Solving large-scale unit-commitment problems using dual dynamic programming and open-source solvers

Bruno Colonetti[a,*], Samuel Souza Brito[b], Erlon Finardi[a,c], Victor Zavala[d], Lucas Borges Picarelli[e]

[a]*Federal University of Santa Catarina, Brazil.*
[b]*Departamento de Computação e Sistemas, Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, Brazil.*
[c]*INESC P&D Brasil, Santos 11055-300, Brazil.*
[d]*Department of Chemical and Biological Engineering, University of Wisconsin-Madison, Madison, WI.*
[e]*Norte Energia S.A., Brasilia 70390-025, Brazil.*

Abstract

The astonishing dimensions and complexity of power systems render them impossible to be managed without the help of cutting-edge software. Due to a lack of scalable, reliable and well documented free and open-source solutions, system operators, regulators, and government agencies often rely on proprietary software to provide them information that ultimately will be used to serve energy to millions of people and shape the economy. However, proprietary software is frequently closed source, meaning that its users cannot generally see its source code. Consequently, the crucial step of computational-aided analysis can be secretive to the general public, allowing mistrust and potentially preventing valuable information from being incorporated into the decision-making process. In particular, a problem that has been dominated by proprietary software is the unit commitment (UC). Worldwide, system operators use proprietary software to daily solve their UCs and define the best scheduling of the generating units under their supervision to satisfy the forecast load. So far, attempts to solve UC instances with open-source solutions have been mainly restricted to small cases. Here, we propose a flexible decomposition framework based on dual dynamic integer programming that allows us to solve realistic instances of the UC in reasonable time. We are particularly interested in hydrothermal systems, and, for one such system with 7,475 buses, we show through exhaustive numerical experiments that our approach is a promising alternative to proprietary software. For this system, we can solve the UC with a completely free and open-source package in less than one hour, both with an asynchronous and a synchronous strategy.

Keywords: unit commitment, dual dynamic programming, Benders decomposition, free solver, open source.

Nomenclature

Indices and Sets

$\mathcal{L}_b^+, \mathcal{L}_b^-$ Lines with positive flow in (+) and out (−) of bus b

$\underline{\mathcal{G}}_i, \overline{\mathcal{G}}_i$ Thermal units ($\mathcal{G}$) participating in the combined generation limit i

$\underline{\mathcal{I}}_t^{\mathrm{g}}, \overline{\mathcal{I}}_t^{\mathrm{g}}$ Upper (overline) and lower (underline) bounds on the combined generation of groups of thermal units

$b \in \mathcal{B}, g \in \mathcal{G}, h \in \mathcal{H}, l \in \mathcal{L}$ Buses, thermals units, hydro plants, transmission lines

$j \in \mathcal{A}$ Linear inequalities in the cost-to-go function

$j \in \mathcal{F}_h$ Linear inequalities of hydro plant h's approximation of its hydropower function

$t \in \mathcal{T}, \mathcal{T} = \{1, ..., \mathrm{T}\}$ Time periods in the planning horizon

Constants

$\mathrm{A}_j^{\mathrm{coeff}}, \mathrm{A}_j^{\mathrm{coeff}}$ Coefficient and constant of the cost-to-go function's j inequality (\$/hm^3, \$)

C^{h} Flow-to-volume conversion ($1.8 \cdot 10^{-3}$ hm$^3 \cdot$ s/m^3)

$\mathrm{C}_h^{\mathrm{pump}}$ Pumping rate in $(p.u./(m^3/s))$

$\mathrm{F}^{\mathrm{vol}}, \mathrm{F}^{\mathrm{turb}}, \mathrm{F}^{\mathrm{spil}}, \mathrm{F}^{\mathrm{const}}$ Coefficients of the hydropower function's approximation: volume, turbine discharge, spillage, and constant (p.u./hm^3, p.u./(m^3/s), p.u./(m^3/s), p.u.)

I Incremental inflow (m^3/s)

$\mathrm{L}_{b,t}$ Bus load (p.u.)

$\mathrm{SD}_{g,i}^{\mathrm{step}}$ ith step in the start-up trajectory. From 0 to $\mathrm{SD}_g - 1$

$\mathrm{SU}_g, \mathrm{SD}_g$ Periods in the start-up and shut-down trajectories

$\mathrm{SU}_{g,i}^{\mathrm{step}}$ ith step in the start-up trajectory. From 0 to $\mathrm{SU}_g - 1$

*Corresponding author.
Email addresses: colonetti.bruno@posgrad.ufsc.br (Bruno Colonetti), samuelbrito@ufop.edu.br (Samuel Souza Brito), erlon.finardi@ufsc.br (Erlon Finardi), zavalatejeda@wisc.edu (Victor Zavala), lucaspicarelli@norteenergiasa.com.br (Lucas Borges Picarelli)

T_g^{up}, T_g^{down} Minimum up and down times

$\overline{P}_h^h$ Upper bounds on hydro generation (p.u.)

$\overline{Q}_h^{pump}, \overline{Q}_h^{bypass}$ Upper bounds on pumping and bypassing in (m^3/s)

$\underline{F}, \overline{F}, X$ Bounds on the flows in transmission lines, and line reactance (p.u., p.u., p.u./rad)

$\underline{P}_g^t, \overline{P}_g^t$ Bounds on thermal generation (p.u.)

$\underline{R}_g, \overline{R}_g$ Ramping limits (p.u./(30 min))

$\underline{V}_h, \overline{V}_h$ Bounds on reservoir volume in (hm^3)

Hydro variables

α Estimate future cost ($)

p^h, d^h Generation (p.u.), Status of the power plant (0/1)

$p_{h,j,t}^{hpf}$ Generation output associated with plane j of plant h in time t (p.u.)

$v, q, q^{pump}, q^{bypass}, s$ Reservoir volume, turbine discharge, pumping discharge, water bypass, and spillage (hm^3, m^3/s, m^3/s)

v^{out}, v^{in} Water outflow and water inflow (hm^3, hm^3)

$v^{s,\,out}, v^{s,\,in}$ Slacks for the water balance (hm^3, hm^3)

$z_{h,j,t}^{hpf}$ Binary variable associated with plane j of plant h in time t (0/1)

Network variables

θ, f Bus angles and line flows (rad, p.u.)

$p^{s,\,n,\,+}, p^{s,\,n,\,-}$ Slacks variables: virtual generation, virtual load (p.u., p.u.)

Thermal variables

d^t Status of the dispatch phase (0/1)

$p^t, p^{t,\,disp}$ Total generation (p.u.), and generation in dispatch phase (p.u.)

y, x Start-up and shut-down decision (0/1)

1. Introduction

In any country, power systems are to be operated as economically, reliably and securely as possible. However, owing to their unparalleled complexities, it is humanly impossible to properly manage such systems without the help of computers. Therefore, mathematical models are at the center of energy management, being used from long-term expansion planning to day-ahead and near-real-time operation. One of the most intensively researched models of this chain is known as unit commitment (UC). In centrally dispatched, cost-based pool markets, this problem is used in the decision-making process of determining the status (ON/OFF) and generation level of generating units to meet the day-ahead electrical-energy demand while minimizing costs and satisfying system-wide and plant-based constraints.

In practice, UC is formulated as a deterministic mixed-integer linear programming (MILP) problem, and, despite extensively studied, it continues to be an open problem. According to [35], the first literature records of solution methods developed to UC date back to the 60s and consist of exhaustive explicit enumeration. With the development of powerful computing machines and advancements in mathematical programming, decomposition approaches dominated the literature and were applied in practice across the world. An example is EdF's (Électricité de France, the French Electricity Board) Lagrangian-relaxation-based method [14]. More recently, however, the evolution of general-purpose commercial solvers primarily based on branch-and-cut enabled system operators to solve commitment problems on a black-box framework with nearly no need for in-house solutions. (For instance, the transition from Lagrangian-based solution to a commercial solver was made in 2005 and 2014 at the PJM Interconnection [30] and the New York Independent System Operator [29], respectively.) Regardless of the solution process, the computational tools used by system operators share at least one important characteristic: they are proprietary software [19]. As it is mainly the case with such packaged solutions, their source codes are not open and they effectively work as black boxes. Thus, auditing, modifying, or improving these software is not generally possible other than by the owner or some delegated third party.

In Brazil, for instance, the organization responsible for developing the models and solution methods used in the planning process is the Center for Energy Research (CEPEL) — a public company that is part of the largest energy company in Brazil, Eletrobras (Centrais Elétricas Brasileiras S.A.). Today, CEPEL is responsible for developing and maintaining the three main optimization software used for planning the operation of the Brazilian system: the medium-term model (`NEWAVE`), the short-term model (`DECOMP`), and the day-ahead model (`DESSEM`). All three are proprietary, closed source, commercial software. Thus, agents cannot independently verify their correctness instead having to rely on input-output analysis. Also, any customization that one particular agent might need to fit their purposes is not possible. In addition, as pointed by [31], the closedness of such computational tools can be a source of distrust among the public, who, one could argue, is the most interest part in energy management. Still according to [31], among the reasons for software's closedness are "business confidentiality, concerns over the security of critical infrastructure, a desire to avoid exposure and scrutiny, worries about data being misrepresented or taken out of context, and a lack of time and resources". More specific to the research community, [33] asserts that, without source codes, research results are often irreproducible and building up on previous published work by other authors is generally not possible due to lack of information.

Faced with the aforementioned concerns, a growing number of academic works has been dedicated to developing free,

open-source tools to energy management [19]. The trend seems to be extending to the industry itself, where initiatives like The Global Power System Transformation Consortium, an organization funded and founded by system operators, has one of its main goals defined as to "Support rigorous planning, operational analysis and enhanced real-time system monitoring through open data and tools" [17].

However, for the UC, most published works either still rely on commercial solvers or are developed only for educational purposes. In the extensive review [19], 11 free, open-source tools are listed as solutions to UC and/or security-constrained UC problems. Although a significant number, the 11 tools all present limitations that prevent them from being used on realistic UCs. Firstly, some of them do not currently provide support for hydro modelling (`Calliope` [32], `ficus` [3], `minpower` [18], `psst` [27]), which is an integral part of hydrothermal systems, such as Brazil's. Others, despite being mostly open source, still rely on commercial components — like `GAMS` [11] (a general modelling tool), used in `Dispa-SET` [24], `EMMA` [22], and `TIMES` [28]. In our opinion, however, the most important shortcoming impeding the practical usage of these solutions with free solvers in a completely free and open-source tool is the unavailability of decomposition approaches (mathematical techniques that break the large original problem in several smaller subproblems that individually are easier to solve). In addition to the previous packages, `MOST` [38], `PyPSA` [6], and `Temoa` [23] are among the solutions that, despite supporting hydro modelling and being completely free and open source, do not employ decomposition. The same is true for the open-source Python package `EGRET` [26], used for assessing UC formulations, and `OATS` [7], and `oemof` [21]: although they are freely available and capable of working with free solvers, they do not currently employ any decomposition scheme.

As we will show through exhaustive testing, decomposition is a fundamental part of a free, open-source package that is intended for solving large-scale UCs. In contrast to the previously mentioned works, researchers have also developed open-source frameworks that exploit decomposition but nonetheless still depend on commercial solvers for solving computationally intensive subproblems. In [25], for instance, the authors tackle large-scale UCs that appear in a long-term analysis by dualizing the time-coupling constraints through Lagrangian relaxation. Despite using open-source packages for parallelization and for generating branch-and-bound trees, they still rely on a commercial solver for the solution of MILP subproblems and the quadratic master dual problem. Another example is [37], where the authors break the original security-constrained UC by relaxing space-coupling constraints and solve the resulting problem using the alternating-direction method of multipliers (ADMM). In this case, the subproblems also solved with the commercial solver `Cplex` [12].

The above bibliography review on open-source packages for UC does not reflect the dominance of commercial solvers in this research area. To illustrate this dominance, we first show in Tab. 1 optimization solvers generally found in the energy-management literature. In this table, the proprietary software are those that commercial users have to paid for. An important exception is `SCIP` (Solving Constraint Integer Programs), which is free for academics under license [40], but this license does not apply for non-academic users. Apart from `SCIP`, all other free software in Tab. 1 are licensed under Eclipse Public License - v 2.0 [15], which permits any user, academic or otherwise, to use, modify, improve, or integrate these software into commercial packages. From the software in this table, it is clear that those licensed under [15] offer the most flexibility in terms of commercial usage.

