Dissertação de Mestrado para obtenção do grau de Mestre em Matemática (Opção Matemática Computacional e Modelagem) pelo Instituto Nacional de Matemática Pura e Aplicada

Programação Dinámica para "Unit Commitment"Térmico com ou sem restrições de Rampa.

María Gabriela Martínez López

Orientador: Prof.Alfredo Noel Iusem
Co-orientadora: Prof.Claudia Alejandra Sagastizábal

19 de Junho de 2007

Agradeçimentos

Agradeço a Claudia pela excelente orientação desta dissertação, em particular meus sinceros agradeçimentos por ter revisado o material escrito em tempo record o que fez possível que terminasse este trabalho quando o tinha previsto, pelas sugestões feitas que contribuíram valiosamente para melhorar a qualidade desta dissertação e da minha exposição no dia da defensa, e por ter se mostrado sempre disposta à me ajudar em tudo o que for possível.

Agradeço a André pelas sugestões feitas que também foram importantes para melhorar esta dissertação, por sua colaboração ao longo da minha estadía no CEPEL esclarecendo dúvidas, me indicando referências novas do problema, modificando o DESSEM para que meus códigos possam ser incorporados e por ter se mostrado sempre interessado com meu trabalho.

Agradeço ao CEPEL (Centro de Pesquisas de Energia Elétrica) pelo auxílio financiero concedido, estos últimos seis meses, e pela disponibilização de suas instalações para realizar esta dissertação. Agradeço também ao IMPA pela formação acadêmica e a CAPES pelo auxílio financiero concedido.

Abstract

This work addresses the modeling and solution of a specific optimization subproblem appearing in the short-term planning model. More precisely, we consider local thermal subproblems of the *Security Constrained Thermal Unit Commitment* ($SCTUC_H$) problem, which is a short-term planning model developed by CEPEL (Centro de Pesquisas de Energia Elétrica). The $SCTUC_H$ problem consists in determining an optimal scheduling for a hydrothermal mix, as well as the corresponding production level for each generating unit in the mix, over a planning horizon of typically one day or one week, so that the resulting total system costs are minimized. At each time step, the total output of all production units in the power system must be equal to the system demand at each time step, data considered to be deterministic in this work. Also, since electricity cannot be stored, a power system must always be able to increase its production in the running units to cover unexpected peaks in the demand or unscheduled stops at production units. In the short-term model, such *reserve* requirement of energy is represented by a set of constraints known as *Spinning reserve constraints*.

For thermal units, optimal schedules and production levels obtained in the short term planning have to satisfy very specific operating rules, including start-up and shutdown processes, as well as minimum up/down times and ramping constraints. From the optimization point of view, problem $SCTUC_H$ is large-scale, non-convex, and in mixed integer variables. In energy planning, Lagrangian Relaxation (LR) is a decomposition technique which has become one of the most commonly used strategies of resolution for the short-term planning problems, [1, 2, 3]. Roughly speaking, LR includes some constraints together with corresponding Lagrangian multipliers into the objective function, so that the original problem decomposes into several independent subproblems of lower dimension called *Local Problems*. In particular, for the short-term planning model, the classi-

cal way to achieve decomposition of the $SCTUC_H$ problem is to relax the demand and reserve constraints, [2, 4, 7]. An alternative decomposition is to duplicate some variables and relax the corresponding equality constraints, [1, 4]. In both cases, the thermal unit commitment problem, called *Local Thermal Problem* (LTP) and subject of this dissertation, is one of the local problems obtained after the decomposition.

This work analyzes two different modelling and solution methods for the LTP, which can be seen as an optimization problem over a graph. To assess the proposed algorithm, some numerical experience is presented for the $SCTUC_H$ problem for the Brazilian power system.

Contents

Chapter 1

Introduction

An electrical power system consists of power generating plants, transmission lines, and tie lines connecting one system to neighboring ones. Power enters the transmission system from nuclear, hydro and/or thermal plants and leaves the system through loads, transfers to neighboring systems, and electrical losses.

1.1 General Context: Power Planning

In a vertically integrated system, the primary objective of power system operation -referred to on the sequel as power planning-, is to ensure that users' demand is met at the lowest cost [9]. This objective explicitly specifies an optimization problem with a cost function to be minimized and a variety of constraints describing the physical system and limits on acceptable performance. Meeting this objective by properly controlling the individual components of the power system is a complex task.

One of the difficulties associated with power planning is the physical size of the system. The network may have several thousands nodes (buses), lines and the generation mix may include a large number of hydro-plants and/or thermal plants. Another major difficulty in dealing with electrical power systems is the vast range of time intervals over which various processes need to be controlled. For this reason, the whole planning problem is usually divided into a hierarchy of problems according to the length of the considered planning period, [7, 11]. Essentially speaking, the hierarchy of planning problems is as follows: *Expansion planning* is related to

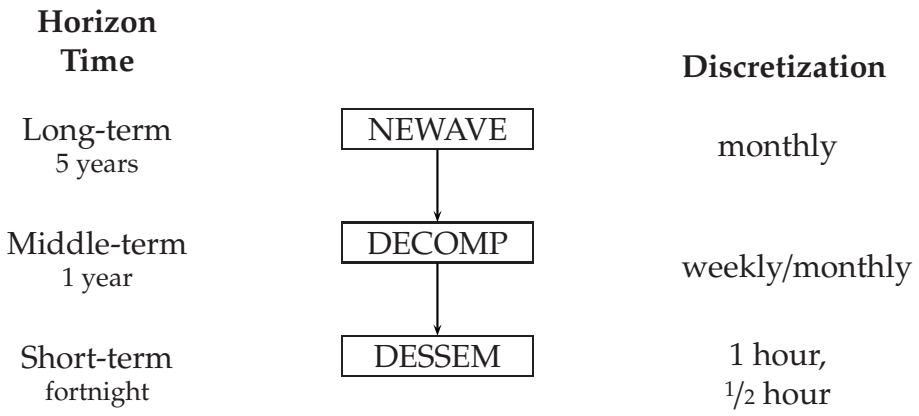

Figure 1.1: Chain of models developed by CEPEL for planning and programming of SIN.

the problem of when and where to build new power plants to meet future energy demand, over an horizon of few decades; this planning is often approached by simulation or stochastic programming. *Long-term planning* of hydrothermal systems include multi-year planning of large reservoirs and thermal plants, in a stochastic setting for water inflows. *Medium-term planning* determines the weekly/monthly scheduling for each power plant by using a scenario tree, composed of many nodes with different hydrological conditions. Finally, *short-term planning* deals with a planning horizon of up to a couple of weeks, usually with a time resolution of one hour. The output of each generating unit in the system is determined for each time period. For such relatively short time ranges, it is possible to make quite accurate predictions of both the hourly demand for power and the natural reservoir inflows. These predictions are based on statistics from earlier days/weeks for the inflows, weather forecast, and known peaks of demand of energy. This partly motivates a deterministic approach to short term power optimization problems in vertically integrated systems.

For the Brazilian National Interconnected System (SIN), the chain of planning models developed by CEPEL (Centro de Pesquisas de Energia Elétrica) and their respective focus is reported in Figure 1.1. Expansion planning is considered separately by a fourth model, called MELP, also developed by CEPEL.

This work addresses the modeling and solution of a specific optimiza-

tion subproblem appearing in the short-term planning model. More precisely, we consider local thermal subproblems of the *Security Constrained Thermal Unit Commitment* ($SCTUC_H$) problem in hydrothermal systems. The $SCTUC_H$ problem consists in determining an optimal scheduling for a hydrothermal mix, as well as the corresponding production level for each generating unit in the mix, over a planning horizon of typically from one day to one week, so that the resulting total system costs are minimized. The schedules and the production levels have to satisfy the demand of energy (with representation of the electrical network), and spinning reserve constraints. Operational constraints of the units, usually coupling different time steps, also have to be taken into account by the optimization problem.

In particular, for the Brazilian case, solving the $SCTUC_H$ problem means to consider simultaneously, [1], section 1:

- thermal power units with high start-up/shutdown costs,
- hydro-plant of interconnected reservoirs, leading to tight dynamic constraints,
- hydraulic plants with nonlinear production functions depending on the water head and outflows through turbines and spillways,
- various types of integral and logical constraints.

The corresponding optimization problem is large-scale, non-convex, in mixed integer variables. A popular approach to solve this type of mathematical programs is Lagrangian Relaxation, also called price decomposition. In energy planning, in particular, Lagrangian Relaxation (LR) is a decomposition technique which has become one of the most commonly used strategies of resolution for short-term planning problems, [1, 2, 4, 7]. Roughly speaking, LR includes some constraints together with corresponding Lagrangian multipliers into the objective function, so that the original problem decomposes into a sequence of several independent subproblems of lower dimension, called *Local Problems*. For the short-term planning model, the classical way to achieve decomposition for the $SCTUC_H$ problem is to relax the demand and reserve constraints, [2, 4, 7]. An alternative decomposition is to duplicate some variables and relax the corresponding equality constraints, [1, 4]. In both cases, the thermal unit commitment problem, called *Local Thermal Problem* (LTP) and subject of this dissertation, is one of the local problems obtained after the decomposition.

1.2 Local Thermal Problem and Contributions of this work

This work considers the LTP, a problem of optimization over a graph that appears when solving short-term planning models by Lagrangian Relaxation. This problem is related to the so called "Thermal Unit Commitment" problem, that analyzes in a detailed manner operational constraints of thermal units or generators.

One thermal plant is usually composed of several units, that have to be switched on or off according to the specific rules, as shown in Figure 1.2.

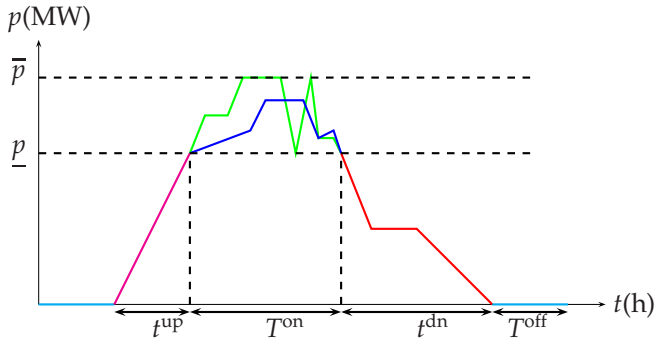

Figure 1.2: Example of Production Profile.

We explain in more details Figure 1.2. If the unit is off-line, its generation is zero, this is represented by the cyan line in the figure.

If we decide to start-up the unit, the generation follows a specific initial curve (magenta line) until is ready for full operation, when it is declared "on-line". The duration of such start-up process is t^{up}.

If the unit is on-line, it can operate freely, between its generation limits. In Figure 1.2, $\underline{p}$ and $\overline{p}$ correspond to these minimum and maximum production levels of the unit, respectively. If we do not constraint the variation of generation in time, an example of the production profile is given by the green line. If we constraint the variation of generationbetween consecutive time steps (a requirement called *ramping constraints* in the literature), an example of the production profile is given by the blue line in Figure 1.2. In this case, the variation of energy cannot be abrupt as it is in the green line. Finally, once the unit is on-line, it must remain on-line at least T^{on} hours

(this is called "minimum up time constraint").

If we decide to shutdown the unit, the generation follows a specific curve (red line). The duration of the shutdown process is t^{dn}, after completion of this process, the unit is considered to be "off-line" . Once the unit is off-line, it must remain off-line at least T^{off} hours (this is called "minimum down time constraint").

Based on previous works, [1, 4, 6], to solve LTP two different graphs and dynamic programming algorithms were devised. The modeling and implementation developed for this work allow to consider simultaneously a variety of important features of thermal plants, including

- Variable Start-up costs.
- Minimum up/down times.
- Specific start-up and shutdown curves, known a priori, with accurate estimation of the generation cost curves when warming/cooling the unit.
- Ramping Constraints.
- Truncated start-up/shutdown process at the end of the planning period.
- Forward Dynamic Programming techniques.

More details of these features are given in Chapter 3.

1.3 Organization

The manuscript is organized in 5 chapters and one appendix. **Chapter 2** presents the short-term planning model and two different decomposition schemes that yield the LTP. **Chapter 3** presents the LTP modeling and the developed solution methods. **Chapter 4** reports numerical results of the implemented algorithms in MATLAB. **Chapter 5** contains some conclusions and future work. Finally, **Appendix A** has the MATLAB codes of the algorithms.

Chapter 2

The Short-Term Planning Model

In this chapter, an abstract formulation of the SCTUC_{H} problem is given, as well as more details of the different decomposition schemes that which yield LTP problems, the main subject of this dissertation.

Section 2.1 gives a general formulation of the problem, and introduces some notation to be used in section 2.2, devoted to the mathematical formulation of the optimization problem. Finally, section 2.3 presents the space and space-time decomposition methodologies.

2.1 General Formulation

Consider a power mix with I power-generation units for a discretized horizon with time steps $\{1, \ldots, \mathcal{T}\}$. Here, the wording "hydro-unit" stands for a hydro-plant, while a "thermal unit" refers to a single generator (one thermal plant may have many thermal units)[1]. We denote by p_i^t the production level of unit i at time period $t \leq \mathcal{T}$, by $p^t = (p_1^t, \ldots, p_I^t)$ the production vector for the mix at time t, by $p_i = (p_i^1, \ldots, p_i^{\mathcal{T}})$ the production vector of unit i for the whole time horizon, by u_i^t, and $\tilde{u}_i^t$ the thermal binary variables, which represent if the thermal unit i is on-line or not at time step t. With this

[1]The individual modeling of hydro-units (i.e. each single turbine-generator group in a hydro plant), a problem known as "hydro-unit commitment", is not considered in this work.

notation, the SCTUC_H problem is given by:

$$\begin{cases} \min\limits_{u,\tilde{u},p} C(p) \\ p_i \in \mathcal{D}_i \qquad i \in I \\ p^t \in \mathcal{S}^t \qquad t \leq \mathcal{T}, \end{cases}$$

where $u, \tilde{u}$ are the matrices $\left\{u_i^t | i = 1, \ldots, I; t = 1, \ldots, \mathcal{T}\right\}$, and $\left\{\tilde{u}_i^t | i = 1, \ldots, I; t = 1, \ldots, \mathcal{T}\right\}$, repectively. p represents the matrix $\left\{p_i^t | i = 1, \ldots, I; t = 1, \ldots, \mathcal{T}\right\}$, $C(p)$ is the total production cost of the whole mix, the set $\mathcal{D}_i$ gathers the operating dynamic constraints of a single unit i, and the set $\mathcal{S}^t$ contains static constraints (such as satisfaction of demand, reserve, and network transmission), to be satisfied at each time step t by the whole mix.

We now give more details of the sets $\mathcal{D}_i$ and $\mathcal{S}^t$.

2.2 Mathematical Model

The development of the mathematical model used in this section is based on the paper [1] and the PhD. thesis [4]. As mentioned, for the Brazilian case, the SCTUC_H problem considers a mix with thermal units and hydro-plants. For notational convenience, we split p in thermal and hydraulic components, p_T and p_H, and denote by $C_T(p_T)$ and $C_H(p_H)$, respectively, the total thermal and hydraulic production cost over the time horizon.

