline{h}} \tag{3.4.1c}$$

$$v_{h+1} - v_h = 3600\tau(I_h - x_h) \qquad \forall h \in H \tag{3.4.1d}$$

$$x_h - x_{h+1} \leq \tau Q^- \qquad \forall h \in H \tag{3.4.1e}$$

$$x_{h+1} - x_h \leq \tau Q^+ \qquad \forall h \in H \tag{3.4.1f}$$

$$y_h = \varphi(x_h, v_h) \qquad \forall h \in H \tag{3.4.1g}$$

$$v_h \in [\underline{V}, \overline{V}] \qquad \forall h \in H \tag{3.4.1h}$$

$$x_h \in [0, \overline{Q}] \qquad \forall h \in H, \tag{3.4.1i}$$

where $\varphi(x, v) = 9.81x(\sum_{l=0}^{6} L_l x^l)(\sum_{k=0}^{6} K_k v^k - \underline{L} - R_0 x^2)$.

The objective is to maximize the revenues from power sales in (3.4.1a). Initial and target volumes are respectively set in constraints (3.4.1b) and (3.4.1c). Constraints (3.4.1d) express conservation of water volume from one time period to the next according to external inflows and plant flows; assuming external inflows and plant flows are constant over one time period, flows are linearly intregrated into volumes with the time coefficient 3600τ. Ramping constraints (3.4.1e) and (3.4.1f) limit large shifts in the flow between two consecutive time periods. In constraints (3.4.1g), $\varphi(x, v)$ is the function expressing the power generated depending on the water flow released x and the water volume v in the reservoir. Bounds on volumes and flows are respectively set in constraints (3.4.1h) and (3.4.1i).

3.4.2 Pointwise reformulation

By replacing each non-linear function $\varphi(x_h, v_h)$ (for $h \in H$) with an affine approximation, we derive a pointwise linear reformulation of problem (3.4.1):

$$\max_{x,y,v} \sum_{h\in H} \tau\Pi_h y_h \tag{3.4.2a}$$

$$v_0 = V_0 \tag{3.4.2b}$$

$$v_{\bar{h}} = V_{\bar{h}} \tag{3.4.2c}$$

$$v_{h+1} - v_h = 3600\tau(I_h - x_h) \qquad \forall h \in H \tag{3.4.2d}$$

$$x_h - x_{h+1} \leq \tau Q^- \qquad \forall h \in H \tag{3.4.2e}$$

$$x_{h+1} - x_h \leq \tau Q^+ \qquad \forall h \in H \tag{3.4.2f}$$

$$y_h = \theta_{3h} + \theta_{4h}(x_h - \theta_{1h}) + \theta_{5h}(v_h - \theta_{2h}) \qquad \forall h \in H \tag{3.4.2g}$$

$$v_h \in [\underline{V}, \overline{V}] \qquad \forall h \in H \tag{3.4.2h}$$

$$x_h \in [0, \overline{Q}] \qquad \forall h \in H, \tag{3.4.2i}$$

We remark that, in the pointwise reformulation (3.4.2), the parameter vector θ is structured as the $\bar{h} \times 5$ matrix $(\theta_{1h}, \theta_{2h}, \theta_{3h}, \theta_{4h}, \theta_{5h})_{h\in H}$.

Given a point $(\tilde{x}, \tilde{v}) \in [0, \overline{Q}] \times [\underline{V}, \overline{V}]$, the idea is to have a first-order approximation when θ is properly chosen:

$$\begin{aligned}\varphi(x, v) &\approx \varphi(\tilde{x}, \tilde{v}) + \frac{\partial\varphi}{\partial x}(\tilde{x}, \tilde{v})(x - \tilde{x}) + \frac{\partial\varphi}{\partial v}(\tilde{x}, \tilde{v})(v - \tilde{v}) \\ &= \theta_3 + \theta_4(x - \theta_1) + \theta_5(v - \theta_2) \\ &\text{if } (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = \left(\tilde{x}, \tilde{v}, \varphi(\tilde{x}, \tilde{v}), \frac{\partial\varphi}{\partial x}(\tilde{x}, \tilde{v}), \frac{\partial\varphi}{\partial v}(\tilde{x}, \tilde{v})\right)\end{aligned}$$

Lemma 3.4.1. *The pointwise reformulation* (3.4.2) *is spanning.*

Proof. For each $h \in H$, the following holds: if $x'_h \in [0, \overline{Q}]$ and $v'_h \in [\underline{V}, \overline{V}]$, the replacement term $\theta_{3h} + \theta_{4h}(x_h - \theta_{1h}) + \theta_{5h}(v_h - \theta_{2h})$ matches the replaced term $\varphi(x_h, v_h)$ for all values of θ satisfying $\theta_{1h} = x'_h$, $\theta_{2h} = v'_h$ and $\theta_{3h} = \varphi(x'_h, v'_h)$. □

Proposition 3.4.2. *For each globally optimal solution $X^* = (x^*, v^*, y^*)$ of problem* (3.4.1)*, there is a $\theta^* \in \mathbb{R}^{5\bar{h}}$ such that X^* is a globally optimal solution of problem* (3.4.2)*.*

Proof. Let us show that X^* is a globally optimal solution of the pointwise problem (3.4.2) for

$$\theta^*_h = (x^*_h, v^*_h, \varphi(x^*_h, v^*_h), \frac{\partial\varphi}{\partial x_h}(x^*_h, v^*_h), \frac{\partial\varphi}{\partial v_h}(x^*_h, v^*_h)),$$

where $h \in H$. Since this definition of θ^* satisfies the spanning property in Proposition 3.4.1, we can invoke Lemma 3.2.4 to conclude that X^* is feasible for (3.4.2). Since X^* is a global optimum of (3.4.1), in particular it is also a local optimum, and therefore it satisfies the Karush-Kuhn-Tucker (KKT) conditions. By the choice of θ^*, the gradients of the objective function and the constraints of (3.4.2) at X^* are identical to gradients of objective and constraints of (3.4.1) at X^*. Therefore, X^* also satisfies first-order optimality conditions of (3.4.2) when parametrized by θ^*. Since (3.4.2) is an LP, any KKT point is also a global optimum, which concludes the proof. □

Theorem 3.4.3. *Formulation* (3.4.2) *is an exact and efficient pointwise reformulation of problem* (3.4.1).

Proof. Exactness of (3.4.2) follows by Proposition 3.4.2. Efficiency follows because (3.4.2) is an LP, which can be solved in polynomial time. □

3.4.3 MWU adaptation for the HUC

We discuss here the adaptation of Algorithm 4 to the HUC application setting.

Computing the MWU costs For each time period of the scheduling horizon, an upper bound on the contribution to the overall revenues is given by:

$$\bar{\Pi}\bar{\varphi} \triangleq \max_h \Pi_h \max_{x,v} \varphi(x,v).$$

This allows us to define MWU costs at each iteration t:

$$\forall h \in H \quad \psi_h^t \leftarrow \frac{\bar{\Pi}\bar{\varphi} - \Pi_h y_h^t}{\bar{\Pi}\bar{\varphi}}. \tag{3.4.3}$$

This definition of ψ^t is close to the interpretation of α^t given in Section 3.3.3. For each time period, the corresponding revenues are scaled with respect to the best contribution possible. We remark that minimizing $\sum \psi_h$ is equivalent to maximizing the orginal objective $\sum \tau \Pi_h y_h$ since the transformation is affine.

Sampling parameters The pointwise reformulation (3.4.2) we employ for the HUC relies on a parameter matrix $\theta^t = (\theta_1^t, \theta_2^t, \theta_3^t, \theta_4^t, \theta_5^t)$. Note that θ^t's dimension is $\bar{h} \times 5$, and that the first two vectors are actually points in the (x,v)-space. The cost vectors ψ^t, and the weights ω^t, are $\bar{h}$-dimensional. We decide to set θ_1^t and θ_2^t deterministically while $(\theta_3^t, \theta_4^t, \theta_5^t)$ are set according to the sample $\tilde{\omega}_h^t \sim \text{Uniform}([0, \omega_h^t])$. Starting from the solution of the previous

iteration (x^{t-1}, v^{t-1}), the power function is modified by changing coefficients of the first-order approximation around (x^{t-1}, v^{t-1}).

More precisely: $\forall h \in H$,

$$\begin{array}{rcl} \theta^t_{1h} & \leftarrow & x^{t-1}_h, \\ \theta^t_{2h} & \leftarrow & v^{t-1}_h, \\ \theta^t_{3h} & \leftarrow & \tilde{\omega}^t_h \varphi(\tilde{x}^t_h, \tilde{v}^t_h), \\ \theta^t_{4h} & \leftarrow & \tilde{\omega}^t_h \frac{\partial \varphi}{\partial x_h}(\tilde{x}^t_h, \tilde{v}^t_h), \\ \theta_{5h} & \leftarrow & \tilde{\omega}^t_h \frac{\partial \varphi}{\partial v_h}(\tilde{x}^t_h, \tilde{v}^t_h). \end{array} \tag{3.4.4}$$

MWU guarantee Since the distance to $\bar{\Pi}\bar{\varphi}$ is always penalized with non-negative ψ, the performance guarantee of the MWU applied to the HUC is:

$$\min_{t \le T} \sum_{h \in H} \psi^t_h p^t_h \le \frac{1}{T} \left(\frac{\ln \bar{h}}{\eta} + (1+\eta) \min_{h \in H} \sum_{t \le T} \psi^t_h \right). \tag{3.4.5}$$

3.5 Computational results

In this section, we present comparative computational results to validate the behavior of the MWU algorithm for (MI)NLP in practice. We always compare the MWU with a "randomized greedy" MS heuristic, which is possibly the most similar existing algorithm to the MWU for (MI)NLP, on a fixed number T of iterations. We chose $\eta = 0.5$ and $T = 20$ for most of the experiments (unless stated otherwise).

The randomized greedy MS we implement constructs a feasible solution at each iteration t. More precisely, in a subloop on $h \in H$, we sample at random an initial flow value x_h in a domain that satisfies locally volume bounds (3.4.1h), and ramping constraints (3.4.1e)-(3.4.1f), updates volume v_{h+1} according to (3.4.1d) and anticipates on final target volume (3.4.1c).

3.5.1 Test configuration

We use the IPOPT solver [20] as the local NLP solver in both the MWU (Step 9 of Algorithm 4) and MS (Step 3 of Algorithm 3) methods; and CPLEX [38] as the LP solver for the pointwise reformulation of the MWU (Step 8 of Algorithm 4). External solvers are invoked without time limits. Tests were executed on a machine configured with eight 64-bit Intel Xeon CPU E5504 running at 2.00 GHz and 11.7 GB of RAM, running the Linux operating system.

The authors of [11] provided us with three simplified instances, sharing the following common characteristics:

- $\bar{h} = 168, \tau = 1$ [h]

- $V_0 = V_{\bar{h}} = 21078580, \underline{V} = 15000000, \bar{V} = 33000000$ [m^3]
- $\bar{Q} = 42$ [m^3/s]
- $Q^- = Q^+ = 70$ [$\mathrm{m}^3/\mathrm{s/h}$]
- $R_0 = 0.01$
- $\underline{L} = 385$
- $L = (L_\ell \mid \ell \in \{0, \ldots, 6\}) =$
  $= (4.0986, -1.2554, 0.1605, -9.762 \times 10^{-3}, 3.0943 \times 10^{-4}, -4.9293 \times 10^{-6}, 3.1152 \times 10^{-8})$
- $K = (K_k \mid k \in \{0, \ldots, 6\}) =$
  $= (3.074 \times 10^2, 3.88 \times 10^{-5}, -4.37 \times 10^{-12}, 0.265 \times 10^{-19}, -8.87 \times 10^{-27}, 1.55 \times 10^{-34}, -1.11 \times 10^{-42})$

All of the the values of the inflows I (m^3/s) and prices Π (currency$/\mathrm{MWh}$) per time period are reported in Table C.1 in the Appendix C.