Table 1: Commercial and academic solvers commonly used in energy management. Column "Problem types" indicates whether the solver can handle MILP and/or linear programming problems (LP) (currently the most important formulations for UC). Although `Ipopt` is mostly used for nonlinear optimization, it can also handle LPs through its interior-point algorithm. Although `Clp` only handles LPs, as `Ipopt`, it can be embedded in a branch-and-cut algorithm. In this table "P" stands for proprietary, "C" for closed and "O" for open.

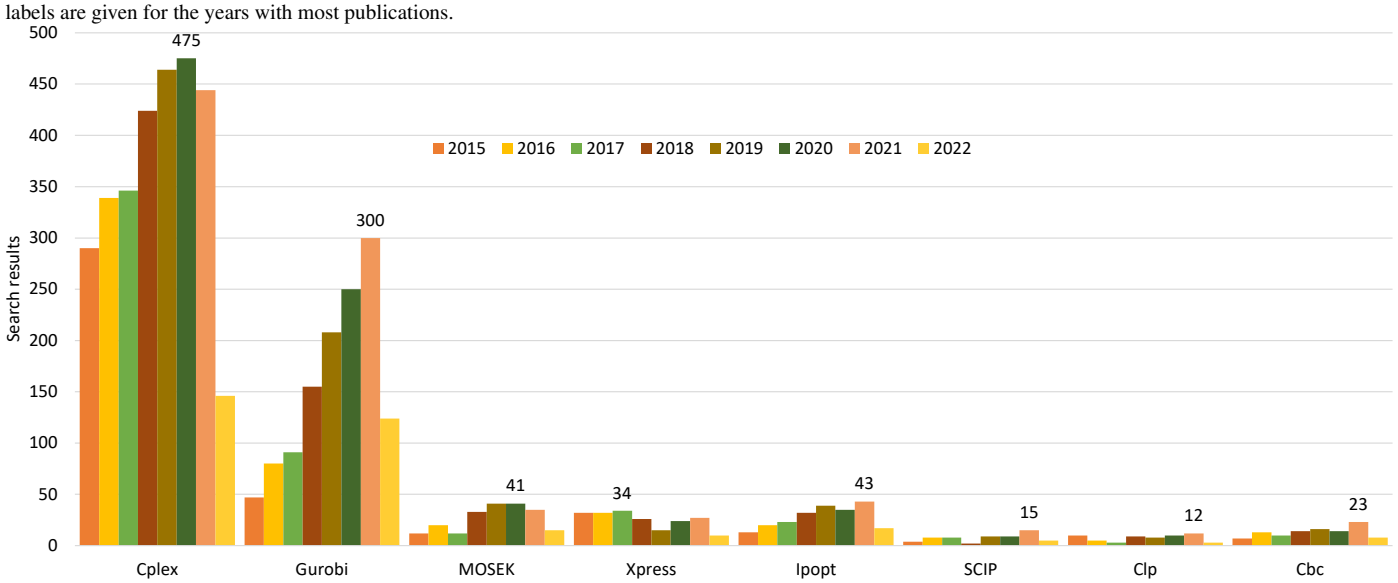
Solver	Problem types	License	Source	Reference
`Gurobi`	MILP/LP	P	C	[20]
`Cplex`	MILP/LP	P	C	[12]
`Xpress`	MILP/LP	P	C	[16]
`MOSEK`	MILP/LP	P	C	[1]
`SCIP`	MILP/LP	[40]*	O	[5]
`Cbc`	MILP/LP	[15]	O	[8]
`Clp`	LP	[15]	O	[9]
`Ipopt`	LP	[15]	O	[36]

*Only valid for academics.

Because the leading optimization companies generally grant academics free access to their commercial solvers for research purposes, they dominate the energy-management literature. A clear advantage of having access to powerful commercial solvers is that academics can focus on aspects of the problem other than solution methods, such as analysis and modelling. The downside is that free, open-source solvers are not developed, ultimately leaving the industry no option but to use commercial solvers. To indirectly quantify this surmise that commercial solvers are generally preferred in academic works, we provide in Fig. 1 the number of publications in each year from 2015 to 2022 that include the expression "unit commitment" and the solvers from Tab. 1. Evidently, this is only meant to illustrate the disparate between commercial and free solvers, and a thorough review on the subject would be necessary to give more accurate numbers.

As Fig. 1 suggests, `Cplex` and `Gurobi` dominate the literature as they are mentioned in about 83% of the total search results over the eight years, with `Cplex` well ahead with 58% compared to `Gurobi`'s 25%. The commercial solver that seems to draw the least attention according to Fig. 1 is FICO's `Xpress` which appears in 4% of the results. Nonetheless, `Xpress` is still ahead of the first non-commercial solver, `Ipopt`, and it appears in a relative number of works that is only 0.5% (or 25 search results) less than the combined three trailing solvers — `SCIP`, `Clp` and `Cbc`. Therefore, despite the recent efforts of the research community, commercial solvers are still used in most published academic workers, which underlines the need for more efficient, free, and open-source solutions.

Figure 1: Search results for selected optimization solvers. The term searched was '"unit commitment" "solver"', where solver can be either "cplex", "gurobi", "mosek", "xpress", "ipopt", "scip", "cbc", or "clp". The websites used were https://ieeexplore.ieee.org/ (IEEE), https://www.sciencedirect.com/ (Elsevier), https://link.springer.com/ (Springer), and https://onlinelibrary.wiley.com/ (Wiley) — in order of total results. The searches were conducted in March, 2022. Data labels are given for the years with most publications.



Considering the lack of applications of free and open-source solvers to large-scale UC, our main contribution is the development of a decomposition strategy that enables the efficient solution of a large-scale hydrothermal UC with `Cbc`. Additionally, we extend the algorithm originally developed in [10] to include inner decompositions of subhorizon problems, and also adapt it to include a synchronous approach. As opposed to the asynchronous strategy of [10], our synchronous strategy, under certain conditions, is deterministic: it returns exactly the same result every time it is run. This feature is appealing specially to industrial applications, where there might be concerns about the reproducibility of the results used to commit and dispatch the generation resources.

Our work is organized as follows: we present the problem in Section 2, and our solution algorithms in Section 3; we introduce the test system and instances of the unit-commitment problem used for assessing our proposed strategies in Section 4; our results are shown and discussed in Section 5; finally, in Section 6, we give our concluding remarks and list possible future research avenues.

2. Problem statement

We address the UC in a centrally dispatched system in a cost-based pool. In this context, the system operator determines the day-ahead commitment and generation levels of the dispatchable generating units to meet the load at minimum cost, while also satisfying several operational constraints that define that feasibility (at least in terms of the mathematical model) of the decision. We are particularly interested in hydrothermal systems. Thus, in this section, we define a hydrothermal UC, although a purely thermal UC is, by definition, also implicitly defined, and extensions to include other components are straightforward. For clarity and reference later, we individually present the main components of the model.

2.1. Thermal generation

This subsection presents the mathematical model for the thermal units.

2.1.1. Binary decisions

Here, we show the components of the thermal model that depend exclusively on binary variables. At any given time, a thermal unit can be in one of four states: off, in its start-up trajectory, in its shut-down trajectory, or in the dispatch phase. To model these states and the transitions between them, our model has three binary variables for each thermal unit — a start-up decision, a shut-down, and a dispatch-phase status. In (1), for an arbitrary period t, we have the logical constraints that ensure changes in the dispatch-phase status are results of start-up and shut-down decisions.

$$y_{g,t-\mathrm{SU}_g} - x_{g,t} - d^{\mathrm{t}}_{g,t} + d^{\mathrm{t}}_{g,t-1} = 0 \quad \forall g \in \mathcal{G}. \tag{1}$$

The following two sets of constraints (2) and (3) are, respectively, the classical minimum up and down times. They guarantee that, once turned on (off), thermal units must remain on (off) for a pre-defined amount of time to avoid damaging mechanical stress.

$$\sum_{i=t-\mathrm{T}^{\mathrm{up}}_g}^{t} y_{g,i} \leq d^{\mathrm{t}}_{g,t} \quad \forall g \in \mathcal{G}. \tag{2}$$

$$\sum_{i=t-\mathrm{T}^{\mathrm{down}}_g}^{t} x_{g,i} \leq (1 - d^{\mathrm{t}}_{g,t}) \quad \forall g \in \mathcal{G}. \tag{3}$$

2.1.2. Continuous decisions

In this part of the thermal model, we deal with the actual power generation. In (4), we limit it to be within the minimum and maximum limits, whereas in (5), variations of generation in consecutive periods are constraint by the units' ramping capabilities. Here, we model the generation in the dispatch phase as the additional generation above the minimum. Thus, if a unit is in the dispatch phase but its generation is at the minimum, variable $p^{\mathrm{t,disp}}_{g,t}$ equals zero. Evidently, the minimum generation is accounted for in (8).

$$0 \leq p^{\mathrm{t,disp}}_{g,t} \leq \left(\overline{\mathrm{P}}^{\mathrm{t}}_{g} - \underline{\mathrm{P}}^{\mathrm{t}}_{g}\right) \cdot d^{\mathrm{t}}_{g,t} \quad \forall g \in \mathcal{G}. \tag{4}$$

$$-\underline{\mathrm{R}}_{g} \leq p^{\mathrm{t,disp}}_{g,t} - p^{\mathrm{t,disp}}_{g,t-1} \leq \overline{\mathrm{R}}_{g} \quad \forall g \in \mathcal{G}. \tag{5}$$

When entering or leaving the dispatch phase (being started up or shut down), the generation output of a thermal unit must be equal to a pre-defined level. We assume that such level is the unit's minimum generation, and show in (6) and (7) the constraints that guarantee that in the first period in the dispatch phase after being started up and the last period in the same phase before being shut down, the generation of the thermal units must be equal to their respective minimums.

$$p^{\mathrm{t,disp}}_{g,t} \leq \left(\overline{\mathrm{P}}^{\mathrm{t}}_{g} - \underline{\mathrm{P}}^{\mathrm{t}}_{g}\right) \cdot \left(d^{\mathrm{t}}_{g,t} - y_{g,t-\mathrm{SU}_g}\right) \quad \forall g \in \mathcal{G}. \tag{6}$$

$$p^{\mathrm{t,disp}}_{g,t-1} \leq \left(\overline{\mathrm{P}}^{\mathrm{t}}_{g} - \underline{\mathrm{P}}^{\mathrm{t}}_{g}\right) \cdot \left(d^{\mathrm{t}}_{g,t-1} - x_{g,t}\right) \quad \forall g \in \mathcal{G}. \tag{7}$$

In (8), we show the unit's total generation. This equation accounts for generation in both trajectories and in the dispatch phase. The start-up trajectory is represented in (8) by term $\sum_{i=0}^{\mathrm{SU}_g-1}(\mathrm{SU}^{\mathrm{step}}_{g,i} \cdot y_{g,t-i})$, which is evidently dependent on the start-up decision. On the other hand, the shut-down trajectory is accounted for by $\sum_{i=0}^{\mathrm{SD}_g-1}(\mathrm{SD}^{\mathrm{step}}_{g,i} \cdot x_{g,t-i})$. Note that the term $\underline{\mathrm{P}}^{\mathrm{t}}_{g}$ is added since $p^{\mathrm{t,disp}}_{g,t}$ only represents the additional generation above the minimum.

$$p^{\mathrm{t}}_{g,t} - p^{\mathrm{t,disp}}_{g,t} - \underline{\mathrm{P}}^{\mathrm{t}}_{g} \cdot d^{\mathrm{t}}_{g,t} - \sum_{i=0}^{\mathrm{SU}_g-1} (\mathrm{SU}^{\mathrm{step}}_{g,i} \cdot y_{g,t-i}) - \sum_{i=0}^{\mathrm{SD}_g-1} (\mathrm{SD}^{\mathrm{step}}_{g,i} \cdot x_{g,t-i}) = 0 \quad \forall g \in \mathcal{G}. \tag{8}$$

In practical problems, the generation of individual plants might be restricted by more them their own operational limits. As an example, a group of units might have their combined output set to be at least a given level to provide ancillary services. Thus, constraints (9) and (10) are included in the model, and they, respectively, define lower and upper limits to the generation of given groups of units over the planning horizon.

$$\sum_{g \in \underline{\mathcal{G}}_i} p^{\mathrm{t}}_{g,t} + p^{\mathrm{s,\,t,\,min}}_{i,t} \geq \underline{\mathrm{P}}^{\mathrm{t,\,group}}_{i,t} \quad \forall i \in \underline{\mathcal{I}}^{\mathrm{t}}_{t}. \tag{9}$$

$$\sum_{g \in \overline{\mathcal{G}}_i} p^{\mathrm{t}}_{g,t} - p^{\mathrm{s,\,t,\,max}}_{i,t} \leq \overline{\mathrm{P}}^{\mathrm{t,\,group}}_{i,t} \quad \forall i \in \overline{\mathcal{I}}^{\mathrm{t}}_{t}. \tag{10}$$