The problem to be solved can be written in an abstract form as follows:

$$\min_{u,\tilde{u},p_T,p_H} C_T(p_T) + C_H(p_H) \tag{2.1}$$

subject to the following conditions:

a) System power balance (demand constraint)

$$\sum_{i=1}^{I_T} {p_T}_i^t + \sum_{j=1}^{I_H} {p_H}_j^t = D^t \quad t = 1, \ldots, \mathcal{T}, \tag{2.2a}$$

where I_T is the number of thermal units, I_H is the number of hydraulic units, ${p_{T/H}}_i^t$ is the power production level of (thermal/hydraulic) unit i at time step t, and D^t is the demand at time step t.

b) System spinning reserve requirements

$$\sum_{i=1}^{I_T} r_i^t + \sum_{j=1}^{I_H} (\overline{p}_{H_j} - p_{H_j}^t) \geq R^t, \quad t = 1, \ldots, \mathcal{T}, \tag{2.2b}$$

where

$$r_i^t = \begin{cases} u_i^t(\overline{p}_{T_i} - p_{T_i}^t), & \text{If ramping constraints are not considered,} \\ \min\{(u_i^t\overline{p}_{T_i} - p_{T_i}^t), \text{RU}\} & \text{If ramping constraints are considered.} \end{cases}$$

The binary control variable u_i^t describes whether the thermal unit i at time step t is on-line or off-line, $\overline{p}_{T_i}$ is the maximum power generation of thermal unit i, RU is the maximum increase in generation between consecutive time steps, $\overline{p}_{H_j}$ is the maximum power generation of hydraulic unit j, and R^t is the system spinning reserve at time step t.

c) Operating Thermal Dynamic Constraints $(\text{TC}(u, \tilde{u}, p_T))_{t \leq \mathcal{T}}$, to be detailed below, see chapter 3.

d) Operating Hydraulic Dynamic Constraints $(\text{HC}(p_H))_{t \leq \mathcal{T}}$. We refer to [4], chapter 5, for details.

Written respect to the sets $\mathcal{D}_i$ and $\mathcal{S}^t$, constraints (a) and (b) are static while constraints (c), and (d) are dynamic.

Since constraints (2.2) are linear, we can write them as $A_T p_T + A_H p_H = D$, $B_u u + B_T p_T + B_H p_H = R$, where $A_{T/H}$, $B_{u/T/H}$, D, R are matrices and vectors of appropriate dimension. The formulation in equality form of the spinning reserve constraint is obtained by introducing slack variables which represent a fictitious unit with unbounded negative contribution for the system reserve and which is not taken into account for satisfying the demand constraints (2.2a).

2.3 Decomposition Scheme

Due to the presence of multiple coupling constraints, decomposition techniques are a natural approach for the SCTUC_H problem. Lagrangian Relaxation (LR) appears in the literature as one of the most powerful decomposition techniques to solve large scale systems, [2, 4, 5, 7].

The classical way to apply LR to decompose the problem consists in relaxing demand and reserve constraints, i.e., (2.2). We will call this decomposition scheme *space decomposition*, because variables are coupled along the "space" of units composing the mix (cf.(2.3) below).

In the SCTUC_{H} problem, the naming "security constrained" refers to a variant of demand constraints (2.2a) which includes a DC representation of the electrical network. When the electrical network and line flow limits are represented in the problem, the number of coupling constraints, (2.2a), becomes too large, specially for large power systems, like the Brazilian one. An approach alternative to space decomposition that overcomes this difficulty consists of duplicating some variables, introducing artificial variables to replace the original ones in some constraints, and then relaxing the newly added equality constraints of artificial and original variables, via LR. Reserve constraints are also relaxed in this approach. We call this scheme *space-time decomposition*, or *variable splitting*, because the equality constraints couple artificial and original variables both along units and time steps, (cf.(2.3.2) below), [1, 10].

Table 2.1 shows the number of dual variables involved when applying either the space decomposition or the space-time decomposition when considering the electrical network, for a system with n units, NB buses (electrical substations), and NL transmission lines.

Table 2.1: Number of dual variables for both decomposition schemes.

Electrical network constraints	**Classical LR approach**	**LR space-time approach**
no	$2\mathcal{T}$	$n\mathcal{T} + \mathcal{T}$
yes	$2(NB + 2NL)\mathcal{T}$	$n\mathcal{T} + \mathcal{T}$

The Brazilian mix has 484 hydro-units, 123 thermal-units, the electrical network has 5046 transmission lines, and 3544 buses. The horizon of time ranges from 24 hours to 168 hours (1 week) [4], pp.170-172. The corresponding number of dual variables for the Brazilian case is reported in Table 2.2.

We see from the table that the space-time decomposition approach yields a much smaller dual problem when the electrical network is represented in the optimization problem, see [5] for more details. We now describe in more details the decomposition schemes when applied to the SCTUC_{H} problem.

Table 2.2: Number of dual variables for time horizon from 24 to 168 hours.

Electrical network constraints	Classical LR approach	LR space-time approach
no	[48, 336]	[14592, 102144]
yes	[654528, 4581696]	[14592, 102144]

2.3.1 Space Decomposition

Consider the following rewriting of problem (2.1)-(2.2):

$$\begin{cases} \min\limits_{u,\tilde{u},p_T,p_H} C_T(p_T) + C_H(p_H) \\ \text{subject to} \\ A_T p_T + A_H p_H = D \\ B_u u + B_T p_T + B_H p_H = R \\ (\mathrm{TC}(u, \tilde{u}, p_T))_{t\leq\mathcal{T}} \\ (\mathrm{HC}(p_H))_{t\leq\mathcal{T}}. \end{cases} \tag{2.3}$$

Note that variables p_T, p_H are coupled by the demand and reserve constraint. One possibility to uncouple them is to relax constraints (2.2), i.e., $A_T p_T + A_H p_H = D$, and $B_u u + B_H p_H + B_T p_T = R$. The Lagrangian of problem (2.3) with respect to the relaxation of such constraints is given by:

$$\mathfrak{L}(y, \lambda) = C_T(p_T)+C_H(p_H)+\lambda_D^\top(A_T p_T+A_H p_H-D)+\lambda_R^\top(B_u u+B_T p_T+B_H p_H-R)$$

where $y = (u, \tilde{u}, p_T, p_H)$ contains the primal variables, and $\lambda = (\lambda_D, \lambda_R)$ the Lagrangian multipliers corresponding to the demand and spinning reserve constraints, respectively. We obtain the following problem, equivalent to (2.3),

$$\begin{cases} \min\limits_{y} \max\limits_{\lambda} \mathfrak{L}(y, \lambda) \\ \text{subject to} \\ (\mathrm{TC}(u, \tilde{u}, q_T))_{t\leq\mathcal{T}} \\ (\mathrm{HC}(q_H))_{t\leq\mathcal{T}}. \end{cases}$$

The corresponding dual problem is:

$$\begin{cases} \max\limits_{\lambda} \min\limits_{y} \mathfrak{L}(y, \lambda) \\ \text{subject to} \\ (\mathrm{TC}(u, \tilde{u}, q_T))_{t\leq\mathcal{T}} \\ (\mathrm{HC}(q_H))_{t\leq\mathcal{T}}. \end{cases}$$

Decomposition is evident if we write

$$\begin{aligned}\mathfrak{L}(y,\lambda) = C_T(p_T) + \lambda_D^\top A_T p_T + \lambda_R^\top (B_u u + B_T p_T) + \\ C_H(p_H) + \lambda_D^\top A_H p_H + \lambda_R^\top B_H p_H - \lambda_D^\top D - \lambda_R^\top R,\end{aligned}$$

then

$$\min_y \mathfrak{L}(y,\lambda) = \theta_{p_T}(\lambda) + \theta_{p_H}(\lambda) - \lambda_D^\top D - \lambda_R^\top R,$$

where we introduced the dual functions

$$\theta_{p_T}(\lambda) = \begin{cases} \min\limits_{u,\tilde{u},p_T} C_T(p_T) + \lambda_D^\top A_T p_T + \lambda_R^\top (B_u u + B_T p_T) \\ \text{subject to} \\ (\mathrm{TC}(u,\tilde{u},q_T))_{t\le\mathcal{T}}, \end{cases} \tag{2.4a}$$

$$\theta_{p_H}(\lambda) = \begin{cases} \min\limits_{p_H} C_H(p_H) + \lambda_D^\top A_H p_H + \lambda_R^\top B_H p_H \\ \text{subject to} \\ (\mathrm{HC}(q_H))_{t\le\mathcal{T}}. \end{cases} \tag{2.4b}$$

For any given multipliers $\lambda = (\lambda_D, \lambda_R)$, the minimization problem $\theta(\lambda) = \min_y \mathfrak{L}(y,\lambda)$ is called Lagrangian subproblem corresponding to (2.3). In particular, subproblems $\theta_{p_T}(\lambda)$, and $\theta_{p_H}(\lambda)$ in (2.4) are called local thermal and hydraulic, respectively.

2.3.2 Space-Time Decomposition

This decomposition scheme follows [1], section 3, and the PhD.thesis [4], chapter 6.

The primal form of problem (2.1)-(2.2) is:

$$\begin{cases} \min\limits_{u,\tilde{u},p_T,p_H} C_T(p_T) + C_H(p_H) \\ \text{subject to} \\ A_T p_T + A_H p_H = D \\ B_u u + B_T p_T + B_H p_H = R \\ (\mathrm{TC}(u,\tilde{u},p_T))_{t\le\mathcal{T}} \\ (\mathrm{HC}(p_H))_{t\le\mathcal{T}}. \end{cases}$$

Variables p_T, p_H are coupled by the demand and spinning reserve constraints. To uncouple these variables, we can introduce artificial variables

q_T (duplicating the variable p_T) and q_H (duplicating the variable p_H), see [1], [4], to obtain the following problem, equivalent to (2.3):

$$\begin{cases} \min\limits_{u,\tilde{u},q_T,q_H,p_T,p_H} C_T(q_T) + C_H(q_H) \\ \text{subject to} \\ A_T p_T + A_H p_H = D \\ B_u u + B_T q_T + B_H q_H = R \\ q_T = p_T \\ q_H = p_H \\ (\mathrm{TC}(u, \tilde{u}, q_T))_{t \le \mathcal{T}} \\ (\mathrm{HC}(q_H))_{t \le \mathcal{T}}. \end{cases} \tag{2.5}$$

The Lagrangian of problem (2.5) with respect to the relaxation of both the spinning reserve and the artificial variable equality constraints is defined by:

$$\begin{aligned} \mathfrak{L}(y, \lambda) = C_T(q_T) + C_H(q_H) + \lambda_R^\top (B_u u + B_T q_T + B_H q_H - R) + \\ \lambda_T^\top (q_T - p_T) + \lambda_H^\top (q_H - p_H) \end{aligned}$$

where, once again, $y = (u, \tilde{u}, q_T, q_H, p_T, p_H)$ contains the primal variables, and $\lambda = (\lambda_R, \lambda_T, \lambda_H)$ is the Lagrangian multiplier corresponding to the spinning reserve, artificial thermal variable constraint, and artificial hydraulic variable constraint, respectively. We obtain the following problem, equivalent to (2.5),

$$\begin{cases} \min\limits_{y} \max\limits_{\lambda} \mathfrak{L}(y, \lambda) \\ \text{subject to} \\ A_T p_T + A_H p_H = D \\ (\mathrm{TC}(u, \tilde{u}, q_T))_{t \le \mathcal{T}} \\ (\mathrm{HC}(q_H))_{t \le \mathcal{T}}. \end{cases}$$

The corresponding dual problem is:

$$\begin{cases} \max\limits_{\lambda} \min\limits_{y} \mathfrak{L}(y, \lambda) \\ \text{subject to} \\ A_T p_T + A_H p_H = D \\ (\mathrm{TC}(u, \tilde{u}, q_T))_{t \le \mathcal{T}} \\ (\mathrm{HC}(q_H))_{t \le \mathcal{T}}. \end{cases}$$

Decomposition is evident if we write

$$\mathfrak{L}(y,\lambda) = C_T(q_T) + \lambda_R^\top(B_u u + B_T q_T) + \lambda_T^\top q_T + C_H(q_H) + \lambda_R^\top B_H q_H + \lambda_H^\top q_H - \lambda_T^\top p_T - \lambda_H^\top p_H - \lambda_R^\top R,$$

then

$$\min_y \mathfrak{L}(y,\lambda) = \theta_{q_T}(\lambda) + \theta_{q_H}(\lambda) + \theta_p(\lambda) - \lambda_R^\top R,$$

where

$$\theta_{q_T}(\lambda) = \begin{cases} \min\limits_{u,\tilde{u},q_T} C_T(q_T) + \lambda_R^\top(B_u u + B_T q_T) + \lambda_T^\top q_T \\ \text{subject to} \\ (\mathrm{TC}(u,\tilde{u},q_T))_{t\le\mathcal{T}}, \end{cases} \tag{2.6a}$$

$$\theta_{q_H}(\lambda) = \begin{cases} \min\limits_{q_H} C_H(q_H) + \lambda_R^\top B_H q_H + \lambda_H^\top q_H \\ \text{subject to} \\ (\mathrm{HC}(q_H))_{t\le\mathcal{T}}, \end{cases} \tag{2.6b}$$

$$\theta_p(\lambda) = \begin{cases} \min\limits_{p_T,p_T} -\lambda_T^\top p_T - \lambda_H^\top p_H \\ \text{subject to} \\ A_T p_T + A_H p_H = D. \end{cases} \tag{2.6c}$$

For any given multiplier $\lambda = (\lambda_R, \lambda_T, \lambda_H)$, the minimization problem $\theta(\lambda) = \min_y \mathfrak{L}(y,\lambda)$ is called Lagrangian subproblem corresponding to (2.5). Subproblems $\theta_{q_T}(\lambda)$, $\theta_{q_H}(\lambda)$, $\theta_p(\lambda)$ in (2.6) are called local thermal, hydraulic, and electric problem, respectively.

We can see that both decomposition schemes lead to a local thermal problem, (2.4a) or (2.6a).

Since for the Brazilian system, network constraints are represented in the short-term model, and the space-time decomposition is preferred in this case, the LTP in (2.6a) is described in more details in the next chapter. However, it is important to note that (2.4a) can be also solved with the same methodology presented in sections 3.3 and 3.4.

Chapter 3

Local Thermal Problem

We now give in details the thermal unit commitment constraints (TC($u, \tilde{u}, q_T$)), and related thermal cost. Section 3.1 gives the mathematical formulation of the LTP problem, and in section 3.2 shows the separable structure by *single unit subproblems* (SUP) of the LTP problem. Section 3.3 and 3.4 are devoted to the solution of the SUP without and with ramping constraints, respectively.

3.1 Mathematical Model of the Local Thermal Problem

The production cost for thermal units includes the fuel cost for generating power and the start-up cost. For the unit i at time step t, let $F(p_i^t)$ be the fuel cost, modeled as a quadratic convex function of p_i^t, and let S_i^t be the start-up cost of i. For convenience, all along this section we denote the production cost by $C_T(u, \tilde{u}, q_T)$ instead of $C_T(q_T)$, in order to make more clear its dependence on the binaries variables u, used to represent if the unit is on-line or not, and $\tilde{u}$, represents if the unit is in start-up/shutdown process.

Similarly to [2], the production cost $C_T(u, \tilde{u}, q_T)$ is defined as:

$$C_T(u, \tilde{u}, q_T) = \sum_{t=1}^{\mathcal{T}} \sum_{i=1}^{I} (u_i^t + \tilde{u}_i^t) F_i(q_i^t) + u_i^t (1 - u_i^{t-1}) S_i^t. \tag{3.1}$$

After relaxation of the SCTUC_H problem, the mathematical expression

for the purely thermal local problem (in the space-time decomposition approach, i.e., as in (2.6a)) is:

$$\min_{u,\tilde{u},q_T} \left\{ C_T(u,\tilde{u},q_T) + \sum_{t=1}^{\mathcal{T}} \sum_{i=1}^{I_T} \lambda_{R_i}^t r^t + \sum_{t=1}^{\mathcal{T}} \sum_{i=1}^{I_T} \lambda_{T_i}^t q_{T_i}^t \right\},$$

subject to the following constraints:

a) The 0-1 control variable u_i^t is defined for $i = 1 \ldots I_T$ and $t = 1, \ldots, \mathcal{T}$ takes the values:

$$u_i^t = \begin{cases} 1 & \text{if unit } i \text{ is on-line at time } t \\ 0 & \text{if unit } i \text{ is off-line at time } t \\ 0 & \text{if unit } i \text{ is in start-up process at time } t \\ 0 & \text{if unit } i \text{ is in shutdown process at time } t. \end{cases} \tag{3.2a}$$

with this convention, the thermal unit is considered to be on-line only after having completed the start-up process. Similarly, the thermal unit is off-line after completing the shutdown process.