In order to work with a larger test set, nine further instances were generated from each of the three original instances by uniformly sampling price vectors $\Pi = (\Pi_h \mid h \in \{1, \ldots, 168\})$ in $[\min \Pi, \max \Pi]$. The generated instances are noted `A2` to `A10`, `B2` to `B10`, `C2` to `C10`. For example, instance `C2` features the same data as `C1` except for the prices, which, however, lay in the same range.

3.5.2 Comparative results on solution quality and CPU time

Both MWU and MS are configured with $T = 20$ iterations. Tests are run on the (`A1-C10`) 30-instance set introduced in Section 3.5.1. The comparative computational results are reported in Table 3.1 as follows:

- the first column shows the instance name
- second and third columns show objective value and CPU time (in seconds of user time) for the MS algorithm
- the fourth and fifth columns show objective value and CPU time (in seconds of user time) for the MWU algorithm
- the fifth column shows relative objective value improvement from MS to MWU computed as
$$\Delta = \frac{\mathsf{val}(\mathsf{MWU}) - \mathsf{val}(\mathsf{MS})}{\mathsf{val}(\mathsf{MWU})} \tag{3.5.1}$$
- the sixth column shows relative time improvement from MS to MWU computed as
$$\Lambda = \frac{\mathsf{cpu}(\mathsf{MS}) - \mathsf{cpu}(\mathsf{MWU})}{\mathsf{cpu}(\mathsf{MWU})}. \tag{3.5.2}$$

For the last two columns, the comparison metrics are summarized in the last line with the average (avg) and the standard deviation (std) across all 30 instances.

	MS		MWU		MS vs. MWU	
Instance	objective	CPU	objective	CPU	Δ	Λ
A1	4.08E+4	3.5	4.17E+4	3.0	2.11%	15.79%
A2	5.24E+4	3.6	5.32E+4	3.0	1.41%	21.81%
A3	4.78E+4	3.6	4.97E+4	3.0	3.75%	20.40%
A4	4.87E+4	3.6	4.98E+4	3.1	2.23%	16.35%
A5	5.11E+4	3.4	5.19E+4	3.0	1.50%	13.80%
A6	5.02E+4	3.6	5.10E+4	3.1	1.62%	14.47%
A7	5.11E+4	3.5	5.20E+4	3.0	1.79%	16.28%
A8	5.18E+4	3.5	5.24E+4	3.0	1.14%	13.86%
A9	5.15E+4	4.1	5.21E+4	3.0	1.10%	35.97%
A10	5.19E+4	3.6	5.13E+4	3.1	-1.24%	14.52%
B1	1.98E+4	2.3	1.98E+4	2.5	0.00%	-7.17%
B2	2.67E+4	2.9	2.67E+4	2.7	0.00%	6.27%
B3	2.53E+4	2.5	2.53E+4	2.5	0.00%	-1.99%
B4	2.53E+4	2.7	2.53E+4	2.6	0.00%	5.08%
B5	2.60E+4	2.9	2.60E+4	2.8	0.00%	2.11%
B6	2.60E+4	2.7	2.60E+4	2.6	0.00%	1.15%
B7	2.62E+4	2.6	2.62E+4	2.4	0.00%	5.74%
B8	2.65E+4	2.7	2.65E+4	2.6	0.00%	4.71%
B9	2.62E+4	2.9	2.62E+4	2.7	0.00%	9.29%
B10	2.61E+4	2.7	2.61E+4	2.5	0.00%	7.20%
C1	4.94E+4	3.9	5.17E+4	2.7	4.39%	47.92%
C2	6.96E+4	3.7	6.99E+4	3.0	0.36%	25.25%
C3	6.45E+4	3.8	6.65E+4	2.9	2.97%	33.33%
C4	6.50E+4	3.5	6.70E+4	2.9	2.96%	19.24%
C5	6.75E+4	3.4	6.87E+4	2.8	1.68%	20.92%
C6	6.74E+4	3.7	6.82E+4	2.8	1.17%	32.97%
C7	6.82E+4	3.7	6.87E+4	2.8	0.78%	34.55%
C8	6.84E+4	3.6	6.86E+4	2.8	0.42%	30.43%
C9	6.80E+4	3.6	6.88E+4	2.7	1.19%	34.34%
C10	6.82E+4	3.5	6.89E+4	2.9	1.13%	20.62%
avg	4.69E+4	3.3	4.75E+4	2.8	**1.08%**	**17.17%**
std	1.73E+4	0.5	1.77E+4	0.2	1.27%	13.01%

Table 3.1: Objective and CPU time of MS and MWU, relative objective improvement Δ and CPU time improvement Λ from MS to MWU.

The MWU algorithm is always better than the MS algorithm objective-wise and the improvement is relatively small. It must be emphasized, however, that a few percentage points in the objective functions of energy-related optimization problems often translate in consistent savings in absolute terms. As for the time performance, the MWU algorithm is almost 20% faster than the MS algorithm on average: this is an extremely desirable feature in short-term scheduling problems such as this.

3.5.3 Sensitivity to instance size

In order to study the relative performance sensitivity of the MWU algorithm on instance size, we artificially vary the time horizon $\bar{h}$ of the instances from one day to two weeks.

The instances introduced in Section 3.5.1 are one week long with hourly time steps ($\bar{h} = 168$); we now consider instances with $\bar{h}$ in $\{24, 48, \ldots, 168, \ldots, 336\}$, defined as follows:

- when $\bar{h} \leq 168$, data from the 168-long instances is cropped to $h \in [1, \bar{h}]$;
- when $\bar{h} > 168$, data for $h \in [1, 168]$ is identical to the 168-long instances, and data for $h \in [169, \bar{h}]$ is duplicated from the first interval and taken from time period $h - 168 \in [1, 168]$.

Both MWU and MS are configured with $T = 20$. For each instance subset `A` (`A1` to `A10`), `B` (`B1` to `B10`) and `C` (`C1` to `C10`), the comparative computational results for the 14 different sizes are summarized by average and standard deviation in Table 3.2, as follows:

- the first column shows the instance subset
- the second column shows instance size $\bar{h}$
- the third and fourth columns show average (avg) and standard deviation (std) of the relative objective improvement Δ taken over the 10 instances in the first column having size specified in the second
- the fifth and sixth columns show average (avg) and standard deviation (std) of the relative CPU time improvement Λ taken over the 10 instances in the first column having size specified in the second.

For the last four columns, the last line shows the averages of the corresponding metrics taken over the whole 420 ($= 3 \times 10 \times 14$) tested instances.

The results from Table 3.2 corroborate those from Table 3.1: almost always, the MWU algorithm outputs a slightly better objective than the MS algorithm while outperforming the MS algorithm in CPU time as instance size grows.

We graphically compare CPU times averaged over instances in Figure 3.1. Each point corresponds to (size, average CPU time) over all the instances `A1-C10`.

Figure 3.1 shows that, for the range of tested sizes, the CPU time taken by the MS algorithm to solve larger instances increases more compared the MWU.

3.5.4 Comparative results on primal integral

The "primal integral" ([9]) is a performance measure that is relevant to evaluate and compare primal heuristics. Given a time limit T_{lim} and an upper bound (of a maximization problem), the

Instance subset	Instance size $\bar{h}$	Δ avg	Δ std	Λ avg	Λ std
A	24	0.71%	5.92%	-7.89%	15.19%
	48	1.59%	3.57%	1.83%	5.31%
	72	1.22%	1.96%	0.87%	3.25%
	96	0.94%	1.81%	11.52%	7.46%
	120	1.93%	1.55%	12.35%	7.58%
	144	0.89%	1.45%	14.91%	6.31%
	168	1.52%	1.25%	19.37%	9.73%
	192	2.04%	1.49%	30.93%	4.28%
	216	2.41%	1.62%	34.27%	6.97%
	240	1.64%	1.33%	36.43%	5.85%
	264	2.27%	1.13%	50.68%	11.56%
	288	2.35%	1.57%	41.72%	7.56%
	312	2.31%	1.17%	42.19%	8.28%
	336	2.16%	1.43%	38.00%	6.33%
B	24	0.00%	0.00%	-6.29%	2.43%
	48	0.00%	0.00%	0.51%	3.10%
	72	0.00%	0.00%	0.78%	3.04%
	96	0.00%	0.00%	2.43%	2.61%
	120	0.00%	0.00%	3.56%	3.88%
	144	0.00%	0.00%	3.54%	3.21%
	168	0.00%	0.00%	2.79%	4.30%
	192	0.00%	0.00%	6.99%	4.66%
	216	0.00%	0.00%	7.55%	6.37%
	240	0.00%	0.00%	10.97%	6.36%
	264	0.00%	0.00%	11.07%	8.49%
	288	0.00%	0.00%	7.44%	5.10%
	312	0.00%	0.00%	12.93%	7.33%
	336	0.00%	0.00%	18.93%	8.37%
C	24	0.00%	0.00%	-10.15%	16.72%
	48	0.28%	0.87%	-0.75%	5.07%
	72	1.95%	3.14%	7.46%	7.41%
	96	2.73%	4.41%	11.02%	7.22%
	120	2.73%	2.42%	12.97%	7.86%
	144	2.28%	3.21%	21.41%	8.02%
	168	1.71%	1.32%	26.86%	8.40%
	192	1.53%	1.36%	34.41%	6.27%
	216	1.41%	1.90%	29.89%	8.66%
	240	1.72%	1.40%	42.63%	7.66%
	264	1.81%	1.85%	50.31%	14.46%
	288	1.95%	2.13%	44.40%	13.97%
	312	1.92%	2.07%	50.88%	11.96%
	336	1.32%	1.61%	47.13%	12.83%
overall avg		1.13%	2.01%	18.54%	19.46%

Table 3.2: Relative objective improvement Δ and CPU time improvement Λ from MS to MWU, according to instance subset and instance size $\bar{h}$.

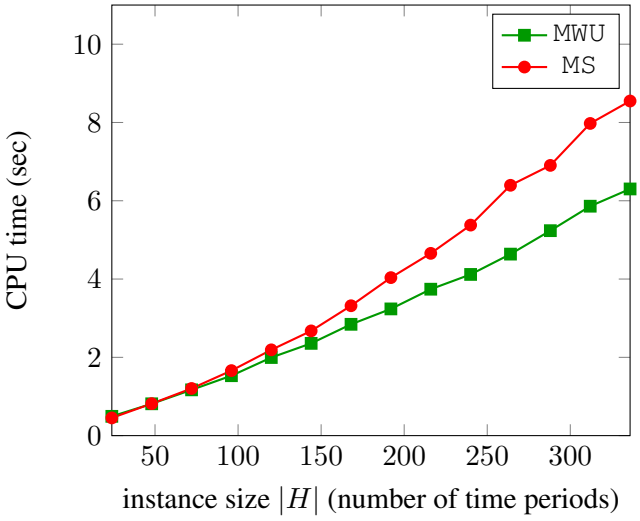

Figure 3.1: Average of CPU time (sec) vs. instance size $|H| = \bar{h}$

primal integral is the time integral over the solution process $[0, T_{lim}]$ of the gap of the incumbent solution with respect to the upper bound. Thus both solution quality and speed performance are taken into account. Indeed, when, for example comparing algorithms A and B, if, at a given time, A finds a better solution, or if A finds the same solution than B earlier in time, A's primal integral will be lower than B's.