2.2. Hydro generation

In (11), lower and upper limits on the main hydro variables are enforced. Although not all hydro plants have pumping and/or bypass capabilities, we include pumping variables q^{pump} and bypass variables q^{bypass} for all plants to avoid over notation. Thus, for plants with no pumping and/or bypassing, the respective limits can be appropriately set to zero. In the particular hydrothermal system that we are interested in, hydro plants equipped with pumping units have no turbines. Thus, for this particular case, if there is pumping for a plant h, then $\overline{\mathrm{Q}}^{\mathrm{pump}}_{h} > 0$ and $\overline{\mathrm{P}}^{\mathrm{h}}_{h} = 0$. Naturally, a generalization to accommodate plants with pumping units and turbines is straightforward, although omitted here for briefness.

$$\begin{aligned} \underline{\mathrm{V}}_{h} \leq v_{h,t} \leq \overline{\mathrm{V}}_{h}, 0 \leq q_{h,t} \leq \overline{\mathrm{Q}}_{h}, 0 \leq q^{\mathrm{pump}}_{h,t} \leq \overline{\mathrm{Q}}^{\mathrm{pump}}_{h}, \\ 0 \leq q^{\mathrm{bypass}}_{h,t} \leq \overline{\mathrm{Q}}^{\mathrm{bypass}}_{h}, 0 \leq s_{h,t} \leq \overline{\mathrm{S}}_{h}, 0 \leq p^{\mathrm{h}}_{h,t} \leq \overline{\mathrm{P}}^{\mathrm{h}}_{h} \\ \forall h \in \mathcal{H}. \end{aligned} \tag{11}$$

Outflows from to and inflows to reservoirs are defined in (12) and (13), respectively.

$$v^{\mathrm{out}}_{h,t} - \mathrm{C}^{\mathrm{h}} \cdot \left(q_{h,t} + q^{\mathrm{pump}}_{h,t} + q^{\mathrm{bypass}}_{h,t} + s_{h,t}\right) = 0 \quad \forall h \in \mathcal{H}. \tag{12}$$

$$v^{\mathrm{in}}_{h,t} - \mathrm{C}^{\mathrm{h}} \cdot \left(\sum_{u \in \mathcal{U}_h} q_{u,t-\mathrm{d}_{u,h}} + q^{\mathrm{pump}}_{u,t-\mathrm{d}_{u,h}} + q^{\mathrm{bypass}}_{u,t-\mathrm{d}_{u,h}} + s_{u,t-\mathrm{d}_{u,h}} \right) = \mathrm{C}^{\mathrm{h}} \cdot \mathrm{I}_{h,t} \quad \forall h \in \mathcal{H}. \tag{13}$$

Mass balance must be satisfied at all times and reservoirs. Nonetheless, due to the nature of the method used in this work, it is possible that, without slack variables, the mass-balance constraints can be violated. For instance, suppose that we have a hydro valley with only two reservoirs. The upper-stream reservoir is large and its current volume is far from the limits, while the down-stream one is run-of-river and has no regularization. Assuming that the water-travelling time is not instantaneous, then, in extreme cases, it may happen that the upper-stream reservoirs releases more water than the down-stream reservoir is capable of handling. In that scenario, the mass balance of the run-of-river reservoir would be violated. Therefore, as much as violations of mass balance are to be avoided, in the mass balance constraints (14), we include slack variables $v^{\mathrm{s,\,out}}_{h,t}$ and $v^{\mathrm{s,\,in}}_{h,t}$ to account for possible violations. Evidently, variables $v^{\mathrm{s,\,out}}_{h,t}$ and $v^{\mathrm{s,\,in}}_{h,t}$ are heavily penalized, as it will be seen shortly.

$$v_{h,t} - v_{h,t-1} + v^{\mathrm{out}}_{h,t} - v^{\mathrm{in}}_{h,t} + v^{\mathrm{s,\,out}}_{h,t} - v^{\mathrm{s,\,in}}_{h,t} = 0 \quad \forall h \in \mathcal{H}. \tag{14}$$

Another important aspect of the mathematical representation of hydro plants is the hydropower function. Here, we model the hydropower function as a piece-wise linear model in which the hydro generation is bounded above by linear functions of the plant's volume, turbine discharge and spillage, as in (15).

$$p^{\mathrm{h}}_{h,t} - \mathrm{F}^{\mathrm{vol}}_{h,j} \cdot v_{h,t} - \mathrm{F}^{\mathrm{turb}}_{h,j} \cdot q_{h,t} - \mathrm{F}^{\mathrm{spil}}_{h,j} \cdot s_{h,t} \leq \mathrm{F}^{\mathrm{const}}_{h,j} \quad \forall h \in \mathcal{H}, \forall j \in \mathcal{F}_h. \tag{15}$$

Lastly, we present in (16) the cost-to-go function.

$$\alpha - \mathrm{A}^{\mathrm{coeff}}_{j,h} \cdot v_{h,\mathrm{T}} \leq \mathrm{A}^{\mathrm{const}}_{j} \quad \forall j \in \mathcal{A}. \tag{16}$$

2.3. Network

The last main component in our model is the network. We use a DC model to represent the transmission lines' flows. In (17), we have the power balance at each bus of the network; in (18), we show the dependence of the flows on the buses' voltage angles and their respective limits in (19). Note that the power drawn by the pumping units is explicitly included in (17) by the term $\mathrm{C}_h^{\mathrm{pump}} \cdot q_{h,t}^{\mathrm{pump}}$.

$$\begin{aligned} \sum_{g\in\mathcal{G}} p_{g,t}^{\mathrm{t}} + \sum_{h\in\mathcal{H}} p_{h,t}^{\mathrm{h}} - \sum_{h\in\mathcal{H}} \mathrm{C}_h^{\mathrm{pump}} \cdot q_{h,t}^{\mathrm{pump}} + \sum_{l\in\mathcal{L}_b^+} f_{l,t} \\ - \sum_{l\in\mathcal{L}_b^-} f_{l,t} + p_{b,t}^{\mathrm{s,\,n,\,+}} - p_{b,t}^{\mathrm{s,\,n,\,-}} = \mathrm{L}_{b,t} \quad \forall b \in \mathcal{B}. \end{aligned} \tag{17}$$

$$f_{l,t} = \mathrm{X}_l \cdot \left(\theta_{\mathrm{B}_l^+,t} - \theta_{\mathrm{B}_l^-,t}\right) \quad \forall l \in \mathcal{L}. \tag{18}$$

$$\underline{\mathrm{F}}_{l,t} \le f_{l,t} \le \overline{\mathrm{F}}_{l,t} \quad \forall l \in \mathcal{L}. \tag{19}$$

2.4. Hydrothermal unit commitment

The components described above are combined to form the hydrothermal unit-commitment model in (20).

$$\begin{aligned} \phi = \min \quad & \sum_{t\in\mathcal{T}} \sum_{g\in\mathcal{G}} \mathrm{C}_g^{\mathrm{t,cost}} \cdot p_{g,t}^{\mathrm{t}} + \alpha + \\ & \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}} \sum_{b\in\mathcal{B}} \left(p_{b,t}^{\mathrm{s,\,n,\,+}} + p_{b,t}^{\mathrm{s,\,n,\,-}}\right) + \\ & 10^6 \cdot \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}} \sum_{h\in\mathcal{H}} \left(v_{h,t}^{\mathrm{s,\,out}} + v_{h,t}^{\mathrm{s,\,in}}\right) + \\ & \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}} \left(\sum_{i\in\underline{\mathcal{I}}_t^{\mathrm{t}}} p_{i,t}^{\mathrm{s,\,t,\,min}} + \sum_{i\in\overline{\mathcal{I}}_t^{\mathrm{t}}} p_{i,t}^{\mathrm{s,\,t,\,max}} \right) \\ \text{s.t.} \quad & (1)\text{–}(10) \quad \forall t \in \mathcal{T}, (11)\text{–}(15) \quad \forall t \in \mathcal{T}, \\ & (16), (17)\text{–}(19) \quad \forall t \in \mathcal{T}, \\ & 0 \le y_{g,t}, x_{g,t}, d_{h,t}^{\mathrm{t}} \le 1 \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}, \\ & y_{g,t}, x_{g,t}, d_{g,t}^{\mathrm{t}} \in \{0,1\} \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}. \end{aligned} \tag{20}$$

As we mention in Subsection 2.2, the slack variables in the mass balance constraints (14) are heavily penalized: we choose here a penalty that is 10^6 times the penalty for imbalances in the power balance constraints of the network buses.

3. Parallel dual dynamic integer programming

Dynamic programming consists of breaking a problem over time into smaller subhorizons that are sequentially solved to, at the end, recover a solution to the problem. A subhorizon is a block containing one or more periods of the original planning horizon. To each subhorizon, there is a associated optimization problem whose feasible set is defined by the original problem's constraints for the respective time periods, plus nonanticipativity constraints of the coupling variables (i.e., constraints that enforce that decisions taken in previous subhorizons must not change). Similarly, its objective function comprises costs for the periods in the subhorizon and, possibly, an approximation of the cost-to-go function for subsequent subhorizons.

In (20), there are nine coupling variables: y, x, d, $p^{t,disp}$, v, q, s, q^{pump}, and q^{bypass}. Assuming that the planning horizon is partitioned into $|\mathcal{S}|$ subsets (subhorizons), with $\mathcal{S}$ being the set of indices for the subhorizons, then, for an arbitrary subhorizon $s \in \mathcal{S}$, its subset of $\mathcal{T}$ is denoted as $\mathcal{T}^s$. Consider that the vectors of coupling variables can be sliced so, for example, y_s contains all y variables from the beginning of the planning horizon up to, and including, those in periods $\mathcal{T}^s$ — all y variables up to subhorizon s. Additionally, we denote as $\mathbf{y}_{s,j}$ the values of y variables up to subhorizon s in the j-nth solution. For conciseness, we define the j-nth vector of coupling variables' decisions taken in subhorizons previous to and including subhorizon s as $\mathbf{w}^{s,j} = \left(\mathbf{y}_{s,j}, \mathbf{x}_{s,j}, \mathbf{d}_{s,j}, \mathbf{p}_{s,j}^{t,disp}, \mathbf{v}_{s,j}, \mathbf{q}_{s,j}, \mathbf{s}_{s,j}, \mathbf{q}_{s,j}^{pump}, \mathbf{q}_{s,j}^{bypass}\right)$. Vector w^s is similarly defined as the vector of variables up to subhorizon s. Furthermore, we define as $\mathcal{J}^s$ the set of indices for available vectors $\mathbf{w}^{s,j}$ ($\mathcal{J}^s$ can be, for instance, the set of previous iterations). Hence, the s-subhorizon problem is

$$\begin{aligned} & \phi_{s,k}\left(\mathbf{w}^{s-1,k}\right) = \\ \min \; & \sum_{t\in\mathcal{T}^s} \sum_{g\in\mathcal{G}} \mathrm{C}_g^{\mathrm{t,cost}} \cdot p_{g,t}^{\mathrm{t}} + \alpha + \beta_s + \\ & \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s} \sum_{b\in\mathcal{B}} \left(p_{b,t}^{\mathrm{s,\,n,\,+}} + p_{b,t}^{\mathrm{s,\,n,\,-}}\right) + \\ & 10^6 \cdot \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s} \sum_{h\in\mathcal{H}} \left(v_{h,t}^{\mathrm{s,\,out}} + v_{h,t}^{\mathrm{s,\,in}}\right) + \\ & \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s} \left(\sum_{i\in\underline{\mathcal{I}}_t^{\mathrm{t}}} p_{i,t}^{\mathrm{s,\,t,\,min}} + \sum_{i\in\overline{\mathcal{I}}_t^{\mathrm{t}}} p_{i,t}^{\mathrm{s,\,t,\,max}} \right) \\ \text{s.t.} \; & (1)\text{–}(10) \quad \forall t \in \mathcal{T}^s, (11)\text{–}(15) \quad \forall t \in \mathcal{T}^s, \\ & (16), (17)\text{–}(19) \quad \forall t \in \mathcal{T}^s, \\ & 0 \le y_{g,t}, x_{g,t}, d_{h,t}^{\mathrm{t}} \le 1 \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\ & y_{g,t}, x_{g,t}, d_{g,t}^{\mathrm{t}} \in \{0,1\} \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\ & w^{s-1} = \mathbf{w}^{s-1,k}, \\ & \beta_s \ge \underline{\phi}_{s+1,j}\left(\mathbf{w}^{s,j}\right) + \left\langle \boldsymbol{\pi}^{s+1,j}, \left(w^s - \mathbf{w}^{s,j}\right) \right\rangle \; \forall j \in \mathcal{J}^s. \end{aligned} \tag{21}$$

Firstly, it is important to note that s-subhorizon problem (21) contains all coupling variables up to subhorizon s itself — even those that do not appear in any of its constraints. In (21), decisions in subhorizons previous to s are fixed through equality $w^{s-1} = \mathbf{w}^{s-1,k}$. Furthermore, $\underline{\phi}_{s+1,j}\left(\mathbf{w}^{s,j}\right)$ is an underestimate of the optimal cost of subhorizon $s+1$ obtained by solving a convex relaxation of (21) for subhorizon $s+1$. Besides $\underline{\phi}_{s+1,j}\left(\mathbf{w}^{s,j}\right)$, solving the relaxation must also provide a subgradient to $\underline{\phi}_{s+1,j}(\cdot)$ at $\mathbf{w}^{s,j}$, $\boldsymbol{\pi}^{s+1,j}$. In this work, we use the continuous relaxation, which is obtained simply by dropping the integrality constraints in (21). Then, feasible dual values associated with equalities $w^{s-1} = \mathbf{w}^{s-1,k}$ provide a valid subgradient to the value function of the relaxation at $\mathbf{w}^{s-1,k}$, i.e., $\boldsymbol{\pi}^{s,j} \in \partial\underline{\phi}_{s,j}\left(\mathbf{w}^{s-1,k}\right)$. We care to note that other relaxations, likely tighter, can be employed [39]. In (21), $\langle , \rangle$ denotes the inner product of two vectors. Additionally, variable α and constraints (16) only appear in the subhorizon problem that contains the last period, but, for simplicity, they are kept in (21).