The additional 0-1 variable $\tilde{u}_i^t$ determines if the thermal unit i at time step t is in start-up/shutdown process. For $i = 1 \ldots I_T$ and $t = 1, \ldots, \mathcal{T}$, v_i^t is defined by:

$$\tilde{u}_i^t = \begin{cases} 0 & \text{if unit } i \text{ is on-line at time } t \\ 0 & \text{if unit } i \text{ is off-line at time } t \\ 1 & \text{if unit } i \text{ is in start-up process at time } t \\ 1 & \text{if unit } i \text{ is in shutdown process at time } t. \end{cases}$$

b) Unit generation limits

$$u_i^t \underline{p}_i \leq q_i^t \leq (u_i^t + \tilde{u}_i^t)\overline{p}_i, \tag{3.2b}$$

where $\underline{p}_i$ is the minimum power production and $\overline{p}_i$ is the maximum power production of thermal unit i, respectively.

c) Unit minimum start-up/shutdown times.

d) Unit start-up/shutdown curves

e) Unit minimum up/down times.

f) Ramping Constraints.

Up/down, start-up/shutdown, and ramping constraints. We now explain the modeling of constraints (c), (d), (e), and (f). As in [1], section 5, constraint (c) represents the minimum time required to start-up/shutdown the unit. These constraints are [3]

$$u_i^{t-j} \leq 1 + u_i^t - u_i^{t+1} \quad \text{if } j < t_i^{\text{up}} \tag{3.3a}$$

$$u_i^{t+j} \leq 1 - u_i^t + u_i^{t+1} \quad \text{if } j < t_i^{\text{dn}}, \tag{3.3b}$$

where t_i^{up} and t_i^{dn} are the minimum time required to start-up and shutdown unit i, respectively.

Restriction (d) models the production output of the unit during the start-up/shutdown process. We give more details of this restriction in section 3.3.

As in [2], restriction (e) is added to avoid too many consecutive modifications in the state of the unit. More precisely, to discourage frequent start-ups and shutdowns, a unit must stay on-line for at least T_i^{on} hours whenever it is started, i.e., after completing the start-up process. Similarly, the unit must remain off-line for at least T_i^{off} hours after completing the shutdown process. To enforce such condition, we define a variable $x_i^{\text{on/off}}(t)$ to keep track of the amount of time that unit i has been on/off continuously up to, and including, time step t. These stage variables are defined recursively by the relations:

$$x_i^{\text{on}}(t) = u_i^t(\Delta + x_i^{\text{on}}(t-1)) \tag{3.4a}$$

$$x_i^{\text{off}}(t) = (1 - u_i^t)(1 - \tilde{u}_i^t)(\Delta + x_i^{\text{off}}(t-1)), \tag{3.4b}$$

where Δ is the length (in hours) of a time step. Minimum up and down time constraints (c) are then given by the relations:

$$(x_i^{\text{on}}(t-1) - T_i^{\text{on}})(u_i^{t-1} - u_i^t) \geq 0 \quad \text{for } t = 1, \ldots, \mathcal{T} \tag{3.5a}$$

$$\left(x_i^{\text{off}}(t - t^{\text{up}} - 1) - T_i^{\text{off}}\right)(u_i^t - u_i^{t-1}) \geq 0 \quad \text{for } t = 1, \ldots, \mathcal{T}. \tag{3.5b}$$

Finally, ramping constraints (f) limit the generating capacity of the unit for two consecutive time steps once it is on-line. These constraints are given by, [6],[13], section 1,

$$p_i^t - p_i^{t-1} \leq \text{RU}_i \quad \text{as generation increases} \tag{3.6a}$$

$$p_i^{t-1} - p_i^t \leq \text{RD}_i \quad \text{as generation decreases.} \tag{3.6b}$$

Figure 3.1 displays constraints (3.2)-(3.6).

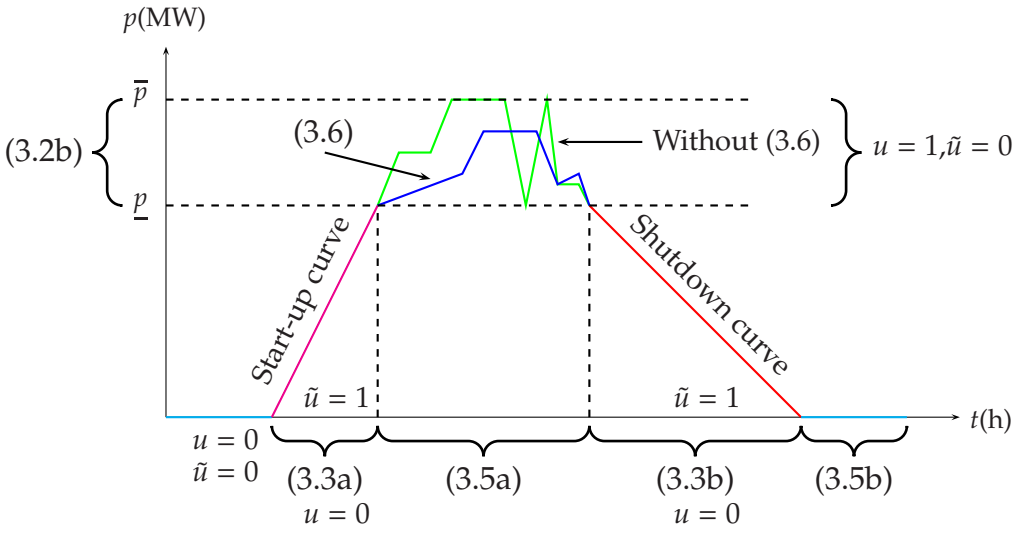

Figure 3.1: Illustration of the Thermal Dynamic Constraints.

Start-up cost. In the objective function (3.1), the start-up cost can be taken fixed [3], section 6.3.1, or time dependent. Similarly to [2], a time varying with exponential growth start-up cost has the form:

$$S_i^t = S_i(x_i^{\text{off}}(t - t^{\text{up}} - 1)) = \alpha_i + \beta_i(1 - e^{-x_i^{\text{off}}(t-t^{\text{up}}-1)/\tau_i}),$$

where the unit specific parameters $\alpha_i, \beta_i, \tau_i$ are non-negative, and estimated separately for each unit. The parameter τ_i represents the *cooling* time of the unit. Since x_i^{off} is non-negative, the term $(1 - e^{-x_i^{\text{off}}(t-t^{\text{up}}-1)/\tau_i})$ lies between 1 and 0. If $x_i^{\text{off}} \ggg 0$ the term $(1 - e^{-x_i^{\text{off}}(t-t^{\text{up}}-1)/\tau_i})$ is close to 1 and the resulting start-up cost is almost $\alpha_i + \beta_i$, this situation is called *cold start-up* [2]. Likewise, a *warm start-up* occurs when the term $(1 - e^{-x_i^{\text{off}}(t-t^{\text{up}}-1)/\tau_i})$ is close to 0, the cost is approximately α_i, which is the fixed therm of the start-up cost, [2].

In some cases the start-up cost can be defined by:

$$S_i^t = S_i(x_i^{\text{off}}(t - t^{\text{up}} - 1)) = \begin{cases} \alpha_i & \text{if } x_i^{\text{off}}(t - t^{\text{up}} - 1) \leq \tau_i \\ \alpha_i + \beta_i & \text{if } x_i^{\text{off}}(t - t^{\text{up}} - 1) > \tau_i, \end{cases}$$

as in the work [7].

We mention that in [7, 4], the backwards variant of DP used to solve the LTP does not allow for accurate representation of the variable start-up

costs. By contrast, our two DP techniques, which are forward, can handle without difficulties variables start-up costs, see 3.3.1 and 3.3.2 below.

3.2 Solution Method

The minimization problem $\theta_{q_T}(\lambda)$ (cf.(2.6a)) has the following separable structure, [1, 2, 3, 4]:

$$\theta_{q_T}(\lambda) = \sum_{i=1}^{I_T} \theta_{q_i}(\lambda),$$

where θ_{q_i} represents the so-called single unit subproblem (SUP), and is given by:

$$\theta_{q_i}(\lambda) = \begin{cases} \min\limits_{u_i, \tilde{u}_i, q_i} \sum\limits_{t=1}^{\mathcal{T}} C_i^t(u_i, \tilde{u}_i, q_i^t) + \lambda_R^t r_i^t + \lambda_{T_i}^t q_i^t. \\ \text{subject to} \\ (\mathrm{TC}(u_i, \tilde{u}_i, q_i))_{t \leq \mathcal{T}}^i. \end{cases} \tag{3.7}$$

where $C_i^t(u_i, \tilde{u}_i, q_i^t)$, is the thermal production cost of unit i at time step t.

If in the commitment constraints $(\mathrm{TC}(u_i, \tilde{u}_i, q_i))_{t \leq \mathcal{T}}^i$, we do not consider ramping constraints (3.6), the minimization in (3.7) with respect to the variable q_i can be carried out explicitly. Minimization with respect to u_i, and $\tilde{u}_i$ subject to (3.2a),(3.5)-(3.3) is done by applying Dynamic Programming (DP) techniques, as explain ed in the next section.

3.3 Solving SUP without Ramping Constraints

This section is based on the reports [2],[3].

To simplify notation, in this section the unit subindex i will be dropped. Similarly, λ will stand for the Lagrangian multiplier, u^t and $\tilde{u}^t$ are scalar 0-1 variables and q^t is the scalar production of the unit at time step t.

We use DP to solve the mixed-integer quadratic programming problem

$$\theta_{q_T}(\lambda) = \begin{cases} \min\limits_{u, \tilde{u}, q} \sum\limits_{t=1}^{\mathcal{T}} u^t F(q^t) + u^t(1 - u^{t-1})S^t + \lambda_R^t u^t(\overline{p} - q^t) + \lambda_T{}^t q^t \\ \text{subject to} \\ (\mathrm{TC}(u, \tilde{u}, q))_{t \leq \mathcal{T}}, \end{cases}$$

where the commitment constraints, $(TC)_t$, are (3.2), (3.5), and (3.3).

Our formulation considers the following issues:

- At each time step the unit can be in 1 out of 2 possible states: *on* or *off*, modeled by variable u^t.
- Auxiliary states are needed to indicate if the unit did not complete the start-up or shutdown processes.
- When the state changes from *off* to *on*, there is a start-up cost for the unit, which is time dependent.
- When the state changes from *on* to *of*, there is a shutdown cost for the unit, which is fixed.
- There are unit minimum start-up/shutdown times constraints.
- There are unit minimum up/down times constraints.
- While the unit is at a start-up/shutdown process, its power generation follows the rules below:
  - If at time t the decision is to start-up the unit, the power generation of the unit will follow a (known) specific curve called "Start-up Curve" until time $t + t^{\text{up}}$, when the power generation will reach the minimum generation level $\underline{p}$.
  - If at time t the decision is to shutdown the unit, the power generation of the unit will follow a (known) specific curve called "Shutdown Curve" until time $t + t^{\text{dn}}$, when generation becomes null, [3].
  - During both, start-up and shutdown processes, the unit does not contribute to the reserve of the system.

  For our study, Start-up and Shutdown curves are given piecewise linear functions. Example of such curves can be found in Figure 3.2 and Figure 3.3.

As in [2], the problem is modeled by a graph whose nodes are indexed by two indices, the first one indicates the unit state *on* or *off*, and the second one indicates the stage variable. Arcs of the graph represent the possible transitions between nodes, see Figure 3.4.

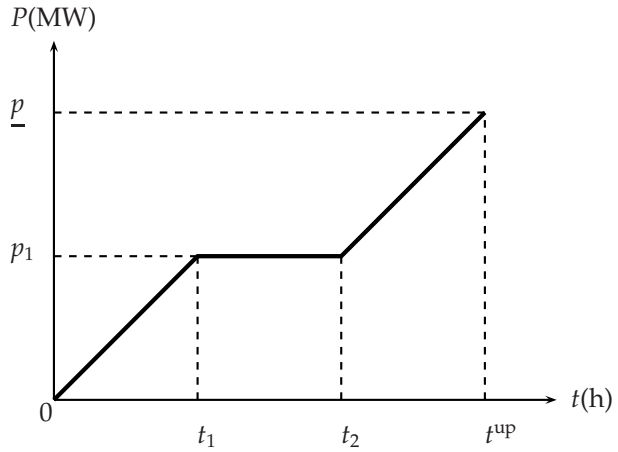

Figure 3.2: Example of a "Start-up Curve".

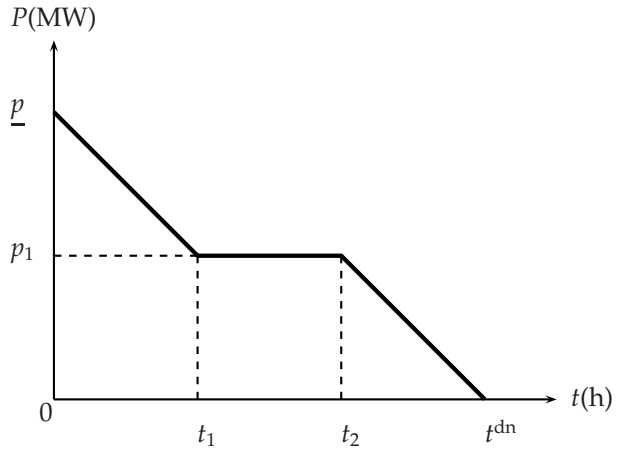

Figure 3.3: Example of a "Shutdown Curve".

Due to minimum up/down times constraint (3.5), we can only make decisions at nodes where the unit has been on-line/off-line for at least $T^{\text{on/off}}$ hours. Since the decision set for a unit that has been on-line/off-line longer than $T^{\text{on/off}}$ hours is the same than for a unit that has been on-line/off-line for exactly $T^{\text{on/off}}$ hours, we only need to consider these nodes in the DP algorithm. This observation reduces dramatically the number of nodes to be explored in the graph. In addition, transitions between nodes must satisfy the minimum start-up/shutdown times constraints, thus reducing

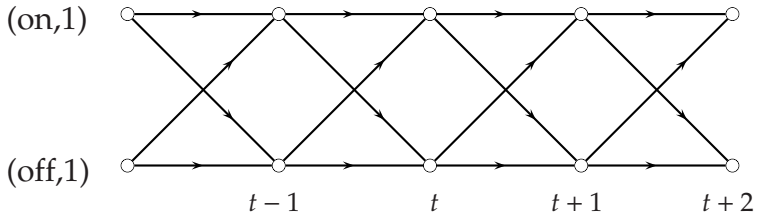

Figure 3.4: DP Formulation for the single unit subproblem with $T^{\text{on}} = 1$ h, $T^{\text{off}} = 1$ h.

the number of arcs of the graph.

Let (on/off, t) be the node at time step t which has been on-line/off-line for at least $T^{\text{on/off}}$. The cost at any node c(on/off, t) represents the minimal cost needed to reach this node in the graph given the initial state at time 0.

The solution of the problem consists in finding the minimal cost path between the initial time step and the final time step, as described in the next section.

3.3.1 Dynamic Programming Algorithm. Alternative 1

The definition of the graph to be explored by DP is, [12], section 4.3:

$$\begin{aligned} \mathcal{N}_{DP} &= \{1,\ldots,\mathcal{T}\} \times \Big(\{(\text{on},\tau)|\tau = 0,1,\ldots,T^{\text{on}}\} \cup \{(\text{off},\tau)|\tau = 0,1,\ldots,T^{\text{off}}\}\Big) \\ \mathcal{A}_{DP} &\subseteq \mathcal{N}_{DP} \times \mathcal{N}_{DP}, \end{aligned}$$

where $\mathcal{N}_{DP}$ is the set of nodes of the graph, $\mathcal{A}_{DP}$ is the set of arcs without loops, $\{1,\ldots,\mathcal{T}\}$ is the time horizon, $T^{\text{on/off}}$ are the minimum up/down times. The sets $\{(\text{on/off},\tau)|\tau = 0,\ldots,T^{\text{on/off}}\}$ represent all the possibles states and stages of the unit (cf.(3.4)). The element (t,(on/off,0)) will represent that at time step t the unit just completed the start-up/shutdown process.