Both MWU and MS are configured with $T_{lim} = 20s$. Tests are run on the (`A1-C10`) 30-instance set with size $\bar{h} = 336$ introduced in Section 3.5.3. The comparative computational results are reported in Table 3.3 as follows:

- the first column shows the instance name
- the second column shows the primal integral Γ for the MS algorithm
- the third column shows the primal integral Γ for the MWU algorithm.

For the last two columns, the comparison metrics are summarized in the last line with the average (avg) and the standard deviation (std) across all 30 instances.

In some sense, the results of Table 3.3 synthesize the ones obtained in Table 3.1: the MWU on average finds better solutions faster.

3.5.5 Sensitivity to varying initializations

In this section we empirically show that the MWU method is robust to varying initial conditions. To do so, we define the *objective dispersion* Ξ as the standard deviation over the average of a

	MS	MWU
Instance	Γ	Γ
A1	158.2%	67.3%
A2	77.5%	26.3%
A3	135.2%	39.0%
A4	81.8%	28.8%
A5	74.0%	38.8%
A6	94.0%	32.8%
A7	104.9%	35.7%
A8	91.0%	31.4%
A9	80.4%	35.8%
A10	71.9%	28.0%
B1	28.0%	22.0%
B2	40.0%	24.0%
B3	35.0%	22.0%
B4	35.0%	24.0%
B5	31.0%	25.0%
B6	35.0%	24.0%
B7	33.0%	21.0%
B8	40.0%	22.0%
B9	34.0%	25.0%
B10	33.0%	22.0%
C1	189.9%	71.0%
C2	102.6%	27.4%
C3	94.3%	26.7%
C4	78.8%	28.3%
C5	93.9%	35.0%
C6	128.5%	30.6%
C7	57.6%	19.9%
C8	78.1%	25.2%
C9	113.7%	33.5%
C10	67.8%	18.0%
avg	77.3	**30.3%**
std	40.7%	**12.0%**

Table 3.3: Primal integral Γ of MS and MWU

sample of objective function values collected from a sequence of runs with randomly chosen starting vectors θ.

The MWU search is somewhat diversified due to the random sampling of θ (Step 6 of Algorithm 4) which are used to define the pointwise reformulation. Whenever the ψ costs are only defined through feasibility (*i.e.*, ψ is proportional to β, see Section 3.3.3), it is easy to show that the weights ω can only decrease during MWU execution (Step 12 of Algorithm 4), which means the sampling domain is increasingly small. In the extreme case where all weights ω are set to zero for all iterations, the search is deterministic and only depends on the values initially set for parameters θ. We therefore test whether MWU results are conditioned by initialization.

To this end, we look at the dispersion of 20 MWU runs (all of them configured with $T = 20$

iterations), started from 20 different randomly sampled vectors θ that are obtained with the "greedy randomized" MS. In addition, for each sampled initial point, a local descent is independently performed to solve problem (3.4.1); the best of these descents is equivalent to the result of a 20-iteration MS run.

Tests are run on the (`A1-C10`) 30-instance set introduced in Section 3.5.1 (with $\bar{h} = 168$ time periods). Computational results are reported in Table 3.4, as follows:

- the first column shows the instance name
- the second column shows the objective dispersion Ξ
- the third column shows average objective function value improvements from MS to each of the 20 MWU runs (Δ_{avg})
- the fourth column indicates whether the worst objective function value obtained over the 20 MWU runs is as good as the MS result.

The comparison metrics are summarized in the last line with the average across all 30 instances.

On average, the dispersion of the MWU results is low (avg(Ξ)=0.38%), which shows low sensitivity to variability in initialization. In addition, the dispersion of the MWU results is lower on average than the average relative improvement ($\mathsf{avg}(\Delta_{\mathsf{avg}}) = 1.02\%$), thus showing that the MWU is consistently better than MS; this comparison is all the more relevant as the 20 MWU runs and the 20 MS iterations are started with the same values. Similarly, resorting to another variation statistic, more three out of four times (76.67%) even the worst MWU run yields a result as good as the MS.

Therefore, the MWU method does not suffer from a mitigated diversification compared to the MS method, and is robust to varying initial conditions.

3.5.6 Importance of the pointwise and refinement steps

In this section we show that solving the pointwise problem within the MWU method (Step 8 of Algorithm 4) partially accounts for its good performance compared to the MS.

Both MWU and MS feature a randomly started local descent. On the one hand, the initialization phase for MS is based on sampling from a uniform distribution. On the other hand, the MWU has a specific routine to sample parameters, which are then used to solve the pointwise problem (3.4.2), whose solution is fed as a starting point to perform a local descent (refinement step) for the original problem (3.4.1). In the MWU loop applied to the HUC, a solution of the pointwise problem is bound to be relatively good for the original problem, whereas MS's initial point may be very poor. To test if the latter characteristic is responsible for the good performances of the MWU, we compare results against an enhanced MS with a pointwise feasibility recovery step – denoted MSptw – as described in Algorithm 5.

Instance	Ξ	Δ_{avg}	MWU $\geq$ MS
A1	0.57%	3.02%	TRUE
A2	0.46%	0.92%	TRUE
A3	0.61%	4.02%	TRUE
A4	0.72%	2.07%	TRUE
A5	0.72%	2.29%	TRUE
A6	0.74%	1.76%	TRUE
A7	0.66%	0.79%	FALSE
A8	0.56%	0.51%	FALSE
A9	0.58%	0.37%	FALSE
A10	0.62%	0.42%	FALSE
B1	0.00%	0.00%	TRUE
B2	0.00%	0.00%	TRUE
B3	0.00%	0.00%	TRUE
B4	0.00%	0.00%	TRUE
B5	0.00%	0.00%	TRUE
B6	0.00%	0.00%	TRUE
B7	0.00%	0.00%	TRUE
B8	0.00%	0.00%	TRUE
B9	0.00%	0.00%	TRUE
B10	0.00%	0.00%	TRUE
C1	0.42%	4.09%	TRUE
C2	0.52%	0.72%	TRUE
C3	0.59%	2.74%	TRUE
C4	0.59%	1.82%	TRUE
C5	0.46%	1.63%	TRUE
C6	0.52%	1.65%	TRUE
C7	0.47%	0.86%	TRUE
C8	0.42%	0.51%	FALSE
C9	0.47%	-0.31%	FALSE
C10	0.63%	0.71%	FALSE
avg	**0.38%**	**1.02%**	**76.67%**

Table 3.4: Objective dispersion Ξ, average relative objective value improvement from MS Δ_{avg}, and worst-case objective value comparison to MS (last column) for the 20 MWU runs.

Algorithm 5 MSptw(P)

1: **while** $t \leq T$ **do**
2: sample θ^t uniformly at random
3: solve $\underset{\mathsf{t}\leftarrow\mathsf{t}'(\theta^t)}{\mathsf{ptw}}\ (P)$, get solution x^t
4: refine x^t (*e.g.*, using local descent)
5: if x^t is better than the incumbent, replace $x^* \leftarrow x^t$
6: increase t
7: **end while**

Both MWU and MSptw are configured with $T = 20$. Tests are run on the (A1-C10) 30-instance set introduced in Section 3.5.1, (with $\bar{h} = 168$ time periods). The comparative computational results are reported in Table 3.5 as follows:

- the first column shows instance name
- the second and third columns show objective value and CPU time (in seconds) for the MSptw algorithm
- the fourth and fifth columns show objective value and CPU time (in seconds) for the MWU algorithm
- the fifth column shows relative objective value improvement Δ from MSptw to MWU
- the sixth column shows relative time improvement ratio Λ from MSptw to MWU.

For the last two columns, the comparison metrics are summarized in the last line with the average (avg) and the standard deviation (std) across all 30 instances.

The MWU method still outperforms the enhanced MSptw method both with respect to the objective function value althoug the improvement margin is less marked as with the plain MS method (see Table 3.1). As for time performance, the MSptw is much slower than the MWU and even slower than the MS. Integrating the pointwise step in the MS yields better solutions at the cost of longer solution times. Only the joint implementation of the pointwise step and the MWU sampling – that is to say the MWU algorithm – allows to obtain better solutions earlier in time than the MS algorithm.

3.6 Conclusion

This chapter was about the adaptation of the Multiplicative Weights Update algorithm to the Mixed Integer Non-Linear Programming setting. Such adaptation was based on the derivation of a parametrized reformulation denoted pointwise. We defined desirable properties for deriving pointwise reformulation and provided guidelines to adapt the algorithm step-by-step. Unlike most heuristics, we showed that the MWU method still retains a relative approximation guarantee in the NLP and MINLP settings. To deliver a proof of its applicability, we implemented the method to solve a hard NLP HUC problem and benchmarked it computationally against the Multi-Start method. We found it compared favorably to the MS, which offers no approximation guarantee.

There are several directions to consolidate this work. Empirical evidence could be extended: testing larger data sets, testing a more sophisticated mathematical model, testing other applications. Also, the analysis of the computational performance could be further investigated. For

Instance	MWU		MSptw		MWU vs. MSptw	
	objective	CPU	objective	CPU	Δ	Λ
A1	4.17E+04	3.05	4.06E+04	5.3	2.65%	73.77%
A2	5.32E+04	2.97	5.24E+04	5.45	1.53%	83.50%
A3	4.97E+04	3.06	4.95E+04	6.38	0.34%	108.50%
A4	4.98E+04	3.19	4.86E+04	5.45	2.41%	70.85%
A5	5.19E+04	3.05	5.15E+04	5.05	0.81%	65.57%
A6	5.10E+04	3.17	5.11E+04	5.81	-0.11%	83.28%
A7	5.20E+04	3.08	5.12E+04	5.66	1.49%	83.77%
A8	5.24E+04	3.09	5.16E+04	5.85	1.47%	89.32%
A9	5.21E+04	3.12	5.19E+04	5.97	0.29%	91.35%
A10	5.13E+04	3.16	5.13E+04	5.55	-0.07%	75.63%
B1	1.98E+04	2.54	1.98E+04	4.59	0.00%	80.71%
B2	2.67E+04	2.71	2.67E+04	4.25	0.00%	56.83%
B3	2.53E+04	2.51	2.53E+04	3.86	0.00%	53.78%
B4	2.53E+04	2.6	2.53E+04	4.11	0.00%	58.08%
B5	2.60E+04	2.84	2.60E+04	4.73	0.00%	66.55%
B6	2.60E+04	2.59	2.60E+04	4.19	0.00%	61.78%
B7	2.62E+04	2.49	2.62E+04	3.85	0.00%	54.62%
B8	2.65E+04	2.54	2.65E+04	4.28	0.00%	68.50%
B9	2.62E+04	2.72	2.62E+04	4.44	0.00%	63.24%
B10	2.61E+04	2.52	2.61E+04	4.1	0.00%	62.70%
C1	5.17E+04	2.65	4.94E+04	5.51	4.38%	107.92%
C2	6.99E+04	2.95	7.02E+04	5.51	-0.46%	86.78%
C3	6.65E+04	2.89	6.55E+04	6.01	1.44%	107.96%
C4	6.70E+04	2.95	6.58E+04	5.43	1.78%	84.07%
C5	6.87E+04	2.85	6.86E+04	5.25	0.17%	84.21%
C6	6.82E+04	2.8	6.68E+04	5.69	2.08%	103.21%
C7	6.87E+04	2.76	6.88E+04	5.84	-0.18%	111.59%
C8	6.86E+04	2.83	6.82E+04	5.39	0.65%	90.46%
C9	6.88E+04	2.69	6.85E+04	5.5	0.42%	104.46%
C10	6.89E+04	2.95	6.89E+04	5.34	0.08%	81.02%
avg	4.75E+04	2.84	4.71E+04	5.14	**0.71%**	**80.47%**
std	1.77E+04	0.22	1.75E+04	0.72	1.10%	17.49%

Table 3.5: Objective and CPU time of MWU and MS with ptw initialization, relative objective improvement Δ and CPU time improvement Λ from MSptw to MWU.

all metaheuristics, characterizing the method – an adaptive, random, sequential algorithm in our case ([30]) – and "deconstructing its components" ([70]) enables to design relevant control experiments to gain better insights. More specifically to the MWU, testing the pointwise property and the tightness of the MWU guarantee could be instructive. Finally, from a more theoretical perspective, we hope that the MWU guarantee can be better exploited with a more elaborate definition of the pointwise reformulation and the MWU costs in order to turn this heuristic into an approximation algorithm.