In dual dynamic programming parlance, sequentially solving the subhorizon problems is called forward pass, or forward step. Likewise, subgradients and underestimates of the subhorizon problems' optimal costs are obtained in the backward step by solving the relaxations of the subhorizon problems in a inverse order, i.e., from last to first. We summarize the dual dynamic integer programming approach for a sequential strategy in Fig. 2.

3.1. Asynchronous optimization

Asynchronous algorithms are often designed to circumvent the need for complete information before the beginning of a new iteration in iterative algorithms, and thus reduce processes' idleness in an attempt to accelerate convergence. For algorithms whose one or more steps consist of the evaluation of multiple, independent functions whose outputs are then used as inputs for the next iteration, asynchronicity is often beneficial if the time taken for evaluating the functions vary substantially and not all functions' outputs are immediately necessary for significant improvements in the following iterations. Suppose, for instance, that three processes, 0, 1, and 2, are available for iteratively solving a problem by sequentially performing tasks that depend on previous iterations' results. More specifically, assume that process 0 sends its output x to processes 1 and 2, which will then perform tasks with x as input and return y_1 and y_2, respectively, to process 0. In possess of y_1 and y_2, then process 0 starts a new iteration. However, because the beginning of a new iteration depends on process 0 receiving both y_1 and y_2 if either process 1 or 2 takes considerable more time to perform its task than the other process, then process 0 (as well as the other quicker process) will remain idle until complete information is obtained, even as partial information is already available. However, if information from either process 1 or 2 is enough for process 0 to make significant progress in the next iteration, then an asynchronous algorithm may be a good alternative. We illustrate this example in Fig. 3, where we compare the synchronous approach with an asynchronous one.

In the purposefully exaggerated example of Fig. 3, the idle times of all processes were reduced in the asynchronous algorithm. This partially resulted in more tasks being performed by all processes. Although those tasks were performed on partial, as mentioned before, for problems and methodologies for which partial information suffices, asynchronous algorithms often outperform their synchronous counterparts in terms of computational times.

In [10], we have developed an asynchronous parallel version of the dual dynamic integer programming algorithm. The basic idea behind our approach is to simultaneously exploit the cheap subgradients given by the continuous relaxation in the backward steps and the good upper bounds provided by the forward step even for temporal decompositions with many subhorizons. Here, as in [10], processes are divided into three categories: a single general coordinator; processes assigned to performing forward steps, forward workers; and those assigned to performing backward steps, backward workers. We improve [10]'s general coordinator's algorithm to include a synchronous option in Alg. 1. In this algorithm, *tol*, *LB*, and UB are respectively, the tolerance stopping criterion for the relative gap, the lower bound and upper bound. Also, *maxIt* is the pre-defined maximum number of iterations, set to infinity in asynchronous optimization; *nDuals* is the total number of dual solutions, which depends on the number of backward workers and their configurations; *dualRec* is a set used for storing received dual solutions. Moreover, the asynchronous strategy is chosen if $synch = False$.

Algorithm 1: Algorithm for the general coordinator. Based on [10].

Input: $tol > 0$, $maxIt \geq 0$, $nDuals \geq 0$, $synch \in \{true, false\}$
Output: LB and UB

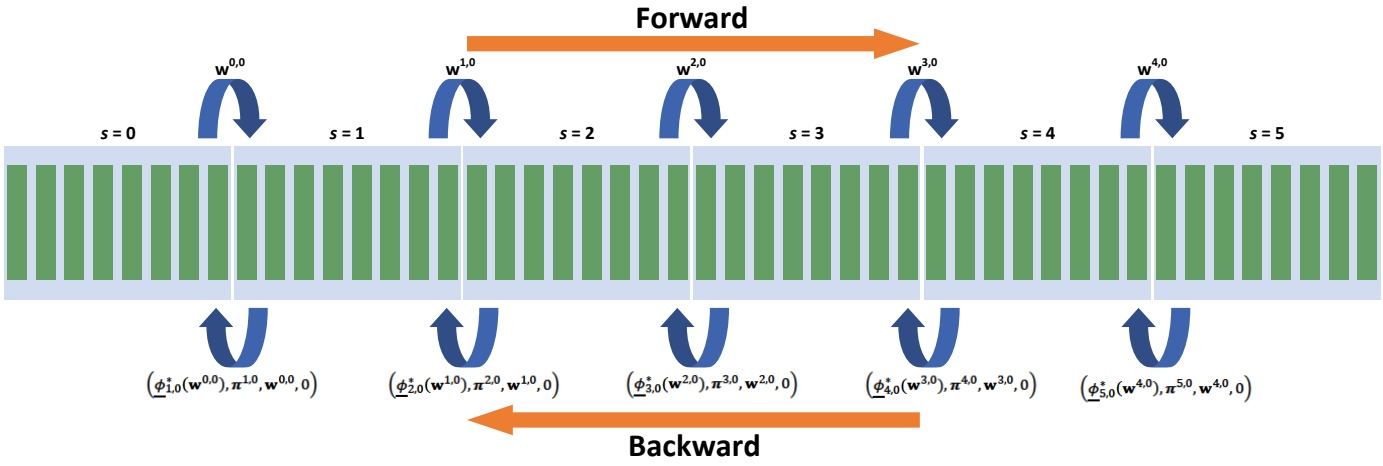
```
1  LB ← 0, UB ← ∞, gap ← 1, it ← 0, dualRec ← ∅;
2  while ((gap > tol) ∧ (it < maxIt)) do
3      (w, newLB, newUB) ← receiveSolution();
4      if w is a primal solution then
5          updateLB(newLB);
6          updateUB(newUB);
7          if ¬synch then
8              gap ← (UB − LB)/UB;
9          sendToBackward(w, LB, UB);
10     else if w is a dual solution then
11         updateLB(newLB) ;
12         if ¬synch then
13             sendToBackward(w, LB, UB);
14             sendToForward(w, LB, UB);
15             gap ← (UB − LB)/UB;
16         else
17             dualRec ← dualRec ∪ {w};
18     if synch ∧ (|dualRec| = nDuals) then
19         for w ∈ dualRec do
20             sendToAllBackward(w);
21             sendToAllForward(w);
22         gap ← (UB − LB)/UB;
23         it ← it + 1;
24         dualRec ← ∅;
```

Likewise, in Alg. 2, we update the workers' algorithm presented in [10] to accommodate the possibility of synchronous optimization. If asynchronous optimization is used ($synch = false$), then forward and backward workers are allowed to proceed as soon as an appropriate dual solution has been received. In contrast, in synchronous optimization, $synch = true$, the workers only continue once all dual information is received. They keep count of the dual solutions received through counter *dCounter*, and check the requirement by comparing it to the total number of dual solutions to be received, *nDuals*.

3.2. Synchronous optimization

Asynchronous algorithms work on the premises that once enough partial information is available, it is not strictly neces-

Figure 2: Illustration of dual dynamic programming for a problem with 48 periods of planning horizon broken into six subhorizons, each with eight periods. The subhorizons are identified from 0 to 5 and the periods in each of them are shown by the light-blue rectangles. In the forward step, decisions for the coupling variables are propagated from the first subhorizon to the last one, at which point, a complete solution to the original problem is available along with an associated upper bound (for a minimization problem). Contrary to the forward step, the backward step starts from the last subhorizon and improves the approximation of the cost-to-go function of the subhorizon problems by adding a new inequality to it with ϕ, $\boldsymbol{\pi}$, and $\mathbf{w}$. In addition to these three data, note that the identification of the solution evaluated in the backward step is also transmitted in the backward step, in this case, solution 0. Although this is not necessary for sequential dual dynamic programming, it is used in our parallel version.



sary to wait for complete information. Partial information, however, can differ each time an algorithm is run because running times for computer tasks are intrinsically non-deterministic: not one logical or arithmetic operation takes always exactly the same time. Provided the same computational environment and input data, if all machine operations always took the same time, then asynchronous algorithms would be deterministic. Take the example of Fig. 3 for instance, if the next time the asynchronous algorithm is run, process 2 returns its $y_{2,0}$ output faster than process 1, then it is likely that the next output of process 0, x'_1, will not be the same, since it will be computed on different information. Consequently, all subsequent outputs will also likely change.

The non-deterministic behaviour of asynchronous algorithms can be undesirable since it might prevent results from being reproducible even under the same conditions. While synchronicity does not necessarily mean reproducibility under any condition (for instance, for different machines or operating systems), it provides reproducible results if the computational resources, hardware and software, are identical.

To ensure that our results are reproducible, our synchronous strategy must have no time dependent components, this includes the black box optimization solvers used for solving the subhorizon problems. `Gurobi` [20] is deterministic as long as the time limit for solving the problem is not set, concurrent optimization is not used, and a time limit for heuristically finding a solution before solving the root relaxation (for mixed integer problems) is also not set. To the best of the authors' knowledge, there is no official documentation of `Cbc` and `Clp` on determinism. However, in our experiments, we have found that using a single thread, no heuristics, and no time limit render these solvers deterministic. Another important factor for ensuring determinism, for any solver, is solving *exactly* the same problems. To understand why, we first need to remember that the optimization model actually solved by solvers is often different from, and not always strictly equivalent to, the one passed to them. Modern optimization solvers apply pre-solve techniques to obtain smaller and often tighter models. Forsaking presolving can significantly compromise the solvers' ability to solve the model in a timely fashion — or to solve it at all if the model is numerically challenging. One model characteristic that influences presolving is variable and constraint ordering. Therefore, adding constraints in a different order can produce different pre-solved models, which ultimately might give different solutions or even different optimal values. Thus, in our synchronous strategy, we make sure that all variables and constraints are added to the models in the same order, including the constraints of the cost-to-go functions' approximations that are added on-the-fly.