As mentioned, constraints like minimum start-up/shutdown times and minimum up/down times reduce the number of arcs of the graph. For example in Figure 3.5, we may replace the arcs from ((on,1),$t-1$) to ((off,0),t),

and from ((off,0),t) to ((off,1),$t+1$), by a single arc from ((on,1),$t-1$) to ((off,1),$t+1$), [7]. In turn, these feasible transitions reduce the set of arcs $\mathcal{A}_{DP}$.

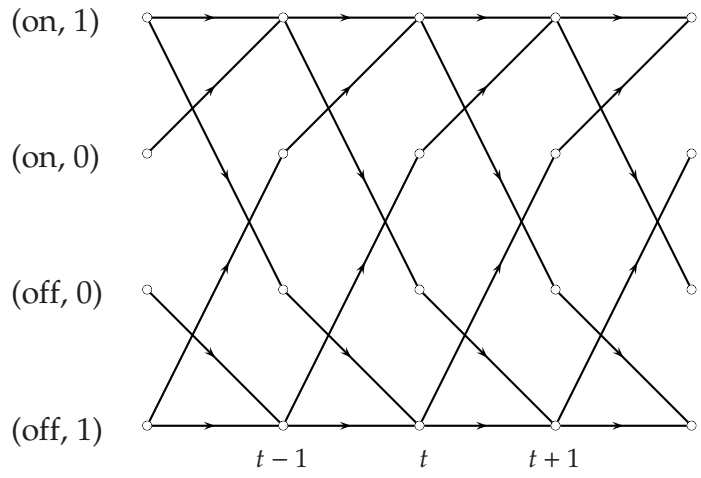

Figure 3.5: DP Formulation for the SUP with $T^{\text{on}} = 1$ h, $T^{\text{off}} = 1$ h, $t^{\text{up}} = 1$ h,$t^{\text{dn}} = 1$ h.

Given $(v, w) = ((v_t, (v_u, v_\tau)), (w_t, (w_u, w_\tau))) \in \mathcal{N}_{DP} \times \mathcal{N}_{DP}$, we have that

$(v, w) \in \mathcal{A}_{DP}$ if it satisfies:

$$
\begin{array}{ll}
\text{if } v_u = w_u & \\
 & \quad \text{if } v_\tau < T^{\text{on/off}} \\
 & \quad\quad w_t = v_t + 1 \\
 & \quad\quad w_\tau = v_\tau + 1 \\
 & \quad \text{if } v_\tau \geq T^{\text{on/off}} \\
 & \quad\quad w_t = v_t + 1 \\
 & \quad\quad w_\tau = v_\tau. \\
\text{if } v_u = w_u + 1 & \\
 & \quad\quad w_t = v_t + t^{\text{dn}} \\
 & \quad\quad w_\tau = 0. \\
\text{if } v_u = w_u - 1 & \\
 & \quad\quad w_t = v_t + t^{\text{up}} \\
 & \quad\quad w_\tau = 0.
\end{array}
$$

Before defining the DP transition cost function, we explain how start-up and shutdown curve costs can be computed, as well as the introduction of auxiliary states used to identify not finalized start-up or shutdown processes.

3.3.2 Start-up and Shutdown curves cost.

Let pu and pd be piecewise continuous linear functions defined as

$$
\begin{array}{rccl}
pu: & [0, t^{\text{up}}] & \rightarrow & [0, \underline{p}] \\
 & t & \mapsto & \sum_{i=1}^{N_u} \chi_{I_i}(t) m_i t + b_i \\
pd: & [0, t^{\text{dn}}] & \rightarrow & [\underline{p}, 0] \\
 & t & \mapsto & \sum_{i=1}^{N_u} \chi_{J_i}(t) \tilde{m}_i t + \tilde{b}_i
\end{array}
$$

where $\cup_i I_i$ is a partition of $[0, t^{\text{up}}]$, $\cup_i J_i$ is a partition of $[0, t^{\text{dn}}]$, $\chi_A(t)$ is the indicator function and $m_i t + b_i$ is the linear segment defined on I_i. Likewise, $\tilde{m}_i t + \tilde{b}_i$ is the linear segment defined on J_i. We consider that pu and pd have the following properties: pu is a non-decreasing function such that $pu(0) = 0$ and $pu(t^{\text{up}}) = \underline{p}$, pd is a non-increasing function such that $pd(0) = \underline{p}$ and $pd(t^{\text{dn}}) = 0$.

Let *ctup* be the *constant start-up curve cost* and *ctdn* be the *constant shut-down curve cost*, defined by:

$$ctup = \int_0^{t^{\text{up}}} F(pu(t))\mathrm{d}t, \tag{3.8a}$$

$$ctdn = \int_0^{t^{\text{dn}}} F(pd(t))\mathrm{d}t, \tag{3.8b}$$

where F is the fuel cost of the unit, modeled as a quadratic convex function.

We define the functions Su, the start-up curve cost, and Sd, the shut-down curve cost, as:

$$Su(t) = ctup + \sum_{i=1}^{t^{\text{up}}} \lambda_T(t+i)pu(i), \quad t + t^{\text{up}} \leq \mathcal{T}, \tag{3.9}$$

$$Sd(t) = ctdn + \sum_{i=1}^{t^{\text{dn}}} \lambda_T(t+i)pd(i), \quad t + t^{\text{dn}} \leq \mathcal{T}, \tag{3.10}$$

where $t \in \{1, \ldots, \mathcal{T}\}$ and λ_T is the Lagrangian multiplier corresponding to thermal artificial variable constraints.

Remark 1. The costs Su, Sd could also be calculated as below:

$$Su(t) = \sum_{i=1}^{t^{\text{up}}} F(pu(i))\lambda_T(t+i)pu(i), \tag{3.11a}$$

$$Sd(t) = \sum_{i=1}^{t^{\text{dn}}} F(pd(i)) + \lambda_T(t+i)pd(i), \tag{3.11b}$$

where $t \in \{1, \ldots, \mathcal{T}\}$ and λ_T is the Lagrangian multiplier corresponding to the artificial variable constraints, equations (3.11) are easy to be implemented but they are not a good estimation of the curves cost, because equations (3.11) depend on the length of a time step and they do not consider the whole curve process cost.

3.3.3 Auxiliary States

In our study, we defined the following auxiliary states at time step $\mathcal{T}$ or $t = 0$:

- State *up* indicates that the unit did not complete the start-up process. The stage at this state, τ_{up}, is defined as the number of hours spent in the start-up process. For example, $\tau_{\text{up}} = t^{\text{up}} - (\mathcal{T} - t)$ at time step $\mathcal{T}$.

- State *dn* indicates that the unit did not complete the shutdown process. Similarly, the stage at this state, τ_{dn}, is given by the number of hours spent in the shutdown process, i.e., $\tau_{\text{dn}} = t^{\text{dn}} - (\mathcal{T} - t)$ at time step $\mathcal{T}$.

These auxiliary states do not affect the DP procedure. They are used if we want to allow changes, at time step t, in the state of the unit if $t + t^{\text{up}} > \mathcal{T}$, or $t + t^{\text{dn}} > \mathcal{T}$. With this states, the curves cost take the form:

$$Su(t) = \int_0^{\tau_{\text{up}}} F(pu(\tau))\mathrm{d}\tau + \sum_{i=1}^{\tau_{\text{up}}} \lambda_T(t+i)pu(i), \quad t + t^{\text{up}} > \mathcal{T}, \tag{3.12}$$

$$Sd(t) = \int_0^{\tau_{\text{dn}}} F(pd(\tau))\mathrm{d}\tau + \sum_{i=1}^{\tau_{\text{dn}}} \lambda_T(t+i)pd(i), \quad t + t^{\text{dn}} > \mathcal{T}. \tag{3.13}$$

3.3.4 Transition Cost Function.

As in [4], the DP transition cost function (Ct), considering unit minimum up/down times and minimum unit start-up/shutdown times, between the nodes is computed for each possible transition:

- Transition cost "on-on": Ct is the power generation cost at time step $t + 1$:

$$Ct[(\text{on}, t) \to (\text{on}, t+1)] = F(q^{t+1}) + \lambda_R^{t+1}(\overline{p} - q^{t+1}) + \lambda_T^{t+1} q^{t+1}.$$

- Transition cost "on-off" is computed as below:

  i) If $t + t^{\text{dn}} \leq \mathcal{T}$, we first explore the graph until the node $(\text{off}, t + t^{\text{dn}})$,i.e, when the unit completes the shutdown process. The transition cost is:

$$Ct[(\text{on}, t) \to (\text{off}, t + t^{\text{dn}})] = H + Sd(t),$$

  where H is the fixed shutdown cost, $Sd(t)$ is the shutdown curve cost, defined in (3.10), which is equal to the cost of generation

power between the time steps $t+1$ and $t+t^{\text{dn}}$. Then we advance until the next *must-off node*, $(\text{off}, t+t^{\text{dn}}+r)$, where $r = \min\{\mathcal{T} - t - t^{\text{dn}}, T^{\text{off}}\}$. The *must-off* transition cost C_{off}, which is null, is the transition cost between the time steps $t+t^{\text{dn}}+1$ and $t+t^{\text{dn}}+r$. The total transition cost to reach the *must-off node* is given by:

$$Ct[(\text{on}, t) \rightarrow (\text{off}, t+t^{\text{dn}}+r)] = Ct[(\text{on}, t) \rightarrow (\text{off}, t+t^{\text{dn}})] + C_{\text{off}}.$$

ii) If $t+t^{\text{off}} > \mathcal{T}$, we consider two possibilities:

- Changes in the state of the unit are allowed: work with the auxiliary state *dn*. In this case, the transition cost is:

$$Ct[(\text{on}, t) \rightarrow (\text{dn}, \mathcal{T})] = H + Sd(t),$$

where the node $(\text{dn}, \mathcal{T})$ is an artificial node which indicates that unit did not complete the shutdown process, and $Sd(t)$ is given by (3.12).

- Changes in the state of the unit are not allowed. In this case, the transition cost is:

$$Ct[(\text{on}, t) \rightarrow (\text{off}, \mathcal{T})] = \infty.$$

- Transition cost "off-on", similarly as in the previous item, is computed as follows:

i) If $t+t^{\text{up}} \leq \mathcal{T}$, we first explore the graph until the node $(\text{on}, t+t^{\text{up}})$, i.e, when the unit completes the start-up process. The transition cost is:

$$Ct[(\text{off}, t) \rightarrow (\text{on}, t+t^{\text{up}})] = S^t + Su(t),$$

where S^t is the start-up unit cost, $Su(t)$ is the start-up curve cost, defined in (3.9), which is equal to the cost of generation power between the time steps $t+1$ and $t+t^{\text{up}}$. Then we advance until the next *must-on node*, $(\text{on}, t+t^{\text{up}}+r)$, where $r = \min\{\mathcal{T} - t - t^{\text{up}}, T^{\text{on}}\}$. The *must-on* transition cost is the generation cost between the time steps $t+t^{\text{up}}+1$ and $t+t^{\text{up}}+r$. C_{on} is given by

$$C_{\text{on}} = \sum_{i=1}^{r} \Theta_{t+t^{\text{up}}+i},$$

where for $i = 1, \ldots, r$, $\Theta_{t+t^{\text{up}}+i}$ is solution of:

$$\Theta_{t+t^{\text{up}}+i} = \begin{cases} \min\limits_{q^{t+t^{\text{up}}+i}} F(q^{t+t^{\text{up}}+i}) + \lambda_R^{t+t^{\text{up}}+i}(\overline{p} - q^{t+t^{\text{up}}+i}) + \lambda_T^{t+t^{\text{up}}+i} q^{t+t^{\text{up}}+i} \\ \text{subject to} \\ \underline{p} \leq q^{t+t^{\text{up}}+i} \leq \overline{p}. \end{cases}$$

The total transition cost to reach the *must-on node* is given by:

$$Ct[(\text{off}, t) \rightarrow (\text{on}, t + t^{\text{up}} + r)] = Ct[(\text{off}, t) \rightarrow (\text{on}, t + t^{\text{up}})] + C_{\text{on}}.$$

ii) If $t + t^{\text{up}} > \mathcal{T}$, we consider two possibilities:

- Changes in the state of the unit are allowed: work with the auxiliary state *up*. In this case, the transition cost is:

$$Ct[(\text{off}, t) \rightarrow (\text{up}, \mathcal{T})] = S^t + Su(t),$$

  where the node $(\text{up}, \mathcal{T})$ is an artificial node which indicates that the unit did not complete the start-up process, and $Su(t)$ is given by (3.12).
- Changes in the state of the unit are not allowed. In this case, the transition cost is:

$$Ct[(\text{off}, t) \rightarrow (\text{on}, \mathcal{T})] = \infty.$$

- Finally, the transition cost "off-off" is null:

$$Ct[(\text{off}, t) \rightarrow (\text{off}, t + 1)] = 0.$$

3.3.5 DP algorithm absence of ramping constraints

For a given Lagrangian multiplier, λ, the following algorithm is used to find an optimal solution for the SUP, as in [2], section 8.

Alternative 1 of Dynamic Programming (no ramping constraints) Without loss of generality, we assume that the initial stage of the unit satisfies minimum up/down times (cf. Remark 2 below).

$\mathfrak{S}.1$ Initialization: For $t = 1, \ldots, \mathcal{T}$, let q_*^t be an optimal solution to

$$\Theta_t = \begin{cases} \min\limits_{q^t} F(q^t) + \lambda_R^t(\overline{p} - q^t) + \lambda_T^t q^t \\ \text{subject to} \\ \underline{p} \le q^t \le \overline{p}. \end{cases} \tag{3.14}$$

$\mathfrak{S}.2$ Let $c(\text{off}, t) = \infty$ for all $t = 0, \ldots, \mathcal{T} + t^{\text{dn}} + T^{\text{off}} - 1$, and $c(\text{on}, t) = \infty$ for all $t = 0, \ldots, \mathcal{T} + t^{\text{up}} + T^{\text{on}} - 1$. If the initial state is off, then let $c(\text{off}, 0) = 0$; otherwise, let $c(\text{on}, 0) = 0$.

$\mathfrak{S}.3$ General Steps (Forward Dynamic Programming)

$\mathfrak{S}.3.1$ Compute, while $t < \mathcal{T}$

i Possible transition "off-off"

$$c(\text{off}, t+1) := \min\{c(\text{off}, t+1), c(\text{off}, t)\}$$

n.i If $c(\text{off}, t+1) = c(\text{off}, t)$ then label (off, t) as *previous node* of (off, $t+1$).

ii Possible transition "on-on"

$$c(\text{on}, t+1) := \min\{c(\text{on}, t+1), c(\text{on}, t) + \Theta_{t+1}\}$$

n.ii If $c(\text{on}, t+1) = c(\text{on}, t) + \Theta_{t+1}$ then label (on, t) as *previous node* of (on, $t+1$).

iii Possible transition "off-on"

$$\begin{aligned} c(\text{on}, t + t^{\text{up}} + T^{\text{on}}) \;:=\; & \min\{c(\text{on}, t + t^{\text{up}} \\ & + T^{\text{on}}), c(\text{off}, t) + Ct[\text{off} \to \text{on}]\} \end{aligned}$$

n.iii If $c(\text{on}, t + t^{\text{up}} + T^{\text{on}}) = c(\text{off}, t) + Ct[\text{off} \to \text{on}]$ then label (off,t) as *previous node* of (on, $t + t^{\text{up}} + T^{\text{on}}$).

iv Possible transition "on-off"

$$\begin{aligned} c(\text{off}, t + t^{\text{dn}} + T^{\text{off}}) \;:=\; & \min\{c(\text{off}, t + t^{\text{dn}} \\ & + T^{\text{off}}), c(\text{on}, t) + Ct[\text{on} \to \text{off}]\} \end{aligned}$$

n.iv If $c(\text{off}, t + t^{\text{dn}} + T^{\text{off}}) = c(\text{on}, t) + Ct[\text{on} \to \text{off}]$ then label the node (on, t) as *previous node* of (off, $t + t^{\text{dn}} + T^{\text{off}}$).