Chapter 4

Conclusion and perspectives

In Chapter 2, we dealt with feasibility issues of a real-world hydropower unit commitment. The problem was formulated as a mixed-integer linear programming model. Given real-world instances, we reformulated the model to make the problem feasible. Compared with a standard HUC problem, the model featured two additional specifications: discrete operational points of the power-flow curve and mid-horizon and final strict targets for reservoir levels. We used a step-by-step approach to systematically exhibit and cope with one source of infeasibility at a time, namely numerical errors and model infeasibilities.

Chapter 3 was about the adaptation of the Multiplicative Weights Update algorithm to the Mixed Integer Non-Linear Programming setting. Such adaptation was based on the derivation of a parametrized reformulation denoted pointwise. We defined desirable properties for deriving pointwise reformulation and provided generic guidelines to adapt the algorithm step-by-step. Unlike most metaheuristics, we showed that our MWU metaheuristic still retains a relative approximation guarantee in the NLP and MINLP settings. To deliver a proof of its applicability, we implemented the method to solve a hard NLP HUC problem and benchmarked it computationally against the Multi-Start method.. We found it compared favorably to the MS, which offers no approximation guarantee.

On a wider perspective, we have tried to show optimization for short-term deterministic hydropower scheduling is an important and rich field as many issues remain to be investigated. With respect to its application, we restate that the solutions to these scheduling problems are essential. Indeed, producers must devise schedules – to operate hydropower plants safely, to provide electricity reliably on the grid and to make a profit. In addition, Producers are legally bound to communicate enforceable schedules to the TSO and to let the Energy Regulation Committee (CRE) examine its scheduling models.

From a practical viewpoint, posing the right problem is a challenge itself before designing efficient solution methods. Unfortunately it is not always addressed explicitly in optimization literature. hydropower production is a complex system that cannot be represented as is. Therefore it is important to first elicit the driving requirements and to choose how to model them, then to validate the model along with data and solution implementation. The data and model pre-processing proposed in Chapter 2 was a step towards that direction. Several characteristics have yet to be modeled: for example, the time it takes water to flow actually depends on the flow quantity; also hydroelectric plants are themselves made of network of pipes, turbines and other devices that can operate in various configurations. Modeling those features should be appraised with regards to the potential benefit both in terms of economic profit and of usability for operators. Whatever the new features, and however they are modeled, it is tantamount to make sure they are not conflicting and lead to infeasibility. In that case, changing the modeling is necessary. Sequentially considering constraint satisfaction problems according to their priority before solving the original problem, as in Chapter 2 is a relevant option.

Methodologically speaking, hydropower scheduling problems are a challenging boon for mathematical optimization. A model, no matter how close to reality it is, is useful only if there are methods to solve in a timely fashion. Computational efficiency is critical as schedulers have very little time to compute, analyze, edit and transmit the schedules. Yet, conventional mathematical programming methods are not always efficient enough to solve some of these problems or even obtain feasible solutions. The heuristic developed in Chapter 3 is an attempt to provide good schedules very fast for a slightly different HUC problem. Looking at it differently, in Chapter 3, we use a HUC problem – known to be difficult to solve – to apply and benchmark a novel heuristic for MINLP. Also, for the problem presented in Chapter 2, several MILP-inspired methods are to be tried by focusing on specific classes of valleys. All in all, hydropower scheduling problems are an opportunity to develop new adapted solution method and test generic solution methods.

Appendix A

Electricity production management

A.1 Hydroelectric structures

Let us describe the generic equipment of a hydroelectric structure, as presented in [77], following the direction of natural flows:

Upstream Incoming flows from precipitations or upstream rivers are stored in the upstream/ uphill reservoir located above the structure. Run-of-the-river structures are not below a reservoir but directly receives water from an upstream river.

Dam The upstream reservoir is *closed* by a dam to impound water.

Gate/valve When it is open, the gate allows to discharge water from the upstream reservoir.

Penstock Water discharged from the upstream reservoir runs in a penstock pipe to reach the turbine.

Turbine/pump Turbines/pumps are the devices that allow conversion between motion of running water and rotation of generators/engines.

Generator/engine Generators/engines are the devices that allow conversion between rotation of turbines/pumps and generation/consumption of electricity.

Tailrace Water discharged after the turbine runs in a tailrace pipe to go downstream

Spillway Some structures can discharge water from upstream in a spillway pipe that bypasses the turbine so that water flows without engaging the generator.

Downstream Outgoing discharged flows go downstream/downhill. Pump-storage structures feature a downstream reservoir from where water is pumped. Water released from dams

or run-of-the-river structures goes into a downstream river or a downstream reservoir if there is another hydroelectric structure.

A.2 Stakeholders of the electric system

Let us present the roles of and interactions between the major stakeholders involved in the electric system.

Generating companies Producers – also commonly referred to as generating companies (GenCos) – own or lease physical generating assets – that is to say power plants – to produce electricity and physically provide it on the grid. The energy bundle (or mix) of a generating company is the distribution of production assets according to their types. Generating companies can sell their output on on wholesale electricity markets and/or to its end users if they are involved in retailing activities.

End users End users consume electricity from the grid. They may be households or organizations (schools, street lighting, stores, SMEs, industrial facilities, etc.). The load is the aggregation of all electric consumptions at a given time.

Grid operators Basically, the grid can be described as the junction of two networks: the (high voltage) transmission network and the (medium to low voltage) distribution network. In transmission networks, nodes are denoted buses and those buses are connected by transmission lines. Buses can be injection points of generating companies, junction points to distribution networks, or junction points for exchanges with other transmission networks.

Transmission system operators (TSOs) operate and maintain transmission networks where large-scale producers input their productions.

Distribution network operators (DNOs) operate and maintain distribution networks which deliver power from the transmission network to end users.

In their respective networks, TSOs and DNOs must ensure safe and reliable operation of the electric system. Operations are subject to conventions and requirements related to generation-load balancing, line capacities, regulation of magnitude and frequency.

In France, RTE operates the transmission system and ErDF operates the distribution network.

Retailers Retailers sell electricity to their end users according to terms of subscription or pay-as-you-go contracts. The electricity they supply can be produced by their own power plants if they are also generating companies and/or can be procured from wholesale markets.

EDF is both a producer and a retailer.

Markets Wholesale markets gather competing generating companies, retailers, pure traders and even a few large-scale consumers to buy and sell electricity. According the maturity and type of traded products, there are several exchange marketplaces. There is also the possibility of trading over the counter (OTC).

In retail markets, retailers compete to supply electricity to end users.

Authorities International, national or local authorities set the regulatory environment.

For instance, the liberalization of the electricity economy initiated in the 2000s is a directive from the European Union. Liberalization meant the introduction of competition for electricity production and retail. Indeed, before liberalization, electricity was considered as public service rather than a commodity. In France, EDF was a state-controlled vertically-integrated monopoly responsible of the production-transmission-distribution-retail chain. Now, EDF is a corporation under private, whose shares a majorly detained by the French state; RTE and ErDF are subsidies that operate as independent spin-offs. from a state-controlled monopoly to competitive markets for production and retail,

The liberalization of electricity markets is partial however, states still have control over electric systems; for example states can set tariffs or impose the energy bundle as illustrated with the ongoing nuclear power phase-out in Germany. The interaction of power plants with their local surroundings may also be subject to agreements with local authorities.

Within the current regulatory environment in France, the Energy Regulation Commission (CRE) enforces market rules like ensuring there is no abuse of market power or other unfair competition.

A.3 Unit-commitment settings

Three settings can be distinguished for unit commitment: security-constrained [33], price-maker bidding [24, 54] and price-taker bidding [68] scheduling.

Security-constrained scheduling As implied in its name, security-constrained unit commitment (SCUC) is concerned with the security of the system. The fundamental requirement is to ensure the equilibrium between the scheduled controllable production and the fore-

casted net load. The forecasted net load is computed as follows:

$$\begin{aligned}\text{forecasted net load} =& \text{forecasted demand} \\ & - \text{market position} + \text{OTC contracts}) \\ & - \text{forecasted inevitable production}\end{aligned}$$

Market position and OTC contracts are previously given from mid-term planning; in the short term, trading and scheduling are done iteratively and sequentially to ensure the equilibrium at the cheapest cost.

The provision of ancillary services may also be required to ensure system security. Operating reserve is an example of ancillary service; it is the increased or decreased margin of production a producer can bring on the grid, usually on demand of the TSO; scheduling such reserve in advance ensures rescheduling and real-time balancing is possible.

Security-constrained scheduling is centralized: all the power plants must scheduled jointly since system requirements have to be met by the aggregation of the production levels and reserves.

Price-maker bidding scheduling In that case, scheduling consists in bidding supply curves to an auctioneer. A supply curve is a set of point, each point indicating the minimum price at which a producer is willing to sell for a given production level. Finding the coordinates of each point is similar to analyzing the marginal cost of a SCUC problem. In a market environment, a producer is considered a price-maker when it is so prominent that its contribution to the trade is unavoidable: its supply curve is bound to influence the clearing. The remuneration structure is however different from SCUC and depends on the clearing price of the auction. We will not give further details about this setting. The configuration where few price-maker producers dominate the market is an oligopoly.

Price-taker bidding scheduling The aim of a price-taker producer in the bidding scheduling setting is to maximize its payoff knowing that all the produced power will be sold, that it will be sold at the estimated clearing price, and assuming the clearing price is not affected by the power provided. Since there is no requirement binding the power plants, nor do bids for one plant affect remuneration of another plants, the scheduling can be carried independently for each plant, in a decentralized fashion. This kind of scheduling problem is sometimes referred to as price-based unit commitment (PBUC). When no price-maker producer is involved in the market, we find ourselves in the case of pure and perfect competition.

Appendix B

Mathematical optimization

B.1 Brief background

Mathematical optimization is one of a subfield of Operations Research or Management Science and is sometimes synonym to Mathematical Programming. The discipline Operations Research aims at using scientific tools to solve real-world problems such as programming and managing operations. In that sense, mathematical optimization is a subfield of applied mathematics.

Mathematical optimization comprises two main features: translating real-world problems into mathematical models, and solving mathematically-formulated optimization problems.

B.2 Basic notions

Generally, an optimization problem can be cast in the following form:

$$\underset{x \in X}{\text{Min}}\, f(x) \tag{B.2.1}$$

It is about finding the alternative x that brings the best result $f(x)$ among all admissible alternatives X.