The main difference of our synchronous approach from the asynchronous one, as expected, is that a new iteration can only begin once all information, from forward and dual workers, is received by the general coordinator. Additionally, instead of forwarding dual solutions as soon as they are received, as in Steps 13 and 14 of Alg. 1, with the synchronous option, the general coordinator stores the dual solutions in set *dualRec* and only when all dual solutions have been received, identified by *nDuals*, they are sent to backward and forward workers in Step 19. In Alg. 1, we have omitted steps for conciseness. For instance, we do not explicitly state that sending the dual solutions to backward and forward workers in Step 19 must always be done in the same order. Equally important is that the primal solutions sent to backward workers in Step 9 must always respect that same forward-backward worker pair and synchronous optimization is used, i.e., backward workers always receive primal solutions from the same forward workers. Lastly, note that the gap is only updated at the end of the iteration for the syn-

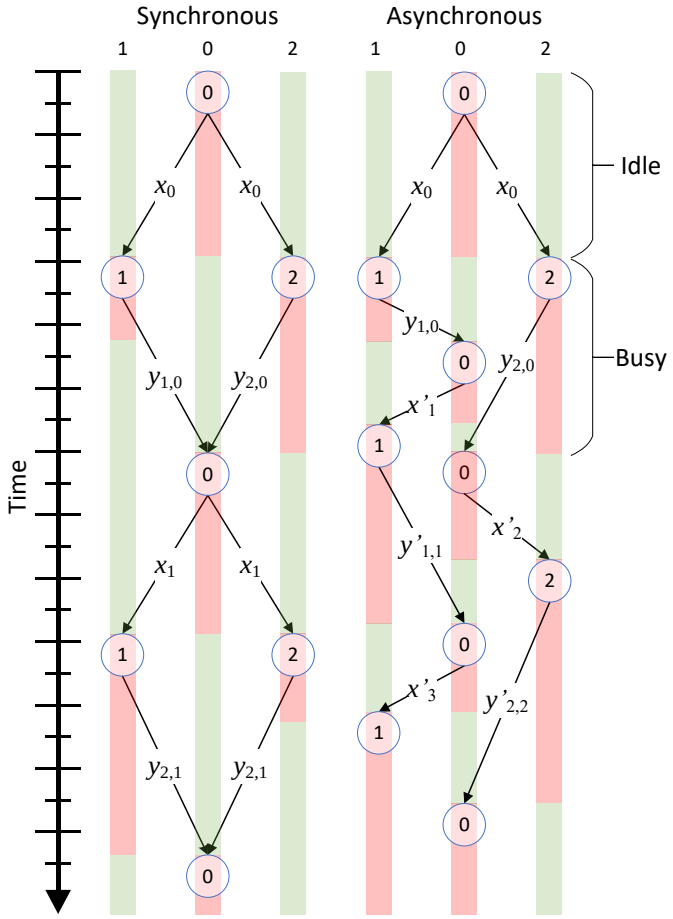
Algorithm 2: Forward and backward workers. Based on [10].

Input: $synch \in \{true, false\}$, $nDuals \geq 0$

Procedure $ForwardWorker(tol)$**:**
- $LB \leftarrow 0$, $UB \leftarrow \infty$, $dCounter \leftarrow 0$;
- **while** $((UB - LB)/UB > tol)$ **do**
  - $x \leftarrow forwardStep()$;
  - $sendPrimalSol(x, LB, UB)$;
  - **if** $synch$ **then**
    - **while** $dCounter < nDuals$ **do**
      - $(y, newLB, newUB) \leftarrow receiveDualSol()$;
      - $updateLB(newLB)$;
      - $updateUB(newUB)$;
      - $addNewCuts(y)$;
      - $dCounter \leftarrow dCounter + 1$;
  - **else**
    - $(y, newLB, newUB) \leftarrow receiveDualSol()$;
    - $updateLB(newLB)$;
    - $updateUB(newUB)$;
    - $addNewCuts(y)$;
  - $dCounter \leftarrow 0$;

Procedure $BackwardWorker(tol)$**:**
- $LB \leftarrow 0$, $UB \leftarrow \infty$, $dCounter \leftarrow 0$;
- **while** $((UB - LB)/UB > tol)$ **do**
  - $(x, newLB, newUB) \leftarrow receivePrimalSol()$;
  - $updateLB(newLB)$;
  - $updateUB(newUB)$;
  - $y \leftarrow backwardStep(x)$;
  - $sendDualSol(y, LB, UB)$;
  - **if** $synch$ **then**
    - **while** $dCounter < nDuals$ **do**
      - $(y, newLB, newUB) \leftarrow receiveDualSol()$;
      - $updateLB(newLB)$;
      - $updateUB(newUB)$;
      - $addNewCuts(y)$;
      - $dCounter \leftarrow dCounter + 1$;
  - **else**
    - $(y, newLB, newUB) \leftarrow receiveDualSol()$;
    - $updateLB(newLB)$;
    - $updateUB(newUB)$;
    - $addNewCuts(y)$;
  - $dCounter \leftarrow 0$;

Figure 3: Illustrative example of a synchronous and a asynchronous algorithm. The connecting arrows represent information exchange between process. Idle and busy times are represent by light green and light red bars, respectively, with the elapsed time in the y axis. For both synchronous and asynchronous, the subscripts of x uniquely identify the different outputs of process 0. Likewise, the outputs of processes 1 and 2 are identified by the first subscript of y, while the second one corresponds to x's id used to generate that y. Finally, note that outputs in the asynchronous strategy are equipped with quotation marks to indicate that they may differ from those in the synchronous strategy.



chronous approach which prevents time-dependent termination. Because the stopping criterion of the workers is the relative gap in Alg. 2 and the UB and LB bounds of workers are only updated through the information received by the general coordinator, which itself only updates its bounds at the end of the iteration, forward and backward workers will terminate only when the general coordinator does so. Furthermore, forward workers and backward workers can only start a new iteration once complete dual solution has been received by them, which, in turn, only happens at the end of the general coordinator's iteration. Thus, iterations of general coordinator, forward and backward workers are synchronized.

3.3. Benders decomposition

Even with temporal decomposition, solving HTUC with a free solver can be challenging due to the size of the resulting subhorizon problems. Therefore, decomposing the subhorizon

problems themselves might be necessary. At this point, there are at least two alternatives: dual decomposition, and primal decomposition. Dual decomposition breaks the original problem by dualizing coupling constraints. If the original problem is non-convex, there might be a duality gap between the optimal dual value and the original problem's optimal value. More importantly, recovering a feasible primal solution from the dual one might require primal recovery techniques. On the other hand, primal decomposition approaches work on complicating variables, and they naturally yield primal-feasible solutions. Among them, Benders decomposition (BD) was originally proposed in [4] for solving mixed integer programming problems in which integer variables are deemed the complicating ones, meaning that once they are fixed, the resulting problem is significantly easier to solve than the original one. The resulting problem with the complicating variables fixed is called the subproblem (SP), whereas the optimization model used deciding the values of the complicating variables is the master problem (MP). If the MP is a relaxation of the original problem, then solving it naturally yields a lower bound to the original problem (given that it is a minimization problem). In turn, evaluating the MP's decisions on the SP returns an upper bound. Moreover, the SP's value function in terms of the MP's variables is approximated from below in the MP by the so-called optimality cuts and it is iteratively improved with subgradients of the SP's value function at evaluated points.

In this work, we apply the classical BD in three different ways. In the first one, as in [4], we define the subhorizon problem (21)'s binary variables as the complicating ones.

3.3.1. Integer separation

As in the seminal work [4], here we simply separate the integer part of the subhorizon problem (21) from its continuous part. Hence, we have (22) as the MP in this inner decomposition, and (23) as the SP. Because the MP does not have all coupling variables, we choose to add the approximation of the subsequent subhorizon's optimal cost to the SP. In (22), we furnish the vector of coupling variables w with an additional superscript b to indicate w is being sliced to get only the appropriate binary variables. Also, in (22), continuous auxiliary variable σ_s is used for approximating from below the optimal cost of the SP. The optimality cuts generated in this inner BD are accumulated in $\mathcal{I}$, which is initially empty. To avoid excessive notation, we omit iteration indices for the inner BD. Furthermore, in (22), we concatenate the binary variables y, x and d into vector r.

Additionally, to make sure that MP's decisions do not induce infeasibilities in the SP, dispatch statuses d in (22) must be set according to the statuses and generation decisions of previous subhorizons. For instance, suppose that an arbitrary thermal unit g has a minimum generation limit of 50 MW, ramping down limit of 5 MW/30 min (considering 30-min periods), and it is generating 70 MW at the period immediately before the beginning of the current subhorizon. Disregarding up-time constraints, in this scenario, g cannot be shut down for four periods because, to be shut-down, it needs to generate at its minimum level first. Therefore, thermal unit g's dispatch status variables d need to forcibly be set to one at least for the next four periods by introducing feasibility cuts. Nonetheless, for simplicity, we omit such constraints in (22).

$$
\begin{aligned}
\nu_s\left(\mathbf{w}^{s-1,k}\right) = \min\ & \sigma_s \\
\text{s.t.}(1)\text{–}(10) \quad & \forall t \in \mathcal{T}^s, \\
& 0 \le y_{g,t}, x_{g,t}, d^{\mathrm{t}}_{h,t} \le 1\ \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\
& y_{g,t}, x_{g,t}, d^{\mathrm{t}}_{g,t} \in \{0,1\}\ \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\
& w^{\mathrm{s\ -\ 1,\ b}} = \mathbf{w}^{s-1,k,b}, \\
& \sigma_s \ge \psi_s\left(\mathbf{r}^i\right) + \left\langle \lambda^i, \left(r - \mathbf{r}^i\right)\right\rangle \forall i \in \mathcal{I}.
\end{aligned}
\tag{22}
$$

In the inner SP (23), the values of variables y, x, and d, which are continuous in this SP, are fixed through constraints $r = \mathbf{r}$. Naturally, the values of the dual variables associated with these constraints give valid subgradients to the SP's value function $\psi_s\left(\mathbf{w}^{s-1,k}, \cdot\right)$. Furthermore, because (23) changes with the input from previous subhorizons $\mathbf{w}^{s-1,k}$, the optimality cuts added to (22) have to be removed from it after each iteration of the outer DDiP.

$$
\begin{aligned}
\psi_s\left(\mathbf{w}^{s-1,k}, \mathbf{r}\right) = \min\ & \sum_{t\in\mathcal{T}^s}\sum_{g\in\mathcal{G}} \mathrm{C}^{\mathrm{t,cost}}_g \cdot p^{\mathrm{t}}_{g,t} + \alpha + \beta_s + \\
& \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s}\sum_{b\in\mathcal{B}} \left(p^{\mathrm{s,\ n,\ +}}_{b,t} + p^{\mathrm{s,\ n,\ -}}_{b,t}\right) + \\
& 10^6 \cdot \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s}\sum_{h\in\mathcal{H}} \left(v^{\mathrm{s,\ out}}_{h,t} + v^{\mathrm{s,\ in}}_{h,t}\right) + \\
& \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s} \left(\sum_{i\in\underline{\mathcal{I}}^{\mathrm{t}}_t} p^{\mathrm{s,\ t,\ min}}_{i,t} + \sum_{i\in\overline{\mathcal{I}}^{\mathrm{t}}_t} p^{\mathrm{s,\ t,\ max}}_{i,t} \right) \\
\text{s.t.}(11)\text{–}(15) \quad & \forall t \in \mathcal{T}^s, \\
& (16), (17)\text{–}(19) \quad \forall t \in \mathcal{T}^s, \\
& w^{s-1} = \mathbf{w}^{s-1,k}, \\
& r = \mathbf{r}, \\
& \beta_s \ge \underline{\phi}_{s+1,j}\left(\mathbf{w}^{s+1,j}\right) + \left\langle \boldsymbol{\pi}^{s+1,j}, \left(w^s - \mathbf{w}^{s,j}\right)\right\rangle \forall j \in \mathcal{J}^s.
\end{aligned}
\tag{23}
$$

3.3.2. Network separation

If the binary variables do not introduce as much computational burden compared to the network model, then separating the subhorizon problem into a generation problem and a network problem might be interesting. In fact, it is not unusual for the majority of a UC's constraints and variables to be due to the network model. Moreover, usually only a fraction of transmission lines' limits are binding at any given solution of a UC, thus making most of them redundant [2]. Hence, we devise the MP (24) encompassing the binary and generation decisions of the subhorizon problem (21) in an attempt to reduce the computational burden of the subhorizon problem and also to exploit the redundancy of most transmission lines' limits. Because this problem contains all coupling variables, we can add the approximation of the cost-to-go function to it. Also, in (24), we

use η_s to approximate the cost of network violations. Different from the integer separation of 3.3.1, the SP in this case does not change with the decisions on the coupling variables. Consequently, the optimality cuts added to (24) do not have to be removed after each outer iteration. Furthermore, given the slack network variables in the SP (25), it is always feasible regardless the MP (24)'s decisions.