$\mathfrak{S}$.3.2 Set $t = t + 1$

$\mathfrak{S}$.4 Check nodes (off, $\mathcal{T}$) , and (on, $\mathcal{T}$), to find the minimum cost. Label the node where the minimum is reached as *last*.

$\mathfrak{S}$.5 Build the optimal path by following all the previous nodes of *last*. □

Remark 2. If the initial stage of the unit, τ^0, does not satisfy minimum up/down times, we keep the initial state until the node, (on/off, $T^{\text{on/off}} - \tau^0$), where constraint (3.5) is satisfied, and compute the corresponding transition cost, defined on subsection 3.3.4, between the nodes $(\text{on}, t) \rightarrow (\text{on}, t)$ or $(\text{off}, t) \rightarrow (\text{off}, t)$, with $t = 0, \ldots, T^{\text{on/off}} - \tau^0$. In this case, the DP algorithm (alternative 1) will begin at time $T^{\text{on/off}} - \tau^0$.

Remark 3. If the initial state of the unit is up/dn, i.e., the unit did not complete the start-up/shutdown process, we advance until the node, (on/off, $t^{\text{up/dn}} - \tau^{\text{up/dn}}$), where the unit completes the process. The transition cost to reach this node is:

- If the initial state is up, there is no decision, the unit must finish the start-up process. The cost is given by:

$$Ct[(\text{up}, \tau^{\text{up}}) \rightarrow (\text{on}, t^{\text{up}} - \tau^{\text{up}})] = ctup - \int_0^{\tau^{\text{up}}} F(pu(\tau))\mathrm{d}\tau + \sum_{i=1}^{t^{\text{up}}-\tau^{\text{up}}} \lambda_T(i) pu(\tau^{\text{up}} + i),$$

- Similarly, if the initial state is dn, the unit must finish the shutdown process. The cost is:

$$Ct[(\text{dn}, \tau^{\text{dn}}) \rightarrow (\text{off}, t^{\text{dn}} - \tau^{\text{dn}})] = ctdn - \int_0^{\tau^{\text{dn}}} F(pd(\tau))\mathrm{d}\tau + \sum_{i=1}^{t^{\text{dn}}-\tau^{\text{dn}}} \lambda_T(i) pd(\tau^{\text{pd}} + i).$$

Finally, we follow remark 2. The DP algorithm will begin at time $T^{\text{on/off}} + t^{\text{up/dn}} - \tau^{\text{up/dn}}$.

This graph explored by algorithm DP1 is frequently used to model LTP without ramping constraints, In works [1, 4, 7] a backwards DP variant is employed, instead of the forward step ☪.3. The advantage of the forward variant is that the stage of the unit can be estimated at each time step as follows:

1. If the condition at step n.i is true, then

$$x^{\text{off}}(t+1) = x^{\text{off}}(t) + \Delta.$$

2. Similarly, if the condition at step n.ii is true, then

$$x^{\text{on}}(t+1) = x^{\text{on}}(t) + \Delta.$$

3. If the condition at step n.iii is true, if $t + t^{\text{up}} \leq \mathcal{T}$, then:

$$x^{\text{on}}(t + t^{\text{up}} + r) = r,$$

where $r = \min\{\mathcal{T} - t - t^{\text{up}}, T^{\text{on}}\}$. If $t + t^{\text{up}} > \mathcal{T}$, and changes of the unit are allowed, then:

$$x^{\text{up}}(\mathcal{T}) = \tau_{\text{up}} = \mathcal{T} - t,$$

where $x^{\text{up}}(\mathcal{T})$ denote the stage of the artificial node $(\text{up}, \mathcal{T})$.

4. Similarly, if the condition at step n.iv is true, if $t + t^{\text{dn}} \leq \mathcal{T}$, then:

$$x^{\text{off}}(t + t^{\text{dn}} + r) = r,$$

where $r = \min\{\mathcal{T} - t - t^{\text{dn}}, T^{\text{off}}\}$. If $t + t^{\text{dn}} > \mathcal{T}$, and changes of the unit are allowed, then:

$$x^{\text{dn}}(\mathcal{T}) = \tau_{\text{dn}} = \mathcal{T} - t,$$

where $x^{\text{dn}}(\mathcal{T})$ denote the stage of the artificial node $(\text{dn}, \mathcal{T})$.

Thus, unlike backwards approaches, time dependent start-up costs can easily be taken into account in our forward DP1 algorithm.

We now consider how to deal with ramping constraints (3.6).

3.4 SUP with Ramping Constraints

This section is based on the paper [8], sections 1,2, and paper [6], sections 1,2,3.

As mentioned, ramping constraints limit the maximum increase or decrease of generated power from one time step to the next one, and reflect the thermal and mechanical inertia that has to be overcome for the unit to increase or decrease its output.

As shown in the previous section, solving the SUP without ramping constraints was modeled as an optimization problem over a straightforward graph (Figure 3.5), essentially proceeding in two main steps:

1. Solving the minimization problem (3.14), Θ_t for $t = 1, \ldots, \mathcal{T}$ at step $\Im.1$ of the algorithm,i.e, finding the optimal generation of one unit at each time step, independently of the whole mix.

2. The combinatorial problem in the binary variables u^t, and v^t is solved by DP, employing the costs found at first step, when solving (3.14).

It is important to remark that the value Θ_t is the cost of commitment of the unit, because Θ_t is the contribution of the variable q^t if the unit is committed at time step t. Otherwise, the contribution is zero. Thus, Θ_t is the cost to be considered along with the curves' costs, fixed shutdown costs, and start-up costs in the DP steps, effectively eliminating the q^t variables from the problem.

Unfortunately, this procedure cannot be applied if ramping constraints are present in the problem. The reason is that variables representing the power output at each time step (q^t) are no longer independent from previous generations, once the commitment decision has been made. In fact, variables q^t and q^{t-1} are linked by the ramping constraints (3.6). Hence, it is no longer possible to determine an optimal output, independently from q^{t-1}, if the unit is on-line. For this reason, it is not possible to apply directly DP alternative 1 presented in subsection 3.3.2 to handle ramping constraints.

Actually, the consideration of ramping constraints requires an entire new graph to efficiently model the different states of the unit. We now describe the new modeling. We mention in passing, even in the absence of ramping constraints, the DP variant algorithm yields a much faster algorithm than DP1.

3.4.1 Solving SUP with Ramping Constraints

To solve the SUP problem with constraints (3.6), we use a different DP procedure. In the DP1 algorithm (see Figure 3.5), nodes in the graph represented states on-off of the unit. Now nodes in the new graph will represent those time steps when the unit switches from state *on* to a state *off*, or viceversa. See Figure 3.6

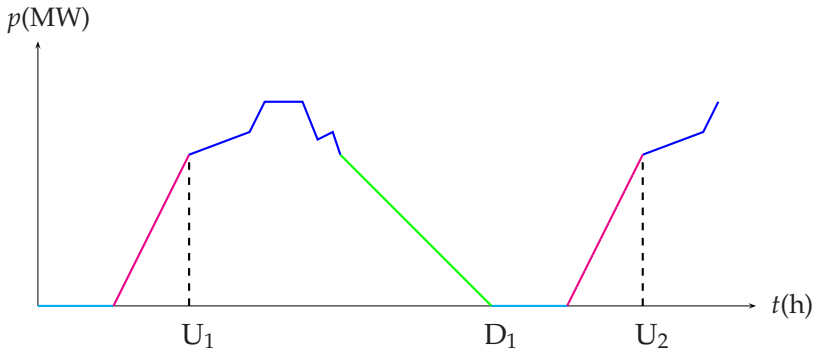

Figure 3.6: Possibles times when unit changes its state.

The state space of this DP alternative comprises, in principle, the set $\mathfrak{U} = \{1, \ldots, f_1\} \cup \mathfrak{D} = \{1, \ldots, f_2\}$, where f_1 and f_2 are two auxiliary sinks corresponding to $\mathcal{T} + t^{\text{up}}$, and $\mathcal{T} + t^{\text{dn}}$, respectively, plus one source denoted s, and one sink denoted f, see Figure 3.7.

In this new graph, the source represents the initial state and stage of the unit at time 0, and the elements of the sets $\mathfrak{U}$ and $\mathfrak{D}$ correspond to possible time steps when the unit completes the start-up and shutdown processes, respectively, i.e., those time steps when the stage of the unit becomes zero. The meaning of each node $t_1 \in \mathfrak{U}$ ($t_2 \in \mathfrak{D}$) is: the state of the unit was off-line (on-line) at time step $t_1 - t^{\text{up}}$ ($t_2 - t^{\text{dn}}$), and it is on-line (off-line) at time step t_1 (t_2). The minimum start-up/shutdown times constraint (3.3) reduce the number of these new nodes, because all nodes such that $t_1 - t^{\text{up}} < 0$ ($t_2 - t^{\text{dn}} < 0$) must correspond to infeasible operations and, hence, may not be considered.

The arcs of the new graph are represented by the pairs (t_1, t_2), (t_2, t_1), for $t_1 \in \mathfrak{U}$ and $t_2 \in \mathfrak{D}$, plus (s, t), and (t, f), for $t \in (\mathfrak{U} \cup \mathfrak{D})$. The meaning of the feasible arcs is explained below:

1. There is an arc, (t_1, t_2), between nodes $t_1 \in \mathfrak{U}$ and $t_2 \in \mathfrak{D}$ if $t_2 - t_1 \geq T^{\text{on}} + t^{\text{dn}}$, i.e., it is feasible to complete the shutdown process at time

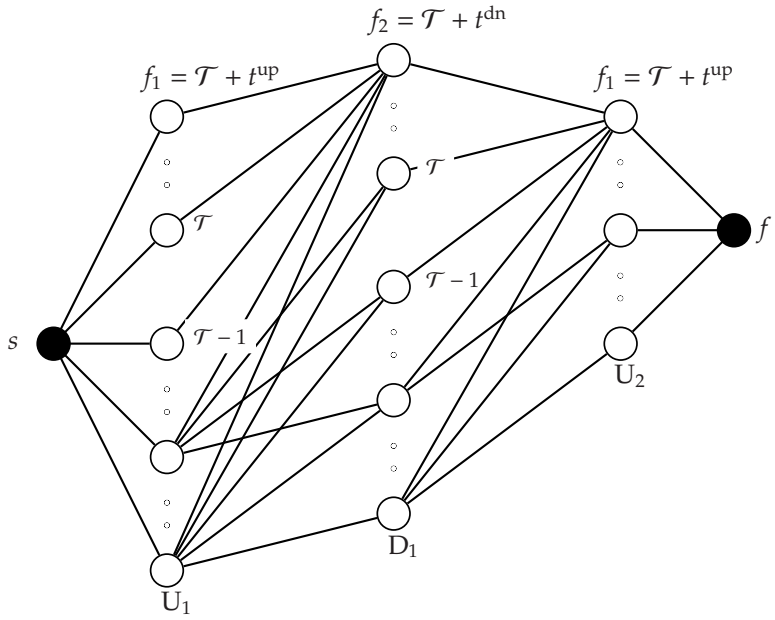

Figure 3.7: Example of a Transition Diagram for Alternative 2.

step t_2 given that the unit was on at time step t_1. The arc (t_1, f_2) means that the unit is on from time step t_1 to $\mathcal{T}$.

2. Similarly, there is an arc, (t_2, t_1), between nodes $t_2 \in \mathfrak{D}$ and $t_1 \in \mathfrak{U}$ if $t_1 - t_2 \geq T^{\text{off}} + t^{\text{up}}$, i.e., it is feasible to complete the start-up process at time step t_1 given that the unit was off-line at time step t_2. As in item 1, the arc (t_2, f_1) means that the unit is off-line from time step t_2 to $\mathcal{T}$.

3. If the initial state of the unit is on, there are arcs from s to all nodes $t \in \mathfrak{D}$, for t such that $t \geq t^{\text{dn}} + k$, with $k = \max\{0, T^{\text{on}} - x^{\text{on}}(0)\}$. The arc (s, f_2) means that the unit is on during the whole time horizon.

4. Similarly, if the initial state of the unit is off-line, there are arcs from s to all nodes $t \in \mathfrak{U}$, for t such that $t \geq t^{\text{up}} + k$, with $k = \max\{0, T^{\text{off}} - x^{\text{off}}(0)\}$. The arc (s, f_1) means that the unit is off-line during the whole time horizon.

5. Finally, there are arcs (t, f), from node $t \in (\mathfrak{U} \cup \mathfrak{D})$, for t such that $t \geq \mathcal{T}$, to the sink. These arcs represent the end of the operation.

Clearly, minimum up/down (3.5) and start-up/shutdown (3.3) constraints reduce the number of these new arcs and the number of nodes.

3.4.2 Dynamic Programming Algorithm. Alternative 2

As in the DP Alternative 1, the sets $\mathcal{N}_{DP}$ and $\mathcal{A}_{DP}$ denote the set of nodes and arcs of the graph, respectively. Let $\tilde{\mathfrak{U}}$ and $\tilde{\mathfrak{D}}$, $\mathcal{A}$, and $\tilde{\mathcal{A}}$ be the sets defined by:

$$\begin{aligned} \mathfrak{T} &= \{1, 2, \ldots, \mathcal{T}\}, \\ \tilde{\mathfrak{U}} &= \{t \in \mathfrak{U} | t \geq t^{\text{up}}\}, \\ \tilde{\mathfrak{D}} &= \{t \in \mathfrak{D} | t \geq t^{\text{dn}}\}, \end{aligned}$$

where $\mathfrak{T}$ represents the number of stages of the graph. We mention that the total number of stages of the graph is lower than $\mathcal{T}$, as we explain in subsection 3.4.4.

New DP Graph. For convenience, we denote the initial state of the unit by τ^0, and the index corresponding to the stage of the graph will be omitted. The definition of the new graph to be explored by DP is:

$$\mathcal{N}_{DP} \subset \mathcal{N}_{sf} \cup \mathcal{N},$$

where the sets $\mathcal{N}_{sf} = \{s, f\}$, and $\mathcal{N} = (\tilde{\mathfrak{U}} \times \mathfrak{T}) \cup (\tilde{\mathfrak{D}} \times \mathfrak{T})$.

Similarly, the arcs are in the set

$$\mathcal{A}_{DP} \subset \mathcal{A}_s \cup \mathcal{A}_{\text{up}} \cup \mathcal{A}_{\text{dn}} \cup \mathcal{A}_f \subset \mathcal{N}_{DP} \times \mathcal{N}_{DP},$$

where the subsets $\mathcal{A}_s, \mathcal{A}_{\text{up}}, \mathcal{A}_{\text{dn}}$, and $\mathcal{A}_f$ are defined below:

$$\begin{aligned}
\mathcal{A}_{\text{s}} &= \{(s,t)|t - k_{\text{on}} \geq t^{\text{up}}\} \cup \{(s,t)|t - k_{\text{off}} \geq t^{\text{dn}}\}, \\
\mathcal{A}_{\text{up}} &= \{(t,r)|r - t \geq T^{\text{off}} + t^{\text{up}}\}, \\
\mathcal{A}_{\text{dn}} &= \{(t,r)|r - t \geq T^{\text{on}} + t^{\text{dn}}\}, \\
\mathcal{A}_f &= \{(t,f)|t \geq \mathcal{T}\},
\end{aligned}$$

where $k_{\text{on}} = \max\{0, T^{\text{on}} - \tau^0\}$, $k_{\text{off}} = \max\{0, T^{\text{off}} - \tau^0\}$.