We will now refer to the following formulation to describe basic notions related to optimization problems:

$$\underset{x \in \mathbb{R}^n}{\text{Min}} \quad f(x) \tag{B.2.2a}$$

$$\forall i \in \{1, \ldots, m\} \quad g_i(x) \leq 0 \tag{B.2.2b}$$

$$\forall j \in \{1, \ldots, n\} \quad x_j \in [l_j, u_j] \tag{B.2.2c}$$

B.2.1 Variables

An alternative or solution to the problem is usually encoded with an n-dimensioned vector x of variables; as we choose between alternatives, they are called decision variables.

B.2.2 Bounds and constraints

A solution x is admissible when it lies in the admissible (or feasible) set $X \subset \mathbb{R}^n$. X can be described as the intersection of bounds and $m \in \mathbb{N}$ constraints. For all variables $x_j (j \leq n)$, l_j is a lower bound and u_j is an upper bound, $l_j \in \mathbb{R} \cup \{-\infty\}, u_j \in \mathbb{R} \cup \{+\infty\}$. A constraint expresses a relationship between variables with the help of external parameters. $\forall i \in \{1, \ldots, m\}, g_i : \mathbb{R}^n \to \mathbb{R}$. A problem is infeasible if there is exists no solution $X = \emptyset$.

B.2.3 Objective function

An admissible solution $x \in X$ is better than another one x' if we can compare their respective results: $f(x) \leq f(x')$. That is why the objective function f is also sometimes called the criterion. The codomain of f must be an ordered set and is usually $\mathbb{R}$: $f : \mathbb{R}^n \to \mathbb{R}$. By convention, for infeasible solution $x \notin X, \text{Min } f(x) = +\infty$.

B.2.4 Optimal solution

An admissible solution $x^* \in X$ is optimal when: $\forall x \in X, f(x^*) \leq f(x)$.

B.3 Problem types

B.3.1 Properties

Let us present different types of optimization problems according to properties of the problem features described in B.2. The interested reader can refer to `http://neos-guide.org/content/optimization-taxonomy` for a thorough taxonomy of optimization problems.

Definition of constraints An optimization problem is unconstrained when $m = 0$, and $\forall j \in \{1, \ldots, n\}, l_j = -\infty, u_j = +\infty$. An optimization problem is box-constrained when $m = 0$, and $\forall j \in \{1, \ldots, n\}, l_j \in \mathbb{R}, u_j \in \mathbb{R}$. In the general default case, an optimization problem is constrained.

Number of objectives An optimization problem is feasibility satisfaction problem when there is no objective function; in that case the objective is to find a feasible solution $x \in X$ or to prove the infeasibility of the problem $X = \emptyset$. The general default case deals with a single objective function $f : \mathbb{R}^n \to \mathbb{R}$. An optimization problem is multi-objective when $f = (f_1, \ldots, f_p) : \mathbb{R}^n \to \mathbb{R}^p$; one possibility is to use a lexicographic order on $\mathbb{R}^p$ to compare solutions; another possibility is to look for Pareto-optimal points, *i.e.*, points $x^* \in X$ such that $\nexists x' \in X : \forall k \in \{1, \ldots, p\}, f_k(x') < f_k(x^*)$.

Uncertainty of parameters In the general default case, an optimization problem is deterministic: parameters to obtain bounds, constraints, and objective functions are known given values. An optimization problem is robust when some of the parameters may vary in a given domain; in that case, we are usually interested in minimizing the worst-case – with respect to the parameters – objective function or satisfying the worst-case constraints. An optimization problem is stochastic when some of the parameters follow a probability law; in that case, we are usually interested in minimizing a probabilistic operator of the objective function – such as the expected value or the value-at-risk – or satisfying constraints in probability.

Expression of terms In the general default case, an optimization problem is explicit as the objective function and the constraints are expressed in closed form. An optimization problem is implicit in the converse case; implicit functions can be solutions of a parametrized subproblem as in bi-level programming for example; when the values of the implicit functions can only be obtained through an oracle, the problem is black-box.

Continuity of variables In the general default case, an optimization problem is continuous when all variables are continuous in $\mathbb{R}$. An optimization problem is discrete when the nature of the variables is mixed: continuous and discrete (integer or binary). A continuous relaxation of a discrete problem is a problem where the discrete variables have been replaced by continuous ones.

Linearity of constraints and objective An optimization problem is linear when the objective function and the constraints are linear combinations of variables.

Convexity of constraints and objective An optimization problem is convex when f and X are convex. Linearity and continuity imply convexity. Discontinuities imply non-convexity.

B.3.2 Problems of interest

In this thesis, we will deal with on Mixed Integer Linear Programming and Non-Linear Programming problems. We present in the section of few problems of interest in deterministic single-objective constrained optimization.

Linear programming (LP) Linear programming deals with problems where the variables are continuous, the objective function and the constraints are linear.

Non-linear programming (NLP) Non-linear programming deals with problems where the variables are continuous, the objective function and/or some of the constraints are non-linear.

Mixed-integer linear programming (MILP) Integer linear programming deals with problems where the variables are integer, the objective function and the constraints are linear. Mixed-integer linear programs also feature continuous variables.

Remark: combinatorial optimization deals with problem where admissible alternatives can be expressed as combinations of finite sub-alternatives; X being thus finite; admissible alternatives are not necessarily encoded with real-valued variables; however, most frequently, combinatorial optimization problems can be cast into ILP problems by representing the choices for sub-alternatives with binary variables.

Mixed-integer non-linear programming (MINLP) Mixed-integer non-linear programming deals with problems where the nature of the variables are is mixed, the objective function and/or some of the constraints are non-linear.

B.3.3 Complexity

LP problems can be theoretically solved in polynomial time. In the general case, ILP, NLP and MINLP can be said to be NP-hard or undecidable.

B.4 Algorithms and solutions

Once problems have been cast in the generic form (B.2.2), we are also interested in systematic methods – or algorithms – to provide solutions to these problems.

Let us present a few properties regarding algorithms and solutions.

B.4.1 Properties

Local/global Let $x \in X$ a feasible solution, and $N \subset X$ such that $x \in N$ a neighborhood of x. x is a local optimal solution if $\forall x' \in N, f(x) \leq f(x')$. By contrast, x is a global optimal solution if $\forall x' \in X, f(x) \leq f(x')$. This notion is usually relevant when variables are continuous; in that case N is usually taken as an open ball of arbitrary size. For convex problems, local optimal solutions are also global optimal solutions. Global optimization is the field of study for algorithms that solve non-convex problems with potentially multiple local optimums.

Specific/generic Solution algorithms work for given type of optimization problems. Generic algorithms work for all problem of a given type. Specific algorithms require additional properties and exploit a particular structure.

Solution quality An algorithm is exact when the solution x^* it provides at the end of the procedure is guaranteed to be optimal: $\nexists x' \in X : f(x') < f(x^*)$. An algorithm is approximate when the solution $\tilde{x}$ it provides at the end of the procedure is guaranteed to be close to the optimal value $f(x^*)$; the guarantee is absolute if $\exists \epsilon > 0 : f(\tilde{x}) \leq f(x^*) + \epsilon$; the guarantee is relative if $\exists \eta > 0 : f(\tilde{x}) \leq f(x^*)(1 + \eta)$. An algorithm is heuristic when there is no guarantee on the quality of the solution.

For example, halting an exact algorithm before its termination can be heuristic. The converse is also interesting: a heuristic can be considered as a potential approximate algorithm whose assumptions have not been unearthed to prove its guarantee.

Computational performance On top of theoretical complexity, algorithms are also evaluated based upon their computational performance: number of basic steps, number of iterations or CPU time for example. An algorithm can be evaluated through the time it takes to reach a given solution quality; an algorithm can also be evaluated by the solution quality reached after a limited number of iterations or CPU seconds.

Algorithms that have good time performances (and/or which require few memory) are said to be efficient.

When exact or approximate algorithms are inefficient for given problems, heuristics often appear as practical workarounds to provide satisfying solutions.

Deterministic/stochastic An algorithm is randomized or stochastic when some of its steps involve a random choice. An algorithm is deterministic otherwise. A randomized algorithm is evaluated by its probability to reach a given solution quality.

Numerical exactness The implementation of an algorithm is exact if it results in solutions that are arithmetically exact; otherwise solutions are accurate up to a given precision. Numerical exactness depends on the way rational numbers are represented: in floating-point precision or exactly. Note that, when working with computational implementations, we do not consider real numbers as encoding them with binaries is impossible/tricky as explained in [26].

Note that algorithm exactness is different from numerical exactness; the former is related to the quality of the solution theoretically obtained at the termination of an algorithm; the latter refers to the precision of the computational output with respect to the arithmetic calculation.

B.4.2 Algorithms related to Mixed-Integer Linear Programming

Simplex The simplex algorithm is a very efficient algorithm to solve LP. Its worst-case running time is exponential in the number of variables. It is essential in mathematical optimization as LP is one of most basic problem types which we can elaborate on. The feasible region of an LP is a polyhedron and the optimal value is attained at one of the vertices of that polyhedron.

Under some conditions, the algorithm is guaranteed to converge to the optimal value or to certify infeasibility of the problem.

Branch-and-bound Branch-and-bound (BB) is an implicit enumeration for MILP based on the solutions of continuous relaxations. The search is carried out through a binary tree which is constructed by branching, that is partitioning the search space.

The size of the binary tree is exponential in the number (and range) of variables; in the worst case, the whole tree must be explored; even if solving LP is considered as a basic operation, the worst-case complexity of the BB is exponential in the number of variables.

At termination, the BB algorithm provides a certificate of optimality or of infeasibility. If halted before termination, the BB algorithm may provide an intermediate feasible integer solution, whose quality can be assessed with the distance – or gap – to the lower bound.

Cutting planes The convex hull of all integer feasible solution to MILP problem is a polyhedron. The cutting planes (CP) method relies on the idea that solving a continuous relaxation of a MILP restricted to that polyhedron yields the optimal integer solution.

The scheme is to iteratively solve a continuous relaxation, find a cutting plane that separates the fractional solution obtained, solve the continuous relaxation strengthened by the cutting plane, and so forth until the solution is integer. The algorithm is guaranteed to converge to the optimal solution, if the separation problem can be solved exactly. When stopped before termination, the algorithm provides a lower bound but does not provide a feasible solution.

There are generic classes of CP as well as valid inequalities dedicated to specific problems.

Branch-and-cut The integration of CP to solve node subproblems in a BB scheme gives rise to the Branch-and-Cut (BC) algorithm. With general-purpose cuts, BC is the core algorithm of most MILP solvers nowadays as described in [50].

Dynamic programming Dynamic programming (DP) is a generic solution method for problems that exhibit time or stage separability property: if we know the state at a given stage, the optimal solution for the subsequent stages does not depend on the decisions taken at the previous stages.

Dynamic programming does not apply only to MILP and does not apply to all MILP.

Metaheuristics In [70], the following definition is given:

> A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.

Ant Colony Optimization, Genetic algorithms, Multi-Start methods, Simulated Annealing, Variable Neighborhood are examples of metaheuristics. The interested reader can refer to [30].

Several algorithmic approaches combine metaheuristics and mathematical programming algorithms as presented in the survey [61]; this resulting hybrid branch of algorithms is sometimes denoted matheuristics [51].

B.5 Computations and solvers

Algorithms for mathematical optimization can be coded into programs in order to compute solutions to problems. Generic optimization algorithms have been coded into software known as solvers.

In this section, we will discuss topics related to solvers and the computations of solutions for optimization problems; an implicit emphasis is given for MILP.

B.5.1 Use cases

Industry In the introduction of [37], the authors praise the merits of invoking solvers to solve MILP-formulated unit commitment problems.

Academia Solvers are also widely in academic works.