$$
\begin{aligned}
\gamma_{s,k}\left(\mathbf{w}^k\right) = \min \; & \sum_{t\in\mathcal{T}^s}\sum_{g\in\mathcal{G}} \mathrm{C}^{\mathrm{t,cost}}_g \cdot p^{\mathrm{t}}_{g,t} + \alpha + \beta_s + \eta_s + \\
& 10^6 \cdot \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s}\sum_{h\in\mathcal{H}} \left(v^{\mathrm{s,\,out}}_{h,t} + v^{\mathrm{s,\,in}}_{h,t}\right) + \\
& \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s} \left(\sum_{i\in\underline{I}^{\mathrm{t}}_t} p^{\mathrm{s,\,t,\,min}}_{i,t} + \sum_{i\in\overline{I}^{\mathrm{t}}_t} p^{\mathrm{s,\,t,\,max}}_{i,t} \right) \\
\text{s.t.} \; & (1)\text{–}(10) \quad \forall t \in \mathcal{T}^s, (11)\text{–}(15) \quad \forall t \in \mathcal{T}^s, (16), \\
& 0 \le y_{g,t}, x_{g,t}, d^{\mathrm{t}}_{h,t} \le 1 \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\
& y_{g,t}, x_{g,t}, d^{\mathrm{t}}_{g,t} \in \{0,1\} \qquad \forall g \in \mathcal{G}, \forall t \in \mathcal{T}^s, \\
& w^{s-1} = \mathbf{w}^{s-1,k}, \\
& \beta_s \ge \underline{\phi}_{s+1,j}\left(\mathbf{w}^{s+1,j}\right) + \left\langle \boldsymbol{\pi}^{s+1,j}, \left(w^s - \mathbf{w}^{s,j}\right)\right\rangle \; \forall j \in \mathcal{J}^s, \\
& \eta_s \ge \omega_s\left(\mathbf{p}^i\right) + \left\langle \boldsymbol{\mu}^i, \left(p - \mathbf{p}^i\right)\right\rangle \; \forall i \in \mathcal{I}.
\end{aligned}
\tag{24}
$$

In (24), the generation decisions p^t and p^h are concatenated into p. Then, the optimal values of these variables given in the MP's optimal solution found, $\mathbf{p}$, are fixed in the network problem with $p = \mathbf{p}$ (we again omit the inner iteration index). Evidently, the values μ of the dual variables associated with these constraints are used in the optimality cuts of (24) for approximating $\omega_{\mathrm{s}}(\cdot)$.

$$
\begin{aligned}
\omega_{\mathrm{s}}(\mathbf{p}) = \min \quad & \mathrm{C}^{\mathrm{s,b}} \cdot \sum_{t\in\mathcal{T}^s}\sum_{b\in\mathcal{B}} \left(p^{\mathrm{s,\,n,\,+}}_{b,t} + p^{\mathrm{s,\,n,\,-}}_{b,t}\right) \\
\text{s.t.} \quad & (17)\text{–}(19) \quad \forall t \in \mathcal{T}^s, \\
& p = \mathbf{p}.
\end{aligned}
\tag{25}
$$

3.3.3. Double separation

Even after the inner decompositions of 3.3.1 and 3.3.2, the resulting MPs and/or SPs might still be too complex to be handled in a timely fashion by a free solver. Therefore, in this approach, firstly we use BD to separate the integer part of the subhorizon problem from the continuous part, as in 3.3.1. Then, we further decompose the resulting SP by separating the network problem from the generation problem, as in 3.3.2. Thus, with this double decomposition, we have (22) as the MP, in which the values of the binary variables are decided, then the SP has as its own inner MP (24), with the important difference that the binary variables are now fixed, and (25) as an inner SP.

4. Experiments

Our test system is based on actual data of the Brazilian power system and it has 161 hydro reservoirs, 329 thermal generating units, 7,475 buses and 10,698 transmission lines. In total, the system has an installed capacity of 132,727 MW, 23,902 MW from the thermal units and the remainder from the hydro plants. The planning horizon is composed by 48 30-min periods. The cases are similar to those of [10], however, the hydro model in this work is simpler than that of [10] due to the limitations of current free solvers. Nonetheless, the hydro model used here is still similar to the model currently used in practice by the Brazilian ISO [34]. All input data, as well as the source code, are available at https://github.com/SPARHTACUS/SPTpy. As in [10], we assess our strategies in 20 cases, Case 1 to Case 20. In Tab. 2, we show the number of variables and constraints for Case 1. Although the dimensions slightly change from case to case, for instance, due to differences in the hydropower functions' approximations, they are similar across all of them.

Table 2: Dimensions of the optimization problem of Case 1 with a single subhorizon after removing non-umbrella constraints [2].

Constraints	Continuous variables	Binary variables
599,513	598,570	47,376

All experiments are conducted on a machine with two Intel Xeon E5-2660 v3 processors clocking at 2.60 GHz, with a total of 20 physical cores, and 128 GB of RAM. The operating system is Ubuntu 20.04.4; all codes were written in Python, and the Python codes' interpreter is PyPy 7.3.5; inter-process communication is done through Open MPI 4.1.1, with mpi4py [13] interfacing Python and Open MPI's libraries. The state-of-the-art, commercial solver used in our comparisons is `Gurobi` [20], version 9.5.0. `Cbc` [8], version 2.10.6, is our free mixed-integer programming solver of choice, with `Clp` [9], version 1.17.6, used for solving the linear programming problems. To ensure that the optimization problems passed to both `Gurobi` and `Cbc` are identical, we use the COIN-OR's Python modelling environment Python-MIP.

Our goal is to timely solve the hydrothermal unit-commitment problem at hand to a 0.1% relative gap, the same gap tolerance currently used in Brazil [34], with a free and open-source solver. Thus, naturally, one of the stopping criteria used in all experiments is 0.1% for the relative gap of the dual dynamic integer programming (in all its versions). The dual dynamic integer programming's mixed-integer subhorizon problems are solved to a 0.01% gap, the exception being the experiments with a single subhorizon — when dual dynamic programming is not used, for which the gap tolerance is set to 0.1%. When the integer separation presented in 3.3.1 is used, its mixed-integer MP is solved to a 0.001% gap, while the mixed-integer MP of 3.3.2 is solved to a 0.01% gap. All linear programming problems, regardless of the decomposition used, are solved to optimality with the barrier algorithm with crossover enabled. For the synchronous strategy, we set an iteration limit of 50. For the inner BD of 3.3.1, the iteration limit is 5, whereas for 3.3.2, the limit is 2. The same limits are used in the double decomposition shown in 3.3.3. These iteration limits were found to be satisfactory for the problems at hand, although it is probable that better ones can be found.

As for time limits, the asynchronous strategy has a time limit of one hour, whereas there is no time limit for the syn-

chronous one. Owing to the non-deterministic nature of asynchronous methods, we run all experiments with the asynchronous strategy five times to obtain more robust results. We do the same for the synchronous algorithm, although for a different reason: to show that it behaves deterministically for the problem at hand and the computational setup used.

Furthermore, we choose a configuration of 20 processes for both the asynchronous and the synchronous methods: one general coordinator, three forward workers, and 16 backward workers. The number of subhorizons for the three forward workers are eight, 16, and 24, and all subhorizons for all workers have the same number of periods, i.e., the planning horizon is uniformly divided. As for the backward workers, we configure four of them with two subhorizons, other three with four subhorizons, three with eight subhorizons, three with 16 subhorizons, and the last three with 24 subhorizons. The groups of three backward workers whose numbers of subhorizons match those of the forward workers are used to ensure that the approximations of the cost-to-go function for each subhorizon of all forward workers is continually improved. Moreover, the backward workers with two and four subhorizons are utilized to further improve the cost-to-go functions' approximations. On the other hand, the additional backward worker with two subhorizons does not yield any cut to the approximations of the cost-to-go functions, but gives valid lower bounds to the optimal value of the original problem. To that end, this additional worker solves only its first subhorizon, all other backward workers do not solve their respective first subhorizons. To improve the lower bounds given by this additional worker, both in the asynchronous and the synchronous strategies, the three other backward workers with two subhorizons feed it with approximations of the cost-to-go function of the last subhorizon. For a better understanding, we illustrate in Fig. 4 how the approximations of the cost-to-go functions are shared among the backward and forward workers.

Finally, the inner decompositions of Subsections 3.3.1, 3.3.2 and 3.3.3 are applied to both forward and backward steps in order to quickly obtain good solutions and bounds. More specifically, we use the double separation of Subsection 3.3.3 on the forward workers with eight subhorizons, and the network separation of Subsection 3.3.2 on all other forward workers. As for the backward workers, we apply the network separation only on those workers with four or fewer subhorizons. We elaborate on our choices in Section 5 with support from empirical results.

5. Results

Table 3 shows the bounds, gaps and running times for the 20 cases solved with `Gurobi`. As shown in this table, the running times vary widely from case to case. Nevertheless, apart from cases 2, 3 and 4, `Gurobi` solves all cases in little more than one hour. The running time for Case 3 stands out as it took `Gurobi` more than 27 hours to reach the required gap. Inspecting the solver's log, we see that `Gurobi` spent most time in the crossover phase of the root relaxation, and later solving the nodes in the branch-and-bound tree. Spending too much time in the crossover phase is usually a sign of numerical problems. (Note that decomposition approaches become even more appealing when dealing with numerically challenging problems since solvers generally handle such models more easily when their sizes is reduced.) The results in Tab. 3 are also important as benchmarks for the quality of the solutions obtained with the proposed approaches.

Table 3: Results for `Gurobi` with a single subhorizon.

Case	LB (10^6 \$)	UB (10^6 \$)	Gap (%)	Time (min)
1	92,182.68	92,182.69	$1.66 \cdot 10^{-5}$	9
2	106,282.71	106,282.79	$7.07 \cdot 10^{-5}$	180
3	61,459.80	61,459.81	$2.7 \cdot 10^{-5}$	1,674 (27.4 hours)
4	60,520.51	60,541.22	$3.4 \cdot 10^{-2}$	330 (5.5 hours)
5	60,882.41	60,882.41	$1.18 \cdot 10^{-6}$	12
6	57,619.38	57,619.38	$1.69 \cdot 10^{-11}$	14
7	83,437.77	83,437.77	$1.49 \cdot 10^{-11}$	8
8	54,053.69	54,053.69	$3.53 \cdot 10^{-11}$	11
9	54,828.72	54,828.76	$6.08 \cdot 10^{-5}$	31
10	92,879.06	92,879.79	$7.8 \cdot 10^{-4}$	12
11	50,870.74	50,870.74	$1.56 \cdot 10^{-7}$	50
12	51,140.11	51,140.19	$1.65 \cdot 10^{-4}$	61
13	73,599.52	73,599.55	$3.38 \cdot 10^{-5}$	22
14	73,472.08	73,472.08	$5.47 \cdot 10^{-6}$	14
15	86,878.18	86,878.18	$7.23 \cdot 10^{-7}$	13
16	68,736.85	68,736.86	$4.64 \cdot 10^{-6}$	11
17	72,746.02	72,746.03	$5.82 \cdot 10^{-6}$	8
18	53,490.43	53,490.46	$5.01 \cdot 10^{-5}$	21
19	72,017.15	72,017.17	$2.57 \cdot 10^{-5}$	65
20	71,769.36	71,769.37	$1.33 \cdot 10^{-5}$	55

As we have done with `Gurobi` in Tab. 3, we present the main results for `Cbc` in Tab. 4. For `Cbc`, we have set a time limit of three hours for the solution of the root relaxation and a limit of another three hours for `Cbc` in its branch-and-cut algorithm. Nonetheless, the time-limit checks of `Cbc` are infrequent, thus the discrepancies seen in Tab. 4. As expected, the performance of `Cbc` is far from `Gurobi`'s. For cases 5, 14, 15, 17, 18, 19, and 20 for which `Cbc` meets the required gap, it takes, respectively, 10.7, 7.9, 4.9, 6.3, 7.5, 19 and 6.2 hours. Evidently, these running times are far from acceptable for a problem that must be solved daily for the day-ahead scheduling. Thus, decomposition methods e.g. DDiP are appealing strategies for solving large-scale UCs with free solvers, and even indispensable ones, given the current capabilities of free solvers.

In [10], we have empirically shown through extensive numerical analyses that finding a single configuration of subhorizons that performs well in the sequential dual dynamic integer programming strategy over a large number of instances of a problem is intricate. Although the problem in this work is simpler than the problem tackled in [10], finding a good subhorizon configuration remains challenging — and this is exacerbated here due to our goal of using a free solver. Therefore, based on the results of [10], we do not analyse here the performance of `Cbc` over the 20 cases with different configurations of subhorizons, but instead we focus on Case 1, also relying on the

Figure 4: Sharing points for the approximations of the cost-to-go functions for a problem with 48 periods and workers with two, four, eight, 16 and 24 subhorizons. The vertical red bars indicate points at which the approximations of the cost-to-go function can be shared. For instance, the approximation for the optimal cost of the last 24 periods can be shared among all workers: workers with four subhorizons will add this approximation to their second subhorizon; those with eight subhorizons will add it to their fourth subhorizon.