3.4.3 DP Transition Cost function

For convenience, in this section we denote by τ^-, the stage off (3.4b) of the unit. In addition, $\Theta_{[h,k]}$ denotes the solution of the following optimization problem:

$$\Theta_{[h,k]} = \theta_{[h,k]}(\lambda) = \begin{cases} \min\limits_{q} \sum\limits_{t=h+1}^{k} F(q^t) + \lambda_R^t(\overline{p} - q^t) + \lambda_T{}^t q^t, \\ \text{subject to} \\ \underline{p} \leq q \leq \overline{p}, \quad h+1 \leq t \leq k \\ \overline{q^t} - q^{t-1} \leq \text{RU}, \quad h+1 \leq t \leq k \\ q^{t-1} - q^t \leq \text{RD}, \quad h+1 \leq t \leq k. \end{cases} \tag{3.15}$$

The DP transition cost function, Ct, considering minimum up/down times (3.5), start-up/shutdown times (3.3), and ramping constraints (3.6), is computed for each possible transition:

1. Transition cost "source-node": Given $(s,t) \in \mathcal{A}_s$, the transition cost $Ct[(s,t)]$ is given by:

- If source represents an "off" state:
  i. For $t \in \tilde{\mathfrak{U}}$ such that $t \leq \mathcal{T}$. The transition cost, $Ct[(s,t)]$, is:
  $$Ct[(s,t)] = S^{\tau^-} + Su(t - t^{\text{up}}),$$
  where $\tau^- = \tau^0 + t - t^{\text{up}}$, S^{τ^-} is the time dependent start-up cost, and Su is the start-up curve cost given by (3.9).
  ii. For $t \in \tilde{\mathfrak{U}}$ such that $\mathcal{T} < t < f_1$. The transition cost, $Ct[(s,t)]$, is:
  $$Ct[(s,t)] = S^{\tau^-} + Su(t - t^{\text{up}}),$$
  where $\tau^- = \tau^0 + t - t^{\text{up}}$, S^{τ^-} is the time dependent start-up cost, and Su is the start-up curve cost given by (3.12). These transitions represent that the state of the unit at time $\mathcal{T}$ is *up*, i.e., the unit did not complete the start-up process.
  iii. For $t = f_1$, $Ct[(s,t)]$ is null, because the unit is off-line the whole time horizon.
  $$Ct[(s,f_1)] = 0.$$
- If source represents an "on" state:
  a. For $t \in \tilde{\mathfrak{D}}$ such that $t \leq \mathcal{T}$. The transition cost, $Ct[(s,t)]$, is:
  $$Ct[(s,t)] = H + Sd(t - t^{\text{dn}}) + \Theta_{[0,t-t^{\text{dn}}]},$$
  where H is the fixed shutdown cost, Sd is the shutdown curve cost given by (3.10), and $\Theta_{[0,t-t^{dn}]}$ solves (3.15) with $[h,k] = [0, t - t^{dn}]$.
  b. For $t \in \tilde{\mathfrak{D}}$ such that $\mathcal{T} < t < f_1$. The transition cost, $Ct[(s,t)]$, is:
  $$Ct[(s,t)] = H + Sd(t - t^{\text{dn}}) + \Theta_{[0,t-t^{\text{dn}}]},$$
  where H is the fixed shutdown cost, Sd is the shutdown curve cost given by (3.13), and $\Theta_{[0,t-t^{dn}]}$ is solution of (3.15). These transitions represent that the state of the unit at time $\mathcal{T}$ will be *dn*, i.e., the unit did not complete the shutdown process.
  c. For $t = f_2$, the unit is on the whole time horizon. $Ct[(s,t)]$ is:
  $$Ct[(s,f_2)] = \Theta_{[0,\mathcal{T}]},$$
  where $\Theta_{[0,\mathcal{T}]}$ solves (3.15).

2. Transitions "off-on": Given $(t, r) \in \mathcal{A}_{\text{up}}$, the transition cost, $Ct[(t, r)]$, is:

   - If $t \leq \mathcal{T}$:
     i. If $r \leq \mathcal{T}$, the transition cost, $Ct[(t, r)]$ is:

     $$Ct[(t, r)] = S^{\tau^-} + Su(r - t^{\text{up}}),$$

     where $\tau^- = r - t^{\text{up}} - t$, S^{τ^-} is the time dependent start-up cost, and Su is the start-up cost curve given by (3.9).
     ii. If $\mathcal{T} < r < f_1$, the transition cost is:

     $$Ct[(t, r)] = S^{\tau^-} + Su(r - t^{\text{up}}),$$

     where $\tau^- = r - t^{\text{up}} - t$, S^{τ^-} is the time dependent start-up cost, and Su is the start-up cost curve given by (3.12). As in item 1.ii, these transitions represent that unit did not complete the start-up process.
   - If $\mathcal{T} < t \leq f_2$, r must be equal to f_1. These transition represent the end of operation and the cost is null.

     $$Ct[(t, f_1)] = 0.$$

3. Transition "on-off": Given $(t, r) \in \mathcal{A}_{\text{dn}}$, the transition cost, $Ct[(t, r)]$, is:

   - If $t \leq \mathcal{T}$:
     i. If $r \leq \mathcal{T}$, the transition cost, $Ct[(t, r)]$ is:

     $$Ct[(t, r)] = H + Sd(r - t^{\text{dn}}) + \Theta_{[t,k]},$$

     where H is the fixed shutdown cost, Sd is the shutdown cost curve given by (3.10), $k = r - t^{\text{dn}}$, and $\Theta_{[t,k]}$ solves (3.15).
     ii. If $\mathcal{T} < r < f_2$, the transition cost is:

     $$Ct[(t, r)] = H + Sd(r - t^{\text{dn}}) + \Theta_{[t,k]},$$

     where H is the fixed shutdown cost, Sd is the shutdown cost curve given by (3.13), $k = r - t^{\text{dn}}$, and $\Theta_{[t,k]}$ solves (3.15). As in item 2.ii, these transitions represent that unit did not complete the shutdown process.

- If $\mathcal{T} < t \leq f_1$, r must be equal to f_2. These transitions represent the end of operation and the cost is null.

$$Ct[(t, f_2)] = 0.$$

4. Finally, transition "node-sink". Given $(t, f) \in \mathcal{A}_f$, the transition cost is null.

$$Ct[(t, f)] = 0.$$

3.4.4 DP New Graph Algorithm

We say that a unit completes a *cycle*, if it goes through a start-up/shutdown process, satisfies the minimum up/down times, and then goes through a start-up/shutdown process again in order to return to its original state. See Figure 3.8.

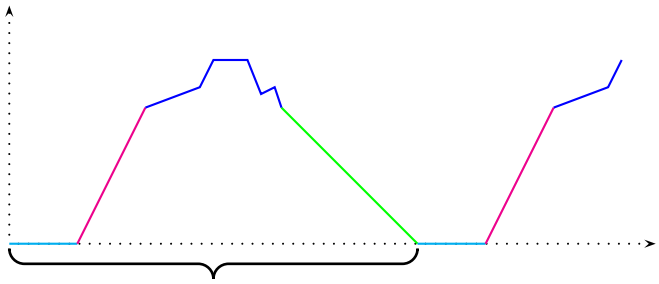

Figure 3.8: Unit Cycles.

We now give more details of how to build the new DP graph (Figure 3.7). The general steps to build the graph are:

G.1 The source and sink are represented by 1.

G.2 Build the vector *UpDn*. This vector has the minimum times required to do consecutive start-up and shutdown processes of the unit during the time horizon. The steps to build this vector are described below:

i. Initialization. If the initial state of the unit is off, do:

$$\begin{aligned} k &= \max\{0, T^{\text{on}} - \tau^0\}, \\ \text{time} &= [t^{\text{up}}, T^{\text{on}}, t^{\text{dn}}, T^{\text{off}}]. \end{aligned}$$

If the initial state of the unit is on, do:

$$\begin{aligned} k &= \max\{0, T^{\text{off}} - \tau^0\}, \\ \text{time} &= [t^{\text{dn}}, T^{\text{off}}, t^{\text{up}}, T^{\text{on}}]. \end{aligned}$$

The vector time has the times required to complete one cycle. Then, recursively define UpDn as follows. Set UpDn(1) = 1, (i.e. the first stage of the graph is represented by the source), and mark the first cycle:

$$\begin{aligned} \text{UpDn}(2) &= k + \text{time}(1), \\ \text{UpDn}(3) &= \text{UpDn}(2) + \text{time}(2) + \text{time}(3), \\ \text{UpDn}(4) &= \text{UpDn}(3) + \text{time}(4) + \text{time}(1), \end{aligned}$$

if UpDn(4) $< \mathcal{T}$, mark the other cycles until an integer i is found for which UpDn(i) $\geq \mathcal{T}$. The dimension of the vector d represents the number of stages of the graph from source to the final cycle found. Then, the last stage of the graph represents the transitions to the sink f, and is marked UpDn(d + 1) = 1.

The auxiliary sinks f_1 and f_2 represent the end of the stages of the graph. If a stage of the graph correspond a time when the unit completes the start-up process, the sink of this stage is f_1. Similarly, if a stage of the graph correspond to a time when the unit completes the shutdown process, the sink of this stage is f_2.

In order to simplify the notation, in the algorithm PD2 below, the end of each stage of the graph is denoted by f_{ud}. We use the wording *stage-up*, if the stage has the time when the unit completes the start-up process. Similarly, *stage-down*, refers to a stage whose time correspond to the unit completing the shutdown process.

As in DP1 algorithm, the cost at any node, $c(t)$, represents the minimal cost needed to reach node t in the graph, given the initial state at time 0. For a given Lagrangian multiplier, λ, the following algorithm is used to find an optimal solution for the SUP with ramping constraints.

Alternative 2 of Dynamic Programming (with ramping constraints)

$\mathfrak{S}.1$ Initialization. Set $c(s) = 0$

$\mathfrak{S}.2$ Costs "source-node": For $t = \text{UpDn}(2), \ldots, f_{\text{ud}}$,

$$c(t) = \{c(s) + Ct[(s,t)]\}.$$

$\mathfrak{S}.3$ Costs "on-off": For $j = 3, \ldots, d$

i. If the stage of the graph j is a stage-up.
For $i = 0, \ldots, f_{\text{ud}} - \text{UpDn}(j)$, do

$$c(t) = \min_{0 \le p \le i} \{c(r) + Ct[(r,t)]\},$$

where $t = \text{UpDn}(j) + i$, and $r = \text{UpDn}(j-1) + p$ with $(r,t) \in \mathcal{A}_{dn}$.

ii. If the stage of the graph j is a stage-down.
For $i = 0, \ldots, f_{\text{ud}} - \text{UpDn}(j)$, do

$$c(t) = \min_{0 \le p \le i} \{c(r) + Ct[(r,t)]\},$$

where $t = \text{UpDn}(j) + i$, and $r = \text{UpDn}(j-1) + p$ with $(r,t) \in \mathcal{A}_{up}$.

iii. Label the node $\tilde{r}$, where the minimum is reached, at steps i and ii, as a "previous node" of t.

$\mathfrak{S}.4$ Minimum cost: The minimum cost is the cost in the sink, given by:

$$c(f) = \min c(t) + Ct[(t,f)],$$

where $t = \text{UpDn}(d) + i$, for $i = 0, \ldots, f_{\text{ud}} - \text{UpDn}(d)$.

$\mathfrak{S}.5$ Label as *last* the node $\tilde{t}$, where the minimum at step $\mathfrak{S}.4$ is reached.

$\mathfrak{S}.6$ Build the optimal path by following all the previous nodes of the node *last*. □

Remark 4. If the initial state of the unit is *up/dn*, i.e., the unit did not complete the start-up/shutdown process, the cost at the source will be:

- If the unit is *up*, it must finish the start-up process. The cost is given by:

$$c(s) \quad = \quad ctup \quad - \quad \int_0^{\tau^{\text{up}}} F(pu(\tau))\mathrm{d}\tau \quad + \quad \sum_{i=1}^{t^{\text{up}}-\tau^{\text{up}}} \lambda_T(i) pu(\tau^{\text{up}} \quad + \quad i),$$

- Similarly, if the unit is *dn*, is must finish the shutdown process. The cost is:

$$c(s) \quad = \quad ctup \quad - \quad \int_0^{\tau^{\text{up}}} F(pu(\tau))\mathrm{dn}\tau \quad + \quad \sum_{i=1}^{t^{\text{dn}}-\tau^{\text{dn}}} \lambda_T(i) pd(\tau^{\text{dn}} \quad + \quad i),$$

The DP, will then begin at time step $t^{\text{up/dn}} - \tau^{\text{up/dn}}$.

3.5 DP Alternatives with extended Time Horizon.

The motivation of this approach is to analyze future impacts of the decisions taken in the time horizon $\{1, \ldots, \mathcal{T}\}$. In both DP alternatives, the following approach was applied:

Given the time horizon $\{1, \ldots, \mathcal{T}\}$, and the Lagrangian multiplier $\lambda = (\lambda_R, \lambda_T)$.

H.1 Set $\mathcal{K} \in \mathbb{Z}$ and given $\mu = (\mu_R, \mu_T)$, where $\mu \in \mathbb{R}^{2\mathcal{K}}$ are estimated a priori, do:

i. Extend the time horizon: $\{1, \ldots, \mathcal{T}, \ldots, \mathcal{T} + \mathcal{K}\}$.

ii. Obtain the *extended Lagrangian multiplier*, $\nu = (\nu_R, \nu_T) \in \mathbb{R}^{2(\mathcal{T}+\mathcal{K})}$, as below:

- For $t = 1, \ldots, \mathcal{T}$:

$$\begin{aligned} \nu_R(t) &= \lambda_R(t) \\ \nu_L(t) &= \lambda_L(t). \end{aligned}$$

- For $k = 1, \ldots, \mathcal{K}$:

$$\begin{aligned} v_R(\mathcal{T}+k) &= \mu_R(k) \\ v_L(\mathcal{T}+k) &= \mu_L(k). \end{aligned}$$

H.2 Apply DP alternative 1 or alternative 2, with the extended time horizon, and the extended Lagrangian multiplier, v. Obtain the optimal scheduling, production levels at each time step, and the minimal cost.

H.3 Finally, with the information obtain at step H.2, find the scheduling, production levels, and the operation cost of the time horizon $\{1, \ldots, \mathcal{T}\}$.

Chapter 4

Numerical Results

In this chapter we report some numerical experiences with the implementation of the algorithms described on chapter 3. Section 4.1 has a description of the algorithm implemented, in section 4.2 presents preliminary tests with MATLAB code.

4.1 Implementation

The implementation of the algorithms was done in MATLAB, and in FORTRAN. The MATLAB DP2 code does not consider ramping constraints. DP2 is implemented in FORTRAN with ramping constraints, the optimization problem (3.15) was implemented by André Diniz (CEPEL) and the problem was solved with `plcbas` package. `plcbas` references are listed below:

> E. Casas, C. Pola. 'PLCBAS,User's Guide', Technical Report Num. 1, Departamento de Matemáticas, Estadística y Computación, Univ. Cantabria, Serie Computación, Octubre 1989.
>
> E. Casas, C. Pola. 'An algorithm for indefinite quadratic programming based on a partial Cholesky factorization'. *RAIRO-Recherche Opérationnelle/Operations Research*, vol. 27, nº 4, 401-426, 1993.

We explain now the main structure of the MATLAB code, because the FORTRAN code follows a similar structure.

Functions used by DP1 and DP2 The code has a group of standard functions, described below:

- `curves.m`. The output of this function is: a vector p of dimension t^{up}/t^{dn} with the production of the unit in the start-up/shutdown curve, and the vector *inc* of dimension t^{up}/t^{dn} with the accumulate constant curve cost, $inc(t^{up}/t^{dn})$ is given by (3.8).
- `costcra.m`. The output of this function is the start-up/shutdown curve cost given by equations (3.9)/(3.10), if the unit completes the start-up/shutdown process or it is given by (3.12)/(3.13), if the unit did not complete the start-up/shutdown process.
- `potmin.m`. This function solves the minimization problem (3.14).
- `startup.m`. This function calculates the start-up cost. It can be fixed, or time dependent with exponential growth.
- `ecodispatch.m`. This function puts in a vector the optimal production level of the scheduling.
- `extend.m`. This function extends the vectors if we work with the approach explained in section 3.5.