As mentioned in Section B.4.1, algorithms are evaluated by their computational performance which requires computational tests. Some solvers include most of the recent advances in optimization, represent the state-of-the-art and are used to benchmark computational performance of newly designed algorithms. Solvers which implement exact algorithms can also be used to obtain optimal solutions and assess the solution quality of newly designed algorithms.

Often, elaborate problems are solved with algorithms that invoke other optimization procedures to solve subproblems; as we have shown for the conventional algorithms presented in Section B.4.2. For computational tests, we can therefore use solvers for those subproblems instead of implementing algorithms that are already known and coded.

In addition, there are even solution methods that explicitly rely on the nested invocations of solvers. For example, several solution methods fall under the MIPping approach (see survey [28]), whose principle is to solve auxiliary MIP during the solution process of a master MIP. Note that Mixed Integer Linear Programming (MILP) is sometimes denoted MIP as linearity is implicit by default.

B.5.2 Advantages

Maturity and performance Depending on the control we want and the solver allows, it is sometimes better to use mature software that rely on state-of-the-art academic advances and efficient implementations instead of re-inventing the wheel.

The computational performance of solvers has consistently improved in the past thanks to progress in hardware and algorithms (see the first two figures of [44] for MILP solvers for instance).

Ease of use Solvers are easy to use as most are general-purpose: for a given problem type, we simply need to provide the model of the problem to solve in a generic form such as (B.2.2) together with data to instanciate parameters involved in the expression of bounds, constraints and objective function.

Solvers are also convenient as there are modelers, Application Programming Interfaces and callbacks that allow to manipulate the problem at hand, to control the algorithmic behavior of the solver and to retrieve all kind of information from the solution process.

For MILP, there are standard file formats compatible with most solvers that facilitate benchmarks between solvers for example.

Several solvers with decent performances are free and/or open-source; even commercial solvers are sometimes free for students and academics.

B.5.3 Solvers of interests

(MI)LP solvers The interested reader can refer to the survey [49].

CPLEX CPLEX (originally named for a C-implemented simPLEX) is a commercial MILP solver based on branch-and-cut. It features presolving techniques, search strategies and embedded heuristic techniques.

SCIP SCIP (Solving Constraint Integer Programs) is a non-commercial MILP solver that integrates constraint programming and satisfiability techniques in the usual MILP toolkit.

QSopt-Ex QSOPT-EX is an LP solver which provides exact rational solutions.

SCIP-Ex Most standard solvers work with finite precision binary floating-point arithmetic, which inherently introduces rounding errors. SCIP-EX is beta-version extension of SCIP which can solve MILP instances exactly over the rational numbers. SCIP-EX is used when exactness is needed for the solution or to check exactness of other solvers' solutions.

Convex NLP solvers

IPOPT IPOPT (Interior Point OPTimizer) is an open-source solver for convex large-scale non-linear optimization. It implements a primal-dual interior point algorithm, which uses a filter line search method to ensure global convergence.

Convex MINLP solvers

BONMIN BONMIN (Basic Open-source Nonlinear Mixed INteger programming) is an open-source solver for Convex Mixed-Integer Non-Linear Programs. It has been developed "within the framework of CBC by essentially replacing everything which was peculiar for the linear case with a non-linear counterpart (IPOPT) but keeping the structure as unchanged as possible" according to [50].

Global MINLP solvers

Couenne COUENNE (Convex Over and Under ENvelopes for Non-linear Estimation) is an open-source solver for non-convex MINLP that aims at finding global optima.

B.5.4 Challenges

For MILP, we make extensive reference to the article [44] that describes the library MIPLIB 2010 of instances that are challenging to solve for several benchmark general-purpose solvers; we will give details about what can be challenging when solving MILP in this section. Note that challenges might be concomitant, independent or correlated.

Performance Until theoretical boundaries are reached, computational performance is an ever-improving direction: we always want to faster compute better solutions to larger problems. Note that, on top of size, structure matters too. Indeed there are hard decently-sized problems, like the Traveling Salesman Problem (TSP) for which solvers have yet to prove their efficiencies; some problem instances require very long solution times, or even remain unsolved.

Solution time can be too long when LP solution time at each node of the BB tree is long and when the branches of the BB tree to explore are numerous.

Automatic reformulation When comparing the performance of two algorithms to solve a given problem, the most efficient one is likely to feature a procedure that is adapted to exploit a structural property of the problem. A given problem can be expressed in different equivalent formulations, each exhibiting different structural properties. The idea of automatic reformulation would be to translate a given formulation of a given problem to a formulation whose structural properties are inherently exploited by a given algorithm.

Performance variability Performance variability is another challenge described in [44]. It denotes "changes in performance measures for the same problem that are caused by seemingly performance-neutral changes in the environment or the input format". When conclusions do not depend on complexity theoretical argumentation but on empirical results, it become critical to take variability into consideration, at least by running experiments with test sets as large as possible along with showing variability statistics like standard deviation. In addition, "variability should be taken into account not only when studying performance, but also when studying the correctness of computation" as drastically different results from seemingly similar solution processes usually indicate numerical issues.

Appendix C

Data

C.1 Parameter values for the NLP HUC

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
1	2.66	50.17	0.53	42.37	1.53	48.06
2	2.66	40.17	0.53	29.75	1.53	48.06
3	2.66	35.17	0.53	30.09	1.53	47.56
4	2.66	35.17	0.53	30.12	1.53	47.00
5	2.66	35.15	0.53	30.20	1.53	47.55
6	2.66	40.05	0.53	30.29	1.53	47.73
7	2.66	57.17	0.53	52.40	1.53	54.20
8	2.66	75.17	0.53	60.24	1.53	75.00
9	2.66	90.00	0.53	93.60	1.53	105.00
10	2.66	147.00	0.53	140.11	1.53	110.61
11	2.66	146.94	0.53	148.11	1.53	110.63
12	2.66	139.95	0.53	141.11	1.53	100.61
13	2.66	95.00	0.53	82.61	1.53	78.61
14	2.66	90.19	0.53	77.61	1.53	77.71
15	2.66	95.00	0.53	90.61	1.53	94.91
16	2.66	95.36	0.53	103.61	1.53	95.63
17	2.66	90.61	0.53	107.60	1.53	188.11
18	2.66	75.49	0.53	97.60	1.53	199.13
19	2.66	60.39	0.53	73.61	1.53	199.13
20	2.66	80.50	0.53	62.61	1.53	110.63
21	2.66	95.62	0.53	61.61	1.53	79.63
22	2.66	65.25	0.53	75.45	1.53	61.62
23	2.51	60.25	0.52	61.25	1.53	58.49
24	2.36	55.05	0.51	59.15	1.52	52.85
25	2.21	50.24	0.50	41.75	1.52	48.60
26	2.06	40.16	0.49	30.24	1.52	48.49
27	1.91	35.15	0.48	30.14	1.52	48.49
28	1.91	35.07	0.48	30.12	1.52	48.38

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
85	2.19	57.99	0.66	84.77	3.06	71.69
86	2.19	61.75	0.66	92.72	3.06	57.64
87	2.19	61.69	0.66	99.75	3.06	55.62
88	2.19	56.24	0.66	103.72	3.06	55.62
89	2.19	56.24	0.66	99.75	3.06	71.62
90	2.19	51.70	0.66	93.68	3.06	96.89
91	2.19	55.63	0.66	76.65	3.06	96.92
92	2.19	64.64	0.66	63.71	3.06	97.00
93	2.19	70.00	0.66	68.75	3.06	96.97
94	2.19	61.00	0.66	76.75	3.06	74.92
95	2.34	57.12	0.67	68.62	2.84	67.67
96	2.48	56.02	0.67	55.67	2.62	50.62
97	2.63	40.50	0.68	60.61	2.39	48.62
98	2.78	40.50	0.68	50.61	2.17	47.63
99	2.93	35.30	0.69	45.61	1.95	47.54
100	2.93	35.31	0.69	30.61	1.95	47.00
101	2.93	30.25	0.69	30.61	1.95	47.00
102	2.93	30.26	0.69	30.61	1.95	47.43
103	2.93	40.25	0.69	41.61	1.95	49.62
104	2.93	48.25	0.69	56.34	1.95	75.82
105	2.93	48.25	0.69	76.61	1.95	92.85
106	2.93	60.56	0.69	92.50	1.95	109.51
107	2.93	60.71	0.69	92.41	1.95	109.02
108	2.93	60.86	0.69	79.41	1.95	93.93
109	2.93	60.50	0.69	61.49	1.95	76.00
110	2.93	47.45	0.69	58.17	1.95	75.92
111	2.93	47.40	0.69	58.00	1.95	76.45
112	2.93	47.45	0.69	58.75	1.95	99.92

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
29	1.91	35.09	0.48	29.75	1.52	48.38
30	1.91	40.16	0.48	29.75	1.52	48.62
31	1.91	57.06	0.48	52.40	1.52	54.62
32	1.91	65.23	0.48	60.13	1.52	75.62
33	1.91	100.61	0.48	93.00	1.52	105.63
34	1.91	147.61	0.48	132.00	1.52	110.63
35	1.91	147.62	0.48	140.00	1.52	105.63
36	1.91	130.61	0.48	133.00	1.52	100.62
37	1.91	95.61	0.48	81.93	1.52	75.63
38	1.91	80.59	0.48	77.14	1.52	75.63
39	1.91	95.61	0.48	90.22	1.52	95.62
40	1.91	95.60	0.48	110.00	1.52	95.63
41	1.91	90.60	0.48	110.00	1.52	185.15
42	1.91	85.60	0.48	96.77	1.52	199.27
43	1.91	70.60	0.48	72.75	1.52	199.27
44	1.91	70.61	0.48	61.75	1.52	110.64
45	1.91	95.61	0.48	74.75	1.52	75.65
46	1.91	80.59	0.48	60.93	1.52	75.63
47	1.82	60.48	0.48	60.88	1.74	70.62
48	1.73	55.29	0.48	58.69	1.96	53.63
49	1.64	55.73	0.48	50.75	2.18	48.63
50	1.56	44.75	0.48	30.60	2.40	48.61
51	1.47	39.75	0.48	30.33	2.63	48.61
52	1.47	34.75	0.48	30.25	2.63	48.61
53	1.47	39.68	0.48	37.25	2.63	48.60
54	1.47	50.59	0.48	40.25	2.63	47.79
55	1.47	52.73	0.48	52.04	2.63	59.55
56	1.47	62.00	0.48	61.10	2.63	75.68
57	1.47	96.00	0.48	88.63	2.63	95.91
58	1.47	145.50	0.48	101.64	2.63	105.97
59	1.47	145.49	0.48	140.60	2.63	105.90
60	1.47	145.00	0.48	121.64	2.63	95.87
61	1.47	94.49	0.48	84.40	2.63	71.82
62	1.47	85.29	0.48	89.40	2.63	59.60
63	1.47	89.34	0.48	90.61	2.63	59.57
64	1.47	85.49	0.48	104.63	2.63	64.50
65	1.47	88.89	0.48	100.64	2.63	74.78
66	1.47	79.70	0.48	94.62	2.63	95.80
67	1.47	61.61	0.48	76.40	2.63	99.89
68	1.47	83.75	0.48	68.40	2.63	95.80
69	1.47	92.75	0.48	64.60	2.63	77.00
70	1.47	80.07	0.48	76.40	2.63	69.64
71	1.61	63.95	0.52	68.40	2.71	59.67
72	1.76	61.24	0.55	56.61	2.80	48.61
73	1.90	62.60	0.59	50.62	2.89	52.62
74	2.05	52.65	0.62	29.61	2.97	50.53