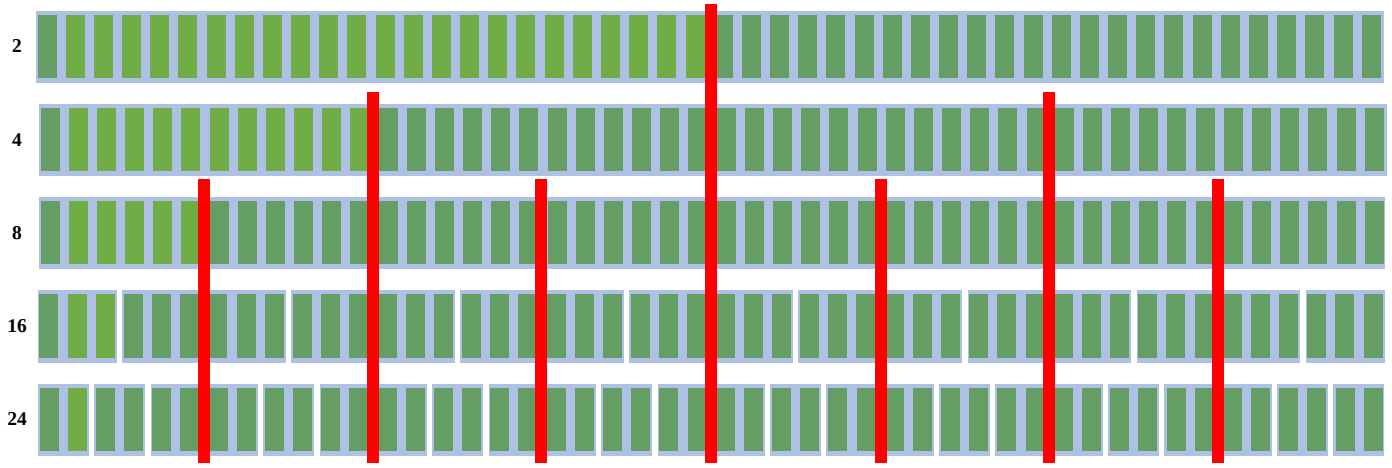


Table 4: Results for Cbc with a single subhorizon.

Case	LB (10^6 \$)	UB (10^6 \$)	Gap (%)	Time (hour)
1	92,182.676	-	-	5.5
2	106,282.69	-	-	6
3	61,459.798	-	-	6
4	60,520.511	-	-	5.4
5	60,882.406	60,906.339	$3.9 \cdot 10^{-2}$	10.7
6	57,619.381	-	-	6
7	83,437.77	-	-	6.3
8	54,053.69	-	-	6
9	54,828.718	-	-	5.7
10	92,879.047	-	-	7.1
11	50,870.739	-	-	6.5
12	51,140.093	-	-	6
13	73,599.513	-	-	6
14	73,472.075	73,472.198	$1.7 \cdot 10^{-4}$	7.9
15	86,878.176	86,878.772	$6.8 \cdot 10^{-4}$	4.9
16	68,736.851	-	-	6
17	72,746.021	72,746.037	$2.2 \cdot 10^{-5}$	6.3
18	53,490.424	53,490.488	$2.1 \cdot 10^{-5}$	7.5
19	72,017.151	72,017.178	$1.5 \cdot 10^{-5}$	19
20	71,769.35	71,769.915	$1.5 \cdot 10^{-5}$	6.2

fact that most conclusions for this case can be extended for the other 19.

Results for the first two iterations of the sequential dual dynamic integer programming with Cbc, with no inner decomposition, applied to Case 1 are shown in Tab. 5. As we have seen in [10], the upper bounds obtained with the different subhorizon configurations are similar, whereas the lower bounds tend to increase as the number of subhorizons is decreased. Also, as we decrease the number of subhorizons, both the forward and backward times increase significantly. It is evident from Tab. 5 that solving the problem at hand with the sequential dual dynamic integer programming strategy and Cbc would be severely obstructed by the time that Cbc takes to solve the mixed-integer subhorizon problems in the forward step. With 24 subhorizons, Cbc requires, on average, 5 min to perform a single forward step for Case 1. We also see in Tab. 5 that Cbc struggles with the large linear programming problems of the backward steps with few subhorizons. This difficulty can hinder the use of Cbc in our parallel strategies since they rely on the subgradients and lower bounds provided by the backward step with few subhorizons. The results for shown in Tab. 5 for the dual dynamic integer programming strategy with no inner decomposition motivate the development of the inner decompositions of Subsections 3.3.1, 3.3.2 and 3.3.3, their results for the same problem are also presented in Tab. 5.

The results from Tab. 5 suggest that the inner-decomposition approaches do not significantly affect the upper bounds obtained in the sequential dual dynamic integer programming, with the largest difference seen for the configuration with 48 subhorizons and the network separation (the upper bound obtained with this inner decomposition is 0.2% greater than the one with no inner decomposition). On the other hand, all inner decompositions provide substantial reductions in forward times, specially as the number of subhorizons is decreased. In contrast to the forward times, the benefits of using an inner decomposition in the backward step, specifically the network separation of Subsection 3.3.2, only becomes beneficial with 16 or fewer subhorizons. Nevertheless, it is important to emphasize that, due to the iterations limits imposed on the inner-decomposition approaches, the respective forward and backward subhorizons are not necessarily solved to optimality, or even to the 0.01% gap tolerance of the forward subhorizon problems. Thus, it is likely that if we were to tighten the convergence criteria for the inner decompositions to meet the outputs obtained for the approach with no inner decomposition, the speed-ups seen in

Table 5: Main results for the sequential dual dynamic integer programming with Cbc and inner decompositions applied to the subhorizon problems of Case 1. For different numbers of subhorizons (NS), from 48 to 2, we test five approaches of inner decompositions: no inner decomposition, in which Cbc is directly used for solving the subhorizons; the integer separation approach introduced in Subsection 3.3.1 applied to the subhorizon problems in the forward step; the network separation of Subsection 3.3.2 again applied in the forward step; the double inner decomposition of Subsection 3.3.3 in the forward step; and finally the double decomposition applied in the forward subhorizon problems combined with the network separation applied to the backward subhorizon problems. For all approaches, we set a maximum number of iterations of two, and a total time limit of three hours. The forward (F) and backward (B) times reported are the total times spent in these steps in the two iterations. Only the results for the "No inner decomposition" approach are given in absolute values, all others are given in relative terms of those. For instance, for 48 subhorizons, using the integer separation for solving the subhorizon problems yields an upper bound (10^6 \$ 92,258) after the two iterations that is only 0.01% greater than the bound given with no inner decomposition, and the elapsed time in the forward steps for the two iterations is reduced by 53% to 2.3 min. For the setting with two subhorizons and no inner decomposition, Cbc is only able to finish the forward step of the first iteration, thus the gap and backward's time information are not available.

	No inner decomposition				**Integer separation**				**Network separation**				**Double separation**				**Double, and backward with network separation**			
NS	UB	Gap	Time (min)		UB	Gap	Time (%)		UB	Gap	Time (%)		UB	Gap	Time (%)		UB	Gap	Time (%)	
	(10^6 \$)	(%)	F	B	(%)	(%)	F	B	(%)	(%)	F	B	(%)	(%)	F	B	(%)	(%)	F	B
48	92,251	1.08	5	1	0.01	66	-53	22	0.20	41	-52	17	0.19	8	-55	27	0.19	18	-55	20
24	92,252	0.33	10	1	0.01	29	-79	5	0.03	2	-73	8	0.06	39	-82	10	0.06	44	-82	3
16	92,244	0.26	15	2	0.02	49	-84	3	0.03	6	-81	12	0.05	42	-89	5	0.05	82	-89	-36
12	92,240	0.25	20	2	0.02	46	-85	6	0.02	2	-87	3	0.05	53	-92	10	0.05	55	-92	-48
8	92,235	0.22	32	4	0.01	39	-87	-1	0.03	20	-90	12	0.04	49	-94	-10	0.04	189	-94	-70
6	92,230	0.18	50	8	0.03	28	-87	-7	0.03	10	-92	-5	0.03	33	-95	-13	0.03	285	-95	-83
4	92,227	0.24	193	6	0.02	27	-93	87	0.03	-27	-96	86	0.01	1	-98	85	0.01	118	-98	-70
3	92,222	0.52	160	15	-0.01	-63	-89	29	0.03	-2	-93	-5	0.00	-62	-96	20	0.00	-7	-95	-85
2	92,227	100	268	-	-0.03	-	-72	-	-0.02	-	-92	-	-0.02	-	-96	-	-0.02	-	-95	-

Tab. 5 would be reduced. Moreover, we can infer from the differences in the gaps and upper bounds obtained that the lower bounds are the most affected by the sub-optimality of the inner decompositions' solutions. For example, the gap for the setting with six subhorizons increases by 28% with the integer separation (to 0.23%), 10% with the network separation, 33% with the double separation, and 285% with the double separation in the forward step combined with network separation in the backward (to 0.69%). Therefore, employing the inner decompositions only becomes advantageous if the time reductions, and corresponding increased number of iterations that can be performed in the same amount of time, outweigh the diminished quality of subgradients and lower bounds obtained.

When choosing the inner decomposition, one must consider not only their individual characteristics but also the context on which they are to be applied. In this work, our objective is to find forward and backward configurations that can best utilize the proposed parallel dual dynamic integer programming approaches. To that end, we look for forward configurations with possibly many subhorizons that offer a good trade-off of quality upper bounds and elapsed time, and similarly for backward configurations that timely yield tight bounds. It is also important to choose configurations that can provide robust results across a variety of instances of the same problem, as opposed to one that would work well only on a small subset of instances. Thus, for the forward workers, as we mentioned earlier, whose numbers of subhorizons are 24, 16 and eight, we apply the network separation for those with 24 and 16 subhorizons, and the double separation for those with eight subhorizons. The network separation is applied in the backward steps of those backward workers whose number of subhorizons is fewer than or equal to four, namely, those backward workers with four and two subhorizons. In the following, we analyse the results obtained with the asynchronous and synchronous approaches with Cbc and the inner decompositions.

Table 6 shows that the proposed asynchronous and synchronous strategies are successful in solving the hydrothermal UC tackled in this work in less than one hour to a gap of 0.1%. As expected, the asynchronous strategy outperforms the synchronous one in all cases in terms of running time, even in its worst runs. The greatest difference in the running times of the synchronous and asynchronous strategies happens for Case 5: while the synchronous approach takes about 23 min to reach the stopping criterion, in its worst run, the asynchronous strategy reaches the gap threshold in 4.8 min (a speed-up of 79%). Additionally, it is interesting to see that despite the non-determinism of the asynchronous strategy, the differences in the running times over the five runs are not significant in absolute terms, although in relative terms these differences are substantial (see Case 13 for example). However, more important than the differences in running times, except for cases 4, 5, 7 and 8, for all other cases there are differences in the upper bounds over the five runs, which implies differences in the solutions obtained by the asynchronous strategy. In contrast, as mentioned before, the synchronous strategy yields the same results in all five runs of each case — although not shown here, we have computed the Euclidean distance between the solutions given by the synchronous method over the five runs to ensure that not only the upper bounds are the same but also that the solutions are in fact identical. We in this table that, whereas there is no clear indication of which strategy provides the best lower bounds, for the majority of cases, the upper bounds given by the asynchronous ones are slightly better, which is likely a consequence of the greater number of solutions that can be generated in this strategy.

If we treat the lower bounds of Tab. 3 yielded by Gurobi as the optimal values of the respective problems, and compare it against the bounds shown in Tab. 6, we observe that, generally, the lower bounds are significantly closer to the optimal values, for both strategies, than the upper bounds. On average,

the lower bounds provided by both methods are only 0.02% smaller than the optimal value, whereas the upper bounds are, on average, about 0.06% greater than the optimal values, again for both strategies. This suggests that future research, aiming at further reducing the gap, has a larger margin of improvement in the upper bounds.

6. Conclusion

We have illustrated with empirical results how decomposition approaches can be used to potentialize free and open-source solvers by solving several instances of a large-scale hydrothermal unit-commitment problem with both asynchronous and synchronous parallel dual dynamic integer programming strategy, further furnished with inner decompositions, with `Cbc`. Our results show that, when combined with decomposition, open sources are promising alternatives for solving real-life unit-commitment problems. Expanding the results of our previous work [10], we devised a synchronous parallel dual dynamic integer programming strategy capable of deterministically solving the problem's instances used in this work, as opposed to the non-deterministic asynchronous version that tends to produce different results every time it solves the same instance, even under the same computational settings. Due to its deterministic characteristic, our synchronous algorithm is more suitable to practical implementations and shows that parallelization, decomposition and free solvers can be combined to solve real problems, while satisfying strict requirements of reproducibility and running times.