DP Alternative 1 The main functions are:

- `dynpro.m`. The output is the optimal scheduling, production level at each time step, and the minimal cost. This function calls `coste.m` or `costcurve.m`.
- `coste.m`. Does the DP1 alternative without *up/dn* states.
- `costecurves.m`. Does the DP1 alternative with *up/dn* states.

DP Alternative 2 The main functions are:

- `dynplo.m`. The output is the optimal scheduling, production level at each time step, and the minimal cost. Dynplo implements the DP2 alternative, and works with *up/dn* states. This function calls `setgraph.m`
- `setgraph.m`. Builds the new transition graph, as explained in section 3.3.2.

4.2 Preliminary Results in MATLAB

The tests done in MATLAB code were the following:

i. For a given λ, compute manually the solution of the problem, and then compare the solution found by the algorithms.

ii. Comparision of the solutions found by dynpro and dynplo.

iii. Test if the scheduling found by the algorithms satisfies constraints (3.2)-(3.5).

iv. Test of consistency in the initial data, (cf. Remark 2, and Remark 3. p.30).

v. Comparision of the execution times of the algorithms.

In item i. we computted the solution a priori, for a time horizon of 8 hours. The solutions found "by hand" and those found by the algorithms were the same.

For item ii., we considered longer time horizons, and the solutions found by both dynpro and dynplo are the same.

In all the cases, the scheduling found satisfies constraints (3.2)-(3.5), as well as Remark 2, i.e., if the unit does no satisfies constraint (3.5), and Remark 3, i.e., if the unit is *up/dn*.

We observe that, for positive values of Lagrangian multiplier, the unit tends to be off-line. For negative values, the unit tends to be on-line.

For illustration, we present one example: The tables 4.1, 4.2 have the unit parameters, and start-up/shutdown curves data, respectively.

Table 4.1: Unit Time Data.

t^{up}	t^{dn}	T^{on}	T^{off}	$\underline{p}$	$\overline{p}$	a_0	a_1	a_2	α, β, τ	H
2	3	1	1	2.4	11	1	2	0.05	(40,30,10)	10

In Table 4.1, a_0, a_1, a_2 are the coefficients of the quadratic thermal cost function F, and (α, β, τ) are the coefficients of the exponential start-up cost. Start-up and shutdown curves are continuous piecewise affine functions, Table 4.2 has the necessary information to build these functions. In this

Table 4.2: Unit Start-up/Shutdown Curve.

Kinks t	Start-up generation pu	Time t	Shutdown generation pd
0	0	0	2.4
0.5	1	1.5	2.3
1	2	2.5	1.8
1.5	2	3	0
2	2.4		

table, "'kinks"' t are the points where a change of slope occurs. For this example, the discrete time horizon is $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, and the values of the Lagrangian multiplier $\lambda = (\lambda_R, \lambda_T)$ are:

$$\begin{aligned} \lambda_R &= [-1, -2, -3, -4, -5, -6, -7, -8, 0, 0, 0, 0] \\ \lambda_T &= [-2, -4, -6, -8, -10, -12, -14, -16, 50, 7, 0, 1] \end{aligned}$$

In the algorithms, the stage *on* of the unit (3.4a) is positive, and the stage *off* (3.4b) is negative, and the states are represented by: *on*=1, *off*=0, *up*=2, *dn*=-1.

The vector *ini* has the initial state and stage of the unit. The matrix *path* has the optimal scheduling, each column of path represent [state;time step;stage]. The optimal scheduling and minimum cost, for different initial values *ini* are:

- *ini*=[1,0]. The unit is on-line, and since $T^{\text{on}} = 1$, it must remain on-line at least one hour, in order to satisfy constraint (3.5a). The results are:

```
>> D.path (dynpro)

ans =

1     1     1     1     1     1     1     1     1     0
0     1     2     3     4     5     6     7     8    12
0     1     2     3     4     5     6     7     8    -1
>> D.cost = -579.7740

>> C.path (dynplo)
```

```
ans =

     1     1     1     0     0
     0     1     8    11    12
     0     1     8     0    -1

>> C.cost = -579.7740
```

In DP1 the graph is explored step by step, for this reason, dynpro prints the schedule at each time step explored. By contrast, DP2 explores the graph by stage-up or stage-down, so dynplo prints the schedule by stages of the graph. As mentioned it was observe that, for negative values λ, the unit tends to be on-line, and for positive values, the unit tends to be off-line. The transition between [1;8;8] and [0;12;-1] satisfies the constraints of the problem, because at time 8 the decision is to shutdown the unit since $t^{\text{dn}} = 3$, we have that the unit is off-line at time 11 and then it must satisfy minimum down times, $T^{\text{off}} = 1$, at time 12 the stage *off* of the unit is 1 (represented in the algorithm by -1).

- *ini*=[-1,1]. In this case, the unit is in shutdown process with stage one hour, which means that the unit must spend 2 hours completing the process of shutdown (3.3b), because $t^{\text{dn}} = 3$, then, the unit must satisfy constraint (3.5b), T^{off}, so we can make decision just after 3 hours. The results are:

```
>> D.path

-1     0     0     1     1     1     0
 0     2     3     6     7     8    12
 1     0    -1     1     2     3    -1

D.cost = -201.1990

>> C.path

-1     0     0     1     1     0     0
 0     2     3     5     8    11    12
```

```
 1     0    -1     0     3     0    -1

C.cost= -201.1990
```

We already mentioned that the algorithms start to make decisions once the unit satisfies minimum down times (3.5b). Since the multipliers are negative (for $t = 4, 5, 6, 7, 8$), the unit tends to be on-line, the transition between [0;3;-1] and [1;6;1] satisfies start-up (3.3a) and minimum up times (3.5a). Finally, for $t = 8, 9, 10, 11, 12$ the multipliers are positive, so the unit tends to be off-line. As noted for the first item, all the transitions satisfy the constraints of the problem.

We compare the execution time of dynplo and dynpro with 30 different cases in Figure 4.1. In almost all the cases the execution time of dynplo (green line) is lower than the one of dynpro (red line). We mention that dynplo code could be improved and be even faster. For example, all the auxiliary sinks of the graph can work as one sink, i.e., they may mark an exit of the main loop of the algorithm, so that less stages are explored in the graph.

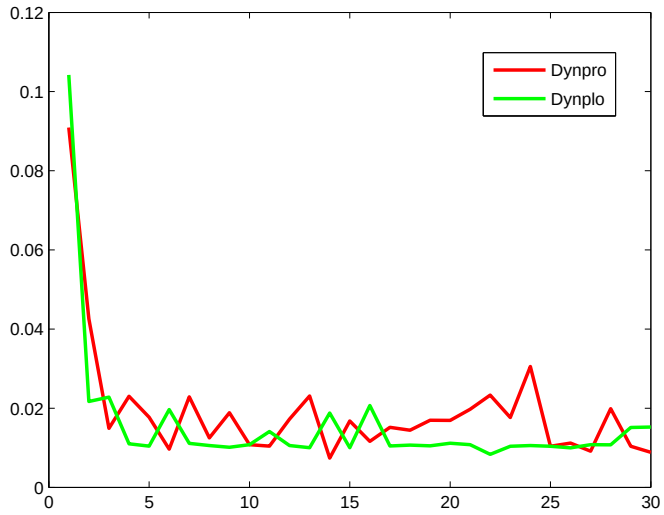

Figure 4.1: Graph 1 vs Graph 2.

Chapter 5

Conclusions and Future Work

We address in this work the thermal unit commitment problem, an optimization problem over a graph, in a hydrothermal power system. Two main topics were discussed, namely the modeling and solution strategy for the thermal problem considering or not ramping constraints. We give in details the graph used to model the problem with and without ramping constraints. With respect to the solution strategy used, we did a forward exploration of the graphs, we additioned the constraints (3.3) which are not considered in [2, 12, 8, 6], and we introduce the auxiliary states *up* and *dn*.

5.1 Conclusions.

In the case without ramping constraints DP1 (cf. section 3.3.1), unlike backwards approaches [12, 7], an expansion of the state space of the graph until the cooling time of the unit is not necessary because with a forward approach it is possible to know the stage of the unit at each time step, (cf. p.31). Hence, a time dependent start-up cost can easily be taken into account. We create the nodes (on, 0) and (off, 0) in order to add the constraints (3.3) in the graph, Figure 3.5.

In the case with ramping constraints DP2 (cf. section 3.4.1), a new graph was build. In order to add the constraints (3.3) we improve the graph introduced in work [6], representing at each stage of the graph the possible times when the unit finish a start-up/shutdown process.

In both DP alternatives, we create the auxiliary states *up*,*dn* to allow

changes of the state of the unit at the end of the time horizon.

After some numerical experience, we observe that DP2 without ramping constraints is more eficient that DP1 (cf. Figure 4.1).

5.2 Future Work.

As we mentioned in chapter 4, we want to improve DP2 treating each auxiliary sink of the graph, Figure 3.7, as one sink, i.e., they may mark an exit of the main loop of the algorithm, so that less stages are explored in the graph.

Finally, the FORTRAN code will be part of the security constrained short-term hydrothermal Unit Commitment model DESSEM, developed by CEPEL.

Appendix A

Matlab Codes

This appendix contains the codes implemented in MATLAB.

Standard Functions

```
function [C]=costcra(mod,l_a,p,tup,tdn,t,T,inc,ref)

% F=[A0,A1,A2]
% L_A LAMBDAS ARTIFICIALES
% P POTENCIAS EN LA CURVA STARTUP/SHUTDOWN
% D ES TUP/TDN DEPENDE DEL COSTO DE LA CURVA QUE
% QUEREMOS CALCULAR
% t PASO TEMPORAL
% T tamanho del horizonte
% mod=2 up; mod=4 down
% ref tiempo que falta para terminar la curva

% C COSTO CURVAS STARTUP/SHUTDOWN calculado con integral

if nargin<9
    ref=0;
end

if mod==2
d=tup;
elseif mod==4
```

```
d=tdn;
else
error('unknown curve')
end

if ref<0
error('ref must be positive')
elseif ref>d
error('ref must be lower than tup/tdn')
end

if d+t-ref>T
error('It is not possible to startup/shutdown at this time')
end

C=0;

for i=1:d-ref
C=C+l_a(t+i)*p(i);
end
C=C+inc(d-ref);

function [R]=curves(mod,x,y,tup,tdn,F)

% Esta funcion calcula las potencias generadas
por la unidad cuando esta en proceso start-up/shutdown
% INPUT:
% mod entero 2,4 indica si la rampa es de startup (2) o
% shutdown (4)
% (x,y) son los puntos de las rampas
% tup, tdn son los tiempos de startuo/shutdown
% t es el paso temporal
% F=[a0;a1;a2]
% l_a multiplicadores de lagrange "artificiales"

% Las curvas de startup/shutdown describen el
comportamiento de la unidad cuando en el intervalo de tiempo
% t+1, ..., t+d donde d=tup/tdn. Estas curvas generalmente
```

```
se aproximan por una funcion continua lineal (afin) por partes
% las lista de puntos (x,y)
son los puntos donde un cambio de pendiente ocurre.

% Consideraciones de las curvas: Start-up la
% curva es lineal por partes cresciente,
% la generacion comienza en cero
% Shutdown la curva es afin, lineal por partes,
% corta el eje x, la generacion termina en cero
% tomar en cuanta que en las rodadas cosidermos
% que genera de pmin a 0 ; 0 a pmin

% OUTPUT: R
% R vector de las potencias generadas en los tiempos t+1,.., t+d

if mod==2
d=tup;
elseif mod==4
d=tdn;
else
error('unknown d')
end
if d>0
if d~=x(length(x))
error('tup/tdn different of start-up/shutdown total time!')
end
elseif d<0
error('d must be positive')
elseif d==0
p=0;
end
k=length(x);
ktemp=length(y);
if ktemp~=k
error('dimension of y must be the same of x')
end
clear ktemp
```

```
% CALCULANDO PENDIENTES

for i=1:k-1
m(i)=(y(i+1)-y(i))/(x(i+1)-x(i));
end

% CALCULANDO POTENCIAS t+1, .. ,t+d

% para t+1,..,t+d
% encuentra la pendiente de generacion esta

j=1;
while (j<k)
for i=1:d %para que funcione si delta=0.5 hay que poner 1;0.5:d
if (x(j)<i)
if (i<=x(j+1))
p(i)=y(j+1)+(i-x(j+1))*m(j);
pos(i)=j;
end
end
end
j=j+1;
end

for i=1:k-1
coef(i)=y(i)-m(i).*x(i);
delta(i)=x(i+1)-x(i);
delta2(i)=x(i+1)^2-x(i)^2;
delta3(i)=x(i+1)^3-x(i)^3;
end

A= F(1)+F(2)*coef+F(3)*(coef.*coef);
% A(i) es para integrar entre ti-t-ti+1
B=0.5*F(2)*m+F(3)*(m.*coef);
C=(1/3)*F(3)*(m.*m);

Aint=A.*delta;
Bint=B.*delta2;
```

```
Cint=C.*delta3;

int=Aint+Bint+Cint;
sum(int);
for i=1:d
inc(i)=sum(int(1:pos(i)-1))+A(pos(i))*(i-x(pos(i)))
+B(pos(i))*(i^2-x(pos(i))^2)+C(pos(i))*(i^3-x(pos(i))^3);
end
R.p=p; %vector con las potencias generadas en los pasos t+1,..,t+d
R.m=m;
%R.coef=coef;
%R.int=int;
R.inc=inc;

function [P]=potmin(F,pot,lambda)

% F=[a0,a1,a2] funcion de costo a0+a1p+a2p^2
% pot=[pmin,pmax] rango de potencias de la unidad
% lambda=[lambda_reserva;lambda_artificial] matriz Tx2
% T discretizacion temporal; t=1,...,T
% esta rutina calcula el minimo de la funcion
% para t=1,..,T
% a0+a1q+a2q^2+lambda_reserva*(qmax-q)+lambda_artificial*q
%

T=length(lambda);
for t=1:T
pmin(t)=0.5*(lambda(t,1)-lambda(t,2)-F(2))/(F(3));
if pmin(t) <pot(1)
pmin(t)=pot(1);
elseif pmin(t)>pot(2)
pmin(t)=pot(2);
end

Fmin(t)=F(1)+lambda(t,1)*pot(2)+(F(2)-lambda(t,1)
+lambda(t,2))*pmin(t)+F(3)*pmin(t)^2;
end
```

```
P.p=pmin;
P.F=Fmin;


function [st]=startup(parameter)

%PARAMETROS PARA COSTO FIJO: parameter=[fixed]
%fixed es el costo fijo
%PARAMETROS PARA COSTO VARIABLE:
%parameter=[alpha;beta;tau;toff]
%toff tiempo que la maquina esta en el estado off, debe ser negativo
%alpha, beta, tau parametros especificos de la unidad, son positivos
%alpha+beta se conoce como cold startup
%tau es el tiempo caracteristico de incremento del costo start-up
% observacion: si se considera el st variable, parameter es
%              de dim 4 si st es fijo parameter es de dim 1

m=length(parameter);

if m==4
if parameter(4)==inf
st=0;
elseif parameter(4)>0
error('toff must be negative')
else
st=parameter(1)+parameter(2)*(1-exp(parameter(4)/parameter(3)));
end
elseif m==1
st=parameter;
elseif m==2
st=parameter(1);
else
error('parameter must be a vector of  dimension 1,2
(fixed) or dimension 4 (variable)')
end

function [ped]=ecodispatch(path,potmin,pu,pd,potini)
```

```
m=length(path);
ped=zeros(m,1);
for i=1:m
if path(1,i)==1
if path(2,i)>0
ped(i)=potmin(path(2,i));
elseif path(2,i)==0
ped(i)=potini;
end
elseif path(1,i)==-1
ped(i)=pd(path(3,i));
elseif path(1,i)==2
ped(i)=pu(path(3,i));
end
end
function [z]=extend(x,y)

%extend, duplica el vector x=(x1,x2,x3,...,xn)
%como y=(x1,..,xn,x1,..,xn)
%y tiene dimension 2n

if nargin<2
y=x;
end

n=length(x);
m=length(y);

for i=1:n
z(i)=x(i);
end
for i=1:m
z(i+n)=y(i);
end
```