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
113	2.93	51.45	0.69	59.75	1.95	184.03
114	2.93	60.40	0.69	59.75	1.95	199.14
115	2.93	60.40	0.69	59.75	1.95	199.14
116	2.93	60.45	0.69	59.75	1.95	120.44
117	2.93	79.71	0.69	70.11	1.95	80.47
118	2.93	69.89	0.69	70.11	1.95	72.43
119	2.63	60.45	0.71	56.75	1.92	59.64
120	2.34	57.45	0.73	50.00	1.88	49.70
121	2.05	40.25	0.75	55.75	1.85	47.70
122	1.76	40.11	0.77	30.48	1.82	47.55
123	1.47	35.22	0.80	30.50	1.79	47.00
124	1.47	30.25	0.80	30.49	1.79	47.00
125	1.47	30.25	0.80	30.50	1.79	47.00
126	1.47	35.29	0.80	30.60	1.79	47.58
127	1.47	40.25	0.80	30.50	1.79	53.69
128	1.47	47.45	0.80	30.50	1.79	75.25
129	1.47	48.26	0.80	50.61	1.79	118.00
130	1.47	60.08	0.80	60.60	1.79	138.10
131	1.47	60.62	0.80	60.50	1.79	118.50
132	1.47	60.78	0.80	60.50	1.79	100.29
133	1.47	60.61	0.80	60.49	1.79	78.25
134	1.47	47.87	0.80	50.61	1.79	78.07
135	1.47	47.45	0.80	50.40	1.79	95.50
136	1.47	48.05	0.80	50.23	1.79	95.63
137	1.47	51.62	0.80	50.20	1.79	184.94
138	1.47	60.08	0.80	50.20	1.79	199.13
139	1.47	60.40	0.80	50.25	1.79	198.50
140	1.47	60.66	0.80	50.40	1.79	100.63
141	1.47	79.86	0.80	60.25	1.79	79.35
142	1.47	70.00	0.80	70.21	1.79	74.89
143	1.58	60.89	0.74	60.00	1.67	69.63
144	1.70	61.00	0.68	59.84	1.56	52.75
145	1.81	40.40	0.62	53.00	1.44	48.00
146	1.93	40.07	0.57	30.25	1.33	47.68
147	2.04	35.07	0.51	30.25	1.22	47.50
148	2.04	29.63	0.51	29.75	1.22	45.51
149	2.04	29.68	0.51	29.40	1.22	46.87
150	2.04	40.07	0.51	29.4	1.22	48.02
151	2.04	56.75	0.51	30.24	1.22	54.61
152	2.04	70.16	0.51	59.40	1.22	75.49
153	2.04	90.47	0.51	94.75	1.22	118.62
154	2.04	147.43	0.51	110.00	1.22	118.62
155	2.04	147.28	0.51	110.06	1.22	118.62
156	2.04	140.21	0.51	100.45	1.22	100.62
157	2.04	75.20	0.51	87.77	1.22	78.02
158	2.04	80.00	0.51	87.88	1.22	78.62

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
75	2.19	36.65	0.66	29.62	3.06	47.64
76	2.19	33.63	0.66	29.62	3.06	47.62
77	2.19	30.64	0.66	36.51	3.06	47.56
78	2.19	32.61	0.66	39.62	3.06	47.56
79	2.19	52.68	0.66	51.65	3.06	47.67
80	2.19	57.75	0.66	64.61	3.06	50.09
81	2.19	59.39	0.66	87.68	3.06	49.44
82	2.19	69.99	0.66	100.81	3.06	71.66
83	2.19	70.00	0.66	139.82	3.06	74.00
84	2.19	65.00	0.66	109.88	3.06	72.59

	instance A1		instance B1		instance C1	
h	I_h	Π_h	I_h	Π_h	I_h	Π_h
159	2.04	90.28	0.51	92.00	1.22	95.62
160	2.04	95.27	0.51	92.11	1.22	95.62
161	2.04	90.25	0.51	92.18	1.22	185.12
162	2.04	75.10	0.51	82.00	1.22	199.12
163	2.04	74.79	0.51	81.75	1.22	199.12
164	2.04	84.89	0.51	60.75	1.22	100.62
165	2.04	109.89	0.51	74.77	1.22	79.62
166	2.04	70.04	0.51	65.45	1.22	75.62
167	2.04	55.19	0.51	55.07	1.22	70.00
168	2.04	54.67	0.51	51.00	1.22	53.07

Table C.1: Inflows I_h $(\mathrm{m}^3/\mathrm{s})$ and prices Π_h (currency$/\mathrm{MWh}$) for time periods $h \in \{1, \ldots, \bar{h} = 164\}$ for the 3 provided instances.

List of Tables

Bibliography

[1] Renewable energy cost analysis - hydropower. Technical Report Volume 1, Issue 3/5, IRENA, 2012. (Cited on page 18.)

[2] Russell L. Ackoff. *Redesigning the Future: A Systems Approach to Societal Problems*. John Wiley and Sons, 1974. (Cited on page 22.)

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - Theory, Algorithms and Applications*. Prentice-Hall, 1993. (Cited on page 26.)

[4] Jean-Christophe Alais. *Risque et optimisation pour le management d'énergies : application à l'hydraulique*. PhD thesis, École des Ponts ParisTech, 2013. (Cited on page 13.)

[5] Alicia Arce, Takaaki Ohishi, and Sérgio Soares. Optimal dispatch of generating units of the itaipu hydroelectric plant. *Power Systems, IEEE Transactions on*, 17(1):154–158, 2002. (Cited on pages 17 and 26.)

[6] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Foundations of Computer Science, 2005*, FOCS, pages 339–348. IEEE, 2005. (Cited on page 61.)

[7] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. (Cited on pages 61, 62, 63, and 65.)

[8] Richard Bellman and Stuart Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962. (Cited on page 26.)

[9] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611 – 614, 2013. (Cited on page 80.)

[10] Alberto Borghetti, Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. An MILP Approach for Short-Term Hydro Scheduling and Unit Commitment With Head-Dependent

Reservoir. *Power Systems, IEEE Transactions on*, 23(3):1115–1124, 2008. (Cited on page 27.)

[11] Alberto Borghetti, Claudia D'Ambrosio, Andrea Lodi, and Silvano Martello. Optimal scheduling of a multiunit hydro power station in a short-term planning horizon. In Katta G. Murty, editor, *Case Studies in Operations Research*, volume 212 of *International Series in Operations Research & Management Science*, pages 167–181. Springer New York, 2015. (Cited on pages 73 and 77.)

[12] Gerald G. Brown and Richard E. Rosenthal. Optimization tradecraft: Hard-won insights from real-world decision support. *Interfaces*, 38(5):356–366, 2008. (Cited on page 60.)

[13] Marc Brunet and Thomas Triboulet. How to track and deal with errors in huge data in entry of a model ? PGMO-COPI (Conference on Optimization and Practices in Industry, Programme Gaspard Monge pour l'Optimisation), 2014. (Cited on page 31.)

[14] Samuel Burer and Adam N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97 – 106, 2012. (Cited on page 13.)

[15] Valentina Cacchiani and Claudia D'Ambrosio. A Branch-and-Bound based Heuristic Algorithm for Convex Multi-Objective MINLPs. Technical Report OR-14-3, DEI, University of Bologna, 2014. (Cited on page 30.)

[16] Jean-Philippe Chancelier and Arnaud Renaud. Daily generation scheduling: decomposition methods to solve the hydraulic problems. *International Journal of Electrical Power & Energy Systems*, 16(3):175–181, 1994. Special Issue 11th Power Systems Computation Conference. (Cited on page 27.)

[17] Gary W. Chang, Mohamed Aganagic, James G. Waight, José Medina, Tony Burton, Steve Reeves, and Milton Christoforidis. Experiences with mixed integer linear programming based approaches on short-term hydro scheduling. *Power Systems, IEEE Transactions on*, 16(4):743–749, 2001. (Cited on page 14.)

[18] John W. Chinneck. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118. Springer Science & Business Media, 2007. (Cited on pages 13 and 60.)

[19] Guy Cohen. Optimization by decomposition and coordination: A unified approach. *IEEE Transactions on Automatic Control*, 23(2):222–232, 1978. (Cited on page 28.)

[20] COIN-OR. *Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT*, 2006. (Cited on page 77.)

[21] Antonio J. Conejo, Jose M. Arroyo, Javier Contreras, and Francisco A. Villamor. Self-scheduling of a hydro producer in a pool-based electricity market. *Power Systems, IEEE Transactions on*, 17(4):1265–1272, 2002. (Cited on page 27.)

[22] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, 2013. (Cited on pages 31 and 47.)

[23] Claudia D'Ambrosio and Andrea Lodi. Mixed integer nonlinear programming tools: a practical overview. *4OR*, 9(4):329–349, 2011. (Cited on page 13.)

[24] A. Kumar David. Competitive bidding in electricity supply. *Generation, Transmission and Distribution, IEEE Proceedings C*, 140(5):421–426, 1993. (Cited on pages 23 and 93.)

[25] Louis Dubost, Robert Gonzalez, and Claude Lemaréchal. A primal-proximal heuristic applied to the French Unit-commitment problem. *Mathematical Programming*, 104(1):129–151, 2005. (Cited on page 28.)

[26] Daniel G. Espinoza. *On Linear Programming, Integer Programming and Cutting Planes*. PhD thesis, Georgia Institute of Technology, 2006. (Cited on pages 31 and 100.)

[27] Stefan Feltenmark and Per Olof Lindberg. *Network Optimization*, volume 450 of *Lecture Notes in Economics and Mathematical Systems*, chapter Network Methods for Head-dependent Hydro Power Scheduling, pages 249–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. (Cited on page 26.)

[28] Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin. Just mip it! In Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, editors, *Matheuristics: Hybridizing metaheuristics and mathematical programming*, volume 10 of *Annals of Information Systems*, pages 39–70. Springer US, 2010. (Cited on page 102.)

[29] Antonio Frangioni. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005. (Cited on page 27.)

[30] Fred Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer Science & Business Media, 2003. (Cited on pages 87 and 101.)

[31] Nicolas Grebille. Numerical methods for stochastic multistage optimization problems applied to the european electricity market. SIAM Conference on Optimization, 2014. (Cited on page 13.)

[32] Olivier Guieu and John W. Chinneck. Analyzing infeasible mixed-integer and integer linear programs. *INFORMS J. on Computing*, 11(1):63–77, 1999. (Cited on pages 30 and 53.)

[33] Jimmie D. Guy. Security constrained unit commitment. *Power Apparatus and Systems, IEEE Transactions on*, PAS-90(3):1385–1390, 1971. (Cited on pages 23 and 93.)

[34] Grace Hechme-Doukopoulos, Sandrine Brignol-Charousset, Jérôme Malick, and Claude Lemaréchal. The short-term electricity production management problem at EDF. *Optima Newsletter-Mathematical Optimization Society*, 84:2–6, 2010. (Cited on pages 25, 27, and 28.)

[35] Michael Held, Philip Wolfe, and Harlan P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974. (Cited on page 28.)

[36] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, 2002. (Cited on page 46.)

[37] Benjamin F. Hobbs, Michael H. Rothkopf, Richard P. O'Neill, and Hung po Chao. *The Next Generation of Electric Power Unit Commitment Models*. International Series in Operations Research & Management Science. Springer US, 2001. (Cited on pages 29 and 101.)

[38] IBM. *ILOG CPLEX 12.2 User's Manual*. IBM, 2010. (Cited on pages 42, 53, and 77.)

[39] Leonid V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960. (Cited on page 13.)

[40] Stephen J. Kapurch. *NASA Systems Engineering Handbook*. DIANE Publishing, 2010. (Cited on page 59.)