Our experience of using the solver `Cbc` to solve a large-scale unit-commitment problem is encouraging and shows the potential of employing `Cbc` to unit-commitment problems beyond academia. Thus, as possible future research, we can further improve the unit-commitment model to test the limits of `Cbc` and our own methodology, as well as assess it on different test systems. Moreover, it would also be interesting to see how other free and open-source solvers perform with our parallel dual dynamic integer programming strategies. Given the recent developments in cloud computing, testing our solution on a cloud environment could also provide interesting results.

Acknowledgements

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Award Number 001, and by Norte Energia through the Agência Nacional de Energia Elétrica (ANEEL) R&D Project PD-07427-0318/2018.

Table 6: Results for the asynchronous and synchronous strategies with the free and open-source solver `Cbc`. For the asynchronous strategy, we give the best lower bound (LB) over the five runs, the best upper bound (UB), best gap, and best running times. For all these metrics, we give in parentheses the difference between the best value and the worse value in each case. For instance, for the LB for Case 1, the best LB given by the asynchronous strategy is 10^6·\$92,157.60, while the worst one for the same case over the five runs is 10^6·(\$92,157.60 -\$2.3). Because the synchronous strategy gives the same results in all five runs, the only differences appear in the running times.

Case	**Asynchronous**				**Synchronous**			
	LB (10^6 \$)	UB (10^6 \$)	Gap (%)	Time (min)	LB (10^6 \$)	UB (10^6 \$)	Gap (%)	Time (min)
1	92,157.60 (-2.3)	92,242.42 (6.42)	0.090 (0.006)	9.0 (1.8)	92,168.73	92,249.80	0.088	28.1 (0.1)
2	106,259.49 (-5.57)	106,352.42 (9.79)	0.090 (0.007)	8.5 (1.3)	106,264.42	106,345.34	0.076	29.9 (0.2)
3	61,447.58 (-1.37)	61,500.79 (4.04)	0.090 (0.008)	7.5 (3)	61,446.00	61,503.47	0.093	13.1 (0)
4	60,505.63 (-0.47)	60,562.81 (0)	0.090 (0.001)	3.9 (0)	60,502.31	60,562.81	0.1	10.6 (0.1)
5	60,867.65 (-0.24)	60,927.05 (0)	0.100 (0)	4.7 (0.1)	60,867.22	60,927.05	0.098	23.0 (0.1)
6	57,608.77 (-6.22)	57,657.01 (4.99)	0.090 (0.007)	8.7 (3.6)	57,600.29	57,653.42	0.092	13.4 (0.1)
7	83,418.88 (-4.31)	83,497.62 (0)	0.090 (0.005)	4.1 (0.5)	83,414.88	83,497.62	0.099	13.6 (0)
8	54,043.46 (-7.65)	54,087.71 (6.39)	0.090 (0.006)	14.1 (4.6)	54,036.57	54,089.22	0.097	48.9 (0.1)
9	54,817.19 (-5.54)	54,866.18 (2.72)	0.090 (0.01)	13.4 (5)	54,811.25	54,864.62	0.097	49.2 (0.2)
10	92,855.12 (-2.53)	92,943.90 (2.79)	0.100 (0.003)	13.6 (3.3)	92,864.86	92,950.09	0.092	31.5 (0.1)
11	50,860.86 (-7.15)	50,901.67 (9.91)	0.090 (0.008)	11.8 (4.5)	50,859.52	50,909.68	0.099	35.1 (0.1)
12	51,124.39 (-0.83)	51,167.20 (7.67)	0.080 (0.015)	7.7 (1.2)	51,127.79	51,172.48	0.087	11.7 (0)
13	73,579.73 (-2.13)	73,645.25 (4.31)	0.090 (0.006)	7.4 (5.5)	73,580.40	73,646.98	0.09	29.2 (0.1)
14	73,450.46 (-3)	73,513.90 (9.74)	0.090 (0.01)	10.0 (4.3)	73,455.16	73,522.76	0.092	22.3 (0.1)
15	86,851.54 (-6.89)	86,920.57 (11.09)	0.090 (0.01)	11.9 (4.4)	86,861.80	86,939.95	0.09	27.0 (0.2)
16	68,725.57 (-11.69)	68,782.08 (10.03)	0.090 (0.006)	12.9 (3.7)	68,719.81	68,786.81	0.097	23.8 (0)
17	72,731.24 (-11.69)	72,784.73 (13.58)	0.090 (0.011)	7.6 (2.9)	72,724.11	72,795.58	0.098	23.2 (0)
18	53,480.89 (-6.33)	53,526.08 (3.89)	0.090 (0.011)	8.8 (3.3)	53,476.78	53,523.48	0.087	13.6 (0.1)
19	71,994.25 (-4.08)	72,055.34 (7.17)	0.090 (0.01)	4.5 (1.4)	71,997.88	72,064.77	0.093	8.3 (0)
20	71,739.41 (0)	71,810.33 (0)	0.100 (0)	3.2 (0)	71,739.41	71,810.33	0.099	3.2 (0)

References

[1] ApS, M., 2022. MOSEK Command Line Tools. Version 9.3.18.
URL: https://docs.mosek.com/9.3/cmdtools.pdf

[2] Ardakani, A. J., Bouffard, F., 2013. Identification of umbrella constraints in dc-based security-constrained optimal power flow. IEEE Transactions on Power Systems 28 (4), 3924–3934.

[3] Atabay, D., 2017. An open-source model for optimal design and operation of industrial energy systems. Energy 121, 803–821.

[4] Benders, J. F., Dec. 1962. Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4 (1), 238–252.
URL: https://doi.org/10.1007/bf01386316
DOI: 10.1007/bf01386316

[5] Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., et al., 2021. The scip optimization suite 8.0. arXiv preprint arXiv:2112.08872.
URL: https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/8530

[6] Brown, T., Hörsch, J., Schlachtberger, D., 2018. PyPSA: Python for Power System Analysis. Journal of Open Research Software 6 (4).
DOI: 10.5334/jors.188

[7] Bukhsh, W., Edmunds, C., Bell, K., 2020. Oats: Optimisation and analysis toolbox for power systems. IEEE Transactions on Power Systems 35 (5), 3552–3561.
DOI: 10.1109/TPWRS.2020.2986081

[8] COIN-OR Foundation, 2020. Coin-or/Cbc: Version 2.10.5.
URL: https://github.com/coin-or/Cbc
DOI: 10.5281/zenodo.3700700

[9] COIN-OR Foundation, 2020. Coin-or/clp: Version 1.17.6.
URL: https://github.com/coin-or/Clp
DOI: 10.5281/zenodo.5839302

[10] Colonetti, B., Finardi, E., Brito, S., Zavala, V., 2022. Parallel dual dynamic integer programming for large-scale hydrothermal unit-commitment.

[11] Corp., G. D., 2022. Gams documentation center.
URL: https://www.gams.com/latest/docs/

[12] Cplex, IBM ILOG, 2022. IBM ILOG CPLEX Optimization Studio documentation.
URL: https://www.ibm.com/analytics/cplex-optimizer

[13] Dalcin, L., Fang, Y.-L. L., 2021. mpi4py: Status update after 12 years of development. Computing in Science Engineering 23 (4), 47–54.
DOI: 10.1109/MCSE.2021.3083216

[14] Dubost, L., Gonzalez, R., Lemaréchal, C., 2005. A primal-proximal heuristic applied to the french unit-commitment problem. Mathematical programming 104 (1), 129–151.
DOI: 10.1007/s10107-005-0593-4

[15] Eclipse Foundation, 2022. Eclipse public license version 2.0.
URL: https://opensource.org/licenses/EPL-2.0

[16] Fair Isaac Corporation, 2022. FICO Xpress Optimization Suite.
URL: https://www.fico.com/en/products/fico-xpress-optimization

[17] Global PST Consortium, 2021. Global Power System Transformation Consortium.
URL: https://www.nrel.gov/docs/fy21osti/79719.pdf

[18] Greenhall, A., 2021. Minpower.
URL: https://github.com/adamgreenhall/minpower

[19] Groissböck, M., 2019. Are open source energy system optimization tools mature enough for serious use? Renewable and Sustainable Energy Reviews 102, 234–248.
DOI: 10.1016/j.rser.2018.11.020

[20] Gurobi Optimization, LLC, 2022. Gurobi Optimizer Reference Manual.
URL: https://www.gurobi.com

[21] Hilpert, S., Kaldemeyer, C., Krien, U., Günther, S., Wingenbach, C., Plessmann, G., 2018. The open energy modelling framework (oemof)-a new approach to facilitate open science in energy system modelling. Energy strategy reviews 22, 16–25.
DOI: 10.1016/j.esr.2018.07.001

[22] Hirth, L., 2007. The european electricity market model: Emma.
URL: https://neon.energy/emma-documentation.pdf

[23] Hunter, K., Sreepathi, S., DeCarolis, J. F., 2013. Modeling for insight using tools for energy model optimization and analysis (temoa). Energy Economics 40, 339–349.
DOI: 10.1016/j.eneco.2013.07.014

[24] Kavvadias, K., Hidalgo Gonzalez, I., Zucker, A., Quoilin, S., 2018. Integrated modelling of future eu power and heat systems-the dispa-set v2. 2 open-source model. Tech. rep., European Commission.

[25] Kim, K., Botterud, A., Qiu, F., 2018. Temporal decomposition for improved unit commitment in power system production cost modeling. IEEE Transactions on Power Systems 33 (5), 5276–5287.
DOI: 10.1109/TPWRS.2018.2816463

[26] Knueven, B., Ostrowski, J., Watson, J.-P., 2020. On mixed-integer programming formulations for the unit commitment problem. INFORMS Journal on Computing 32 (4), 857–876.
DOI: 10.1287/ijoc.2019.0944

[27] Krishnamurthy, D., 2016. psst: An open-source power system simulation toolbox in python. En: 2016 North American Power Symposium (NAPS). IEEE, pp. 1–6.

[28] Loulou, R., Goldstein, G., Kanudia, A., Lettila, A., Remme, U., 2016. Documentation for the TIMES Model: PART I.
URL: https://iea-etsap.org/docs/Documentation_for_the_TIMES_Model-Part-I_July-2016.pdf

[29] Musto, M., 2020. Day ahead network constrained unit commitment performance.
URL: https://www.ferc.gov/sites/default/files/2020-06/T2-2_Musto_0.pdf

[30] Ott, A. L., 2010. Evolution of computing requirements in the pjm market: Past and future. En: IEEE PES General Meeting. pp. 1–4.
DOI: 10.1109/PES.2010.5589842

[31] Pfenninger, S., 2017. Energy scientists must show their workings. Nature 542 (7642), 393–393.
DOI: 10.1038/542393a

[32] Pfenninger, S., Pickering, B., 2018. Calliope: a multi-scale energy systems modelling framework. Journal of Open Source Software 3 (29), 825.
DOI: 10.21105/joss.00825

[33] Saltzman, M. J., 2002. Coin-or: an open-source library for optimization. En: Programming languages and systems in computational economics and finance. Springer, pp. 3–32.

[34] Santos, T., Diniz, A., Saboia, C., Cabral, R., Cerqueira, L., 2020. Hourly pricing and day-ahead dispatch setting in brazil: The dessem model. Electric Power Systems Research 189, 106709.

[35] Sheble, G., Fahd, G., 1994. Unit commitment literature synopsis. IEEE Transactions on Power Systems 9 (1), 128–135.
DOI: 10.1109/59.317549

[36] Wächter, A., Biegler, L. T., 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming 106 (1), 25–57.
DOI: 10.1007/s10107-004-0559-y

[37] Xavier, A. S., Qiu, F., Dey, S. S., 2020. Decomposable formulation of transmission constraints for decentralized power systems optimization. arXiv preprint arXiv:2001.07771.
DOI: 10.48550/arXiv.2001.07771

[38] Zimmerman, R. D., Murillo-Sánchez, C. E., 2020. Matpower optimal scheduling tool (most) user's manual, version 1.1.
URL: https://matpower.org/docs/MOST-manual-1.1.pdf
DOI: 10.5281/zenodo.4073878

[39] Zou, J., Ahmed, S., Sun, X. A., Mar. 2018. Stochastic dual dynamic integer programming. Mathematical Programming 175 (1-2), 461–502.
URL: https://doi.org/10.1007/s10107-018-1249-5
DOI: 10.1007/s10107-018-1249-5

[40] Zuse Institute Berlin, 2022. Zib academic license.
URL: https://scipopt.org/academic.txt