Dynpro

```
function [C]=dynproe(mode,ini,time,potini,pot,
```

```
lambda,F,parameter,H,zu,zd,y,timeini,delta)

%aqui el tamanho del paso temporal es delta unidades son horas
%ini=[on/off/up/dn;stage] dato inicial estado, tiempo en ese  estado
%time=[tup;tdn;ton;toff] tiempos de rampa/must
%de la unidad todos son enteros . must es min up/down times
%potini es la potencia inicial si la unidad esta en ON o en curva
%pot=[pmin;pmax] rango de potencias
%lambda=[lambda reserva, lambda arificial] matriz T*2
%T es la discretizacion del tiempo, t=1,...,T
%F=[a0;a1;a2] funcion de costo
%parameter=[alpha,beta,tau] o st startup fijo
%y=[yr;ya] matriz para extender los lambdas
%timeini es el tiempo inicial en el que empezamos
%la programacion dinamica.

%mode=1 sin extend y nodos on/off
%mode=2 con extend y nodos on/off
%mode=3 sin extend y nodos on/off/up/dn
%mode=4 con extend y nodos on/off/up/dn

%ini(1) tiene las siguientes opciones ON=1, OFF=0
%en mode=3, mode=4 acepta los estados UP=2, DN= -1
%significa que esta en
%proceso de startup, shutdown respectivamente

%stage es positivo/negativo si es on/off 0 si esta recien on/off
%stage en modo curva up/down son positivos y menores que tup/tdn
%necesita extend.m duplica los vectores, para permitir cambios
%necesita coste.m rutina principal, costo t=0 --> t mode1,2
%necesita costecurve.m mode 3,4
%necesita curves.m calcula el costo curva up/down
%necesita path.m calcula el camino optimo
%necesita order.m ordena el camino path desde ini_node a last_node
%necesita pathcurve.m para poner los estados up/dn
%necesita ecodispatch.m para informar el economical dispatch
%necesita nlast.m calcula el nodo en T en extend
%necesita nodof.m caclula los nodos con tiempos t1<T<=t2
```

```
%necesita startup.m calcula el costo startup
%necesita potmin.m calcula el individual dispatch

if nargin < 15
delta=1;
end

%*****************************************************************
% verifica coherencia entre mode y estado inicial
%*****************************************************************

if((mode==1)|(mode==2))
if ((ini(1)==-1)|(ini(1)==2))
error('mode=1 or mode=2 does not work with up/down states')
end
end

%*********************************************************************
% verifica los datos del startup cost
%*********************************************************************
m=length(parameter);
if (m==1)|(m==2)
stcost='fixed';
elseif m==3
stcost='variable';
else
error('parameter must be of dimension 1,2 or 3')
end

%*********************************************
%***** datos de la rutina ***************************

T=length(lambda);
P=potmin(F,pot,lambda);
Q=potmin(F,pot,y);
% datos extend para la rutina y curvas
if ((mode==2)|(mode==4))
Fmin=extend(P.F,Q.F);
```

```
l_a=extend(lambda(:,2),y(:,2));
K=length(y);
potencias=extend(P.p,Q.p);
elseif((mode==1)|(mode==3))
l_a=lambda(:,2);
Fmin=P.F;
potencias=P.p;
K=0;
else
error('Unknown mode')
end

%***** datos para las curvas de startup/shutdown ******************

xu=zu(:,1);
yu=zu(:,2);
xd=zd(:,1);
yd=zd(:,2);

% calculando potencias en las curvas

ru=curves(2,xu,yu,time(1),time(2),F);
rd=curves(4,xd,yd,time(1),time(2),F);
pu=ru.p;
pd=rd.p;
incu=ru.inc;
incd=rd.inc;

% ******* ini costos ********************************

coff=inf*ones(T+K+time(4)+time(2)+1,1);
con=inf*ones(T+K+time(3)+time(1)+1,1);

%*********************************************************

%**********comienza la rutina Foward*********************

% estado inicial es OFF
```

```
if ini(1)==0
if ini(2)>0
error('stage off must be negative')
else
coff(1)=0;
tempoff=ini(2)+time(4);
if tempoff>0
for i=1:tempoff
coff(i+1)=0; %no permite el cambio de estado,
% avanza al nodo de decision costo 0ff-->off es cero
end

if((mode==1)|(mode==2))
g=coste([0;-time(4)],coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta,tempoff);
elseif((mode==3)|(mode==4))
g=costcurve([0;-time(4)],coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta,tempoff);
end
for i=1:tempoff
g.poff(:,i)=[0;i-1]; % inserta los nodos en que no cambiamos el estado
g.soff(i)=ini(2)-(i-1)*delta; % contea el estado
end
else
if((mode==1)|(mode==2))
g=coste(ini,coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta); % calcula los posibles caminos
elseif((mode==3)|(mode==4))
g=costcurve(ini,coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta); % calcula los posibles caminos
end
end
end
% mismo procedimiento si el estado inicial es ON

elseif ini(1)==1
if ini(2)<0
error('stage on must be positive')
```

```
else
con(1)=0;
tempon=time(3)-ini(2);
if tempon>0
for i=1:tempon
con(i+1)=sum(Fmin(1:i));
end
if((mode==1)|(mode==2))
g=coste([1;time(3)],coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta,tempon);
elseif((mode==3)|(mode==4))
g=costcurve([1;time(3)],coff,con,parameter,H,time,T+K,
Fmin,F,l_a,pu,pd,incu,incd,delta,tempon);
end
for i=1:tempon
g.pon(:,i)=[1;i-1];
g.son(i)=ini(2)+(i-1)*delta;
end
else
if((mode==1)|(mode==2))
g=coste(ini,coff,con,parameter,H,time,
T+K,Fmin,F,l_a,pu,pd,incu,incd,delta);
elseif((mode==3)|(mode==4))
g=costcurve(ini,coff,con,parameter,H,time,
T+K,Fmin,F,l_a,pu,pd,incu,incd,delta);
end
end
end
% aqui es estado dn, en proceso de shutdown trabaja con mode=3,4

elseif ini(1)==-1

tgo=time(2)-ini(2);
if tgo<0
error('In shutdown process ini(2) must be lower than tdn');
elseif ini(2)==0
error('In shutdown process ini(2) can not be zero');
elseif ini(2)<0
```

```
error('In shutdown process ini(2) must be positive');
else
% aqui termina el proceso de rampa,
calcula el costo de rampa y llega al estado off
cdn=0;
for i=1:tgo
cdn=cdn+l_a(i)*pd(ini(2)+i);
end
cdn=cdn+incd(time(2))-incd(ini(2));
coff(tgo+1)=cdn;
% esto es el must off

for i=1:time(4)
coff(tgo+1+i)=cdn;
end
end
if ((mode==3)|(mode==4))
g=costcurve([0;-time(4)],coff,con,parameter,H,time,T+K,Fmin,
F,l_a,pu,pd,incu,incd,delta,tgo+time(4));
g.poff(:,tgo)=[-1;0];
for i=1:time(4)
g.poff(:,tgo+i)=[0;tgo+i-1];
g.soff(tgo+i)=-(i-1)*delta;
end
end
% aqui es estado up, en proceso de startup trabaja solo con mode=3,4
elseif ini(1)==2
tgo=time(1)-ini(2);
if ini(2)>=time(1)
error('In startup process ini(2) must be lower than tup');
elseif ini(2)==0
error('In startup process ini(2) can not be zero');
elseif ini(2)<0
error('In startup process ini(2) must be positive');
else
% aqui termina el proceso de startup,
calcula el costo de la curva y llega al estado on
cup=0;
```

```
for i=1:tgo
cup=cup+l_a(i)*pu(ini(2)+i);
end
cup=cup+incu(time(1))-incu(ini(2));
con(tgo+1)=cup;
% esto es el must on
for i=1:time(3)
con(tgo+1+i)=cup+sum(Fmin(tgo+1:tgo+i));
end
end
if((mode==3)|(mode==4))
g=costcurve([1;time(3)],coff,con,parameter,H,time,T+K,Fmin,
F,l_a,pu,pd,incu,incd,delta,tgo+time(3));
g.pon(:,tgo)=[2;0];
for i=1:time(3)
g.pon(:,tgo+i)=[1;tgo+i-1];
g.son(tgo+i)=(i-1)*delta;
end
end
end

% Compara los costos finales
de ON T+K,...,T+K+Tup+Ton y OFF T+K,...,T+K+Tdn+Toff para
% hallar el camino optimo, en fortran hace solo en T, no
% es necesario hace para t>T

temon=g.on(T+K+1:length(g.on));
temoff=g.off(T+K+1:length(g.off));
[minon,kon]=min(temon);
[minoff,koff]=min(temoff);

if minon<minoff
last=[1;kon+T+K-1];
%marcamos el ultimo nodo del camino optimo
cost=minon;
else
last=[0;koff+T+K-1];
cost=minoff;
```

```
end

%****************** construye el camino

S=path([ini(1);0],last,g.poff,g.pon);
Ss=pathcurve(mode,order(S),ini,g.son,g.soff,g.nson,g.nsoff);
%ordenar el camino de ini a last y
% poner el stage en cada nos path(:,i)=[estado;tiempo;stage]

%****************************************

C.off=g.off; %  (*), costo off
C.on=g.on; % costo on (*)
%C.pon=g.pon; % nodos previos a un nodo on (*)
%C.poff=g.poff; % nodos previos a un nodo off (*)
%%C.cost=cost; % indica el costo
%C.son=g.son; % stage on en tiempo t (*)
%C.soff=g.soff; % stage off en tiempo t (*)
%C.nsoff=g.nsoff; % stage curva down (*)
%C.nson=g.nson; % stage curva up (*)
%C.last=last; % nodo last (*)


%*********************************************************
%si mode=2,4 debemos informar el costo hasta
%T y el nuevo nodo last
%*********************************************************

if ((mode==2)|(mode==4))
[nlast,prevn,posn]=nodof(Ss,T);

for i=1:posn-1
C.path(:,i)=Ss(:,i);
end

[costy,newlast]=costefinal(nlast,prevn,g.on,g.off,pu,pd,incu,incd,
F,l_a,Fmin,parameter,H,time,T,K);
C.path(:,posn)=newlast;
```

```
C.cost=costy;
C.costext=cost;
C.ext=Ss;
elseif ((mode==1)|(mode==3))
C.path=Ss;
C.cost=cost;
end

%****************************************
% economical dispatch
%****************************************
ed=ecodispatch(C.path,potencias,pu,pd,potini);
C.ed=ed;
%aqui ponemos el timeini en path ************************
if timeini>0
for i=1:length(C.path)
C.path(2,i)=C.path(2,i)+timeini;
end
if ((mode==2)|(mode==4))
for i=1:length(Ss)
C.ext(2,i)=C.ext(2,i)+timeini;
end
end
elseif timeini<0
fprintf('initial time is positive. I worked with timeini=0');
end

function [g]=costcurve(sini,coff,con,parameter,H,time,T,Fmin,F,
l_a,pu,pd,incu,incd,delta,tempo)

%sini=[state,stage] estado on/off tiempo en ese
%estado coff, con vectores de inicializacion del costo
%dados en dynpro st start-up cost de la unidad
%T es el tiempo final de la discretizacion temporal
%time=[tup;tdn;ton;toff] tiempos de rampa/must de la
%unidad todos son enteros
%P estructura; P.F, P.p son los valores minimos
%de la funcion lagrangeana y la potencia, dados por potmin
```

```
%delta el tamanho del paso temporal
%tempo, dato opcional, por si el estage inicial en
%dynpro no satisface min up/down time

%COSTO calcula el costo (minimo) total en los nodos
%de decision desde t=0 a t
%construye las matrices prevon/off que tienen los
%nodos previos a un estado on/off
%construye soff/on es un vector con el stage en un
%tiempo t en el estado off/on

%observacion: esta rutina considera solamente los nodos
%de decision, decir los nodos que cumplen la restriccion
%min up/down times.en esta rutina consideramos los estados
%curva up/down;para verificar si la unidad termina
%en un estado  de curva
%construimos las matrices nsoff, nson para marcar
%el estado up/down
%cuanto tiempo esta en el proceso de rampa.

%************* RUTINA PRINCIPAL DE DYNPRO ***************

if nargin < 16
tempo=0;
end

%******************** inicializacion de datos **********

t=tempo;
prevoff=inf*ones(2,T+time(2)+time(4));
prevon=inf*ones(2,T+time(1)+time(3));
soff=inf*ones(1,T+time(2)+time(4)+1);
nsoff=inf*ones(2,T+time(2)+time(4)+1);
son=inf*ones(1,T+time(1)+time(3)+1);
nson=inf*ones(2,T+time(1)+time(3)+1);

%*****************************************************
```

```
if sini(1)==1
son(t+1)=sini(2);
%marcamos el stage inicial en el estado on/off
else
soff(t+1)=sini(2);
end

%********************** loop principal ***************
%aqui calculamos el costo minimo de transicion
%off-->off, off-->on, on-->on, on-->off
% note que para las transiciones hay que tomar en
%cuenta curva up/down times,
%min up/down times, por eso sumamos los tiempos
%down/up en las
%transiciones on-->off, off-->on
%marcamos si hay un cambio de estado apuntando
%los nodos previos
%y calculamos el stage

while (t<T)

%posible transicion off-off

[coff(t+2),k1]=min([coff(t+2);coff(t+1)]);
if (coff(t+2)~=inf)&(k1==2)
prevoff(:,t+1)=[0;t];
soff(t+2)=soff(t+1)-delta;
nsoff(:,t+2)=[0;0];
end

%posible transicion off-on, aqui prestamos
%atencion al tiempo de rampa
%si t+tup>T permite proceso de rampa,
%avanza hasta en tiempo T
%si t+tup+1>T no permite must on
%por los multiplicadores
%(se necesitan los de la proxima iteracion),
%solo permite la curva: estado es ON
```

```
%y el stage 0

%CALCULA LA curva SI ES POSIBLE START-UP
%SINO ES POSIBLE PONEMOS INF PARA FORZAR QUE
%NO HAGA STARTUP

if  t+time(1)<=T
ru=costcra(2,l_a,pu,time(1),time(2),t,T,incu);
%VAMOS A CALCULAR EL MUSTON COST
muston=0;
if time(3)==0
tmuston=t+time(1);
monstage=0;
if time(1)==0
monstage=1;
end
else
monstage=0;
for i=1:time(3)
if t+time(1)+i<=T
muston=muston+Fmin(t+time(1)+i);
tmuston=t+time(1)+i;
monstage=monstage+1;
end
end
end
%posible transicion off--on
st=startup([parameter;soff(t+1)]);
[con(tmuston+1),k2]=min([con(tmuston+1);
coff(t+1)+muston+st+ru]);
if tmuston~=0
if (con(tmuston+1)~=inf)&(k2==2)
prevon(:,tmuston)=[0;t];
son(tmuston+1)=monstage;
nson(:,tmuston+1)=[1;0];
end
end
else
```

```
ref=T-t;
ref=time(1)-ref;
ru=costcra(2,l_a,pu,time(1),time(2),t,T,incu,ref);
tup=T;
muston=0;
%posible transicion off--on *estado up*.
%no cuenta el stage porque la maquina no
%esta en on pero el costo
%lo incluimos en con
[con(tup+1),k2]=min([con(tup+1);coff(t+1)+muston+st+ru]);
if (con(tup+1)~=inf)&(k2==2)
prevon(:,tup)=[0;t];
nson(:,tup+1)=[2;time(1)-ref];
end
end

%posible transicion on-on
[con(t+2),k3]=min([con(t+2);con(t+1)+Fmin(t+1)]);
if (con(t+2)~=inf)&(k3==2)
prevon(:,t+1)=[1;t];
son(t+2)=son(t+1)+delta
nson(:,t+2)=[1;0];
end

%posible transicion on-off, aqui prestamos
%atencion al tiempo de curva
%si t+tdn>