[41] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004. (Cited on page 60.)

[42] Ed Klotz and Alexandra M Newman. Practical guidelines for solving difficult linear programs. *Surveys in Operations Research and Management Science*, 18(1):1–17, 2013. (Cited on page 31.)

[43] Ed Klotz and Alexandra M Newman. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(1):18–32, 2013. (Cited on page 60.)

[44] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. (Cited on pages 31, 46, 102, and 104.)

[45] John W. Labadie. Optimal operation of multireservoir systems: State-of-the-art review. *Journal of Water Resources Planning and Management*, 130(2):93–111, 2004. (Cited on pages 26 and 34.)

[46] Jon Lee, Janny Leung, and François Margot. Min-up/min-down polytopes. *Discrete Optimization*, 1(1):77–85, 2004. (Cited on pages 25 and 41.)

[47] Arnaud Lenoir. *Modèles et algorithmes pour la planification de la production à moyen terme en environnement incertain : Application de méthodes de décomposition proximales*. PhD thesis, Université Blaise Pascal - Clermont-Ferrand, 2008. (Cited on page 13.)

[48] Leo Liberti, Carlile Lavor, Nelson Maculan, and Antonio Mucherino. Euclidean distance geometry and applications. *SIAM Review*, 56(1):3–69, 2014. (Cited on page 73.)

[49] Jeffrey T. Linderoth and Andrea Lodi. *MILP Software*. John Wiley & Sons, Inc., 2010. (Cited on page 103.)

[50] Andrea Lodi. Mixed integer programming computation. 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*, pages 619–645. Springer Berlin Heidelberg, 2010. (Cited on pages 30, 31, 101, and 103.)

[51] Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, editors. *Matheuristics: Hybridizing metaheuristics and mathematical programming*, volume 10 of *Annals of Information Systems*. Springer US, 2010. (Cited on pages 66 and 101.)

[52] Harry M. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. (Cited on page 73.)

[53] Luca Mencarelli, Youcef Sahraoui, and Leo Liberti. A multiplicative weights update algorithm for MINLP. Forthcoming in EURO Journal on Computational Optimization, 2016. (Cited on pages 61 and 73.)

[54] Thomas J. Overbye, Jamie D. Weber, and Kollin J. Pattern. Analysis and visualization of market power in electric power systems. In *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, volume Track3, pages 10 pp.–, 1999. (Cited on pages 23 and 93.)

[55] Manfred Padberg. Approximating separable nonlinear functions via mixed zero-one programs. *Operations Research Letters*, 27(1):1–5, 2000. (Cited on page 40.)

[56] Narayana P. Padhy. Unit commitment-a bibliographical survey. *Power Systems, IEEE Transactions on*, 19(2):1196–1205, 2004. (Cited on page 26.)

[57] Ernesto Parrilla and Javier García-González. Improving the b&b search for large-scale hydrothermal weekly scheduling problems. *International Journal of Electrical Power & Energy Systems*, 28(5):339–348, 2006. (Cited on page 27.)

[58] Marian R. Piekutowski, Tadeusz Litwinowicz, and Roderick J. Frowd. Optimal short-term scheduling for a large-scale cascaded hydro system. In *Power Industry Computer Application Conference, 1993. Conference Proceedings*, pages 292–298, 1993. (Cited on page 55.)

[59] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. (Cited on page 61.)

[60] Marc Porcheron, Pascale Bendotti, and Nicolas Dupin. Planification des arrêts des réacteurs nucléaires d'EDF : extensions au challenge EURO/ROADEF 2010. Congrès annuel de la ROADEF, 2012. (Cited on page 13.)

[61] Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In José Mira and José R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer Berlin Heidelberg, 2005. (Cited on pages 66 and 101.)

[62] Emmanuel Rachelson, Ala Ben Abbes, and Sebastien Diemer. Combining mixed integer programming and supervised learning for fast re-planning. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 2*, pages 63–70, 2010. (Cited on page 22.)

[63] Deepak Rajan and Samer Takriti. Minimum up/down polytopes of the unit commitment problem with start-up costs. Technical Report RC23628, IBM Research Division, 2005. (Cited on pages 25 and 41.)

[64] Arnaud Renaud. Daily generation management at Electricite de France: from planning towards real time. *Automatic Control, IEEE Transactions on*, 38(7):1080–1093, 1993. (Cited on pages 27 and 28.)

[65] Mark J. Rentmeesters, Wei K. Tsai, and Kwei-Jay Lin. A theory of lexicographic multi-criteria optimization. In *Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on*, pages 76–79, 1996. (Cited on page 57.)

[66] Fabio Roda. *Integrating high-level requirements in optimization problems: theory and applications*. PhD thesis, École Polytechnique, 2013. (Cited on page 59.)

[67] Claudia Sagastizábal. Divide to conquer: decomposition methods for energy optimization. *Mathematical Programming*, 134(1):187–222, 2012. (Cited on page 28.)

[68] Mohammad Shahidehpour. Tools for generation risk management. In *Power Engineering Society Summer Meeting, 2001*, volume 1, pages 414–416 vol.1, 2001. (Cited on pages 23 and 93.)

[69] Gerald B. Sheble and George N. Fahd. Unit commitment literature synopsis. *Power Systems, IEEE Transactions on*, 9(1):128–135, 1994. (Cited on page 26.)

[70] Kenneth Sörensen. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015. (Cited on pages 65, 87, and 101.)

[71] Daniel E. Steffy. *Topics in exact precision mathematical programming*. PhD thesis, Georgia Institute of Technology, 2011. (Cited on page 13.)

[72] Milad Tahanan, Wim van Ackooij, Antonio Frangioni, and Fabrizio Lacalandra. Large-scale unit commitment under uncertainty. *4OR*, 13(2):115–171, 2015. (Cited on pages 26 and 27.)

[73] Raouia Taktak and Claudia D'Ambrosio. An overview on mathematical programming approaches for the deterministic unit commitment problem in hydro valleys. *Energy Systems*, pages 1–23, 2016. (Cited on pages 26 and 34.)

[74] Raouia Taktak, Claudia D'Ambrosio, and Sonia Toubaline. Problème de gestion de production électrique à court-terme dans les vallées hydrauliques. Congrès annuel de la ROADEF, 2016. (Cited on page 60.)

[75] Chung-Li Tseng. *On Power System Generation Unit Commitment Problems*. PhD thesis, University of California, Berkeley, 1996. (Cited on page 28.)

[76] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958. (Cited on page 21.)

[77] Hermann-Josef Wagner and Jyotirmay Mathur. *Introduction to Hydro Energy Systems - Basics, Technology and Operation*. Green Energy and Technology. Springer-Verlag Berlin Heidelberg, 2011. (Cited on pages 18 and 91.)

[78] William W-G. Yeh. Reservoir management and operations models: A state-of-the-art review. *Water Resources Research*, 21(12):1797–1818, 1985. (Cited on page 34.)

[79] Qiaozhu Zhai, Xiaohong Guan, and Feng Gao. A necessary and sufficient condition for obtaining feasible solution to hydro power scheduling with multiple operating zones. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–7, 2007. (Cited on page 30.)

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Planification de la production hydroélectrique au court terme : faisabilité et modélisation
Mots clés : programmation mixte en nombres entiers, programmation non-linéaire, planification de la production hydroélectrique, infaisabilité, calculs numériques exacts, matheuristiques.

Résumé : Dans le secteur électrique, et en particulier chez le fournisseur et producteur EDF, l'optimisation mathématique est utilisée pour modéliser et résoudre des problèmes liés à la gestion de la production d'électricité.

Nous nous intéressons à des problèmes d'optimisation de Programmation Non Linéaire en Nombres Entiers (PNLNE) liés à la planification de la production hydroélectrique au court terme, Short-Term Hydropower production Scheduling (STHS) en Anglais. En effet, exploiter l'hydroélectricité constitue un enjeu important pour la gestion de la production électrique car c'est une source d'énergie renouvelable, peu chère, flexible mais limitée. Etant données des ressources d'eau finies dans les barrages, l'objet du STHS est de prescrire des programmes de production les plus rentables qui soient compatibles avec les spécifications techniques des usines hydroélectriques.

Les problèmes de PNLNE sont particulièrement difficiles à résoudre. En amont des performances de résolution, la faisabilité est une question préliminaire à aborder puisqu'il faut s'assurer que les PNLNE à résoudre admettent des solutions. De plus, la résolution de PNLNE est rendue encore plus difficile si l'on requiert une précision exacte des résultats car il faut alors se prémunir d'erreurs numériques.

Dans un premier temps, nous traitons de la faisabilité d'un problème réel de STHS formulé en PLNE. Les complications liées aux données réelles et au calcul numérique, associées aux caractéristiques spécifiques du modèle, rendent le problème plus difficile à résoudre et souvent infaisable. Nous procédons par étapes pour identifier et traiter les sources d'infaisabilité une par une, à savoir les erreurs numériques et les infaisabilités de modélisation. Des résultats informatiques étayent l'efficacité de notre méthode sur un ensemble de test de 66 instances réelles qui contenait initialement de nombreuses infaisabilités.

Dans un deuxième temps, nous adaptons l'algorithme du Multiplicative Weights Update (MWU) pour la PNLNE. Cette adaptation est fondée sur une reformulation paramétrée que l'on dénomme pointwise. Nous définissons des propriétés souhaitables pour obtenir de bonnes reformulations pointwise et nous fournissons des règles de haut niveau pour adapter l'algorithme étape par étape. Contrairement à la plupart des heuristiques, nous démontrons que la méthode MWU conserve une garantie d'approximation relative pour les PNLNE. Nous mettons en oeuvre le MWU pour résoudre un STHS formulé en PNL en tant que preuve de concept.

Title : Short-term hydropower production scheduling: feasibility and modeling
Keywords : mixed-integer programming, non-linear programming, hydro unit commitment, short-term hydropower production scheduling, infeasibility, exact computation, matheuristics.

Abstract : In the electricity industry, and more specifically at the French utility company EDF, mathematical optimization is used to model and solve problems related to electricity production management. We are interested in Mixed-Integer Non-Linear Programming (MINLP) optimization problems related to Short-Term Hydropower production Scheduling (STHS). Indeed, harnessing hydropower is critical for electricity production management as it is a renewable, cheap, flexible but limited source of energy. Given finite resources of water in reservoirs, the purpose of STHS is to prescribe production schedules with largest payoffs that are compatible with technical specifications of the hydroelectric plants.

MINLP problems are particularly difficult to solve. Prior to solution performance, feasibility is a preliminary challenge to tackle since we should ensure the MINLP problems to solve admit feasible solutions. Also, certifying the exactness of the results compounds the difficulty of solving MINLP problems as numerical errors may occur.

Firstly, we deal with feasibility issues of a real-world STHS MILP problem. Issues affecting real-world data and numerical computing, together with specific model features, make the problem harder to solve and often infeasible. We follow a step-by-step approach to systematically exhibit and repair one source of infeasibility at a time, namely numerical errors and model infeasibilities. Computational results show the effectiveness of the approach on a test set of 66 real-world instances that demonstrated a high occurrence of infeasibilities.

Secondly, we adapt the Multiplicative Weights Update (MWU) algorithm to solve MINLP. Such adaptation is based on a parametrized reformulation denoted pointwise. We define desirable properties for deriving pointwise reformulations and provide elaborate high-level guidelines to adapt the algorithm step-by-step. Unlike most heuristics, we show that the MWU method still retains a relative approximation guarantee. To deliver a proof of its applicability, we implement the method to solve a hard NLP STHS problem.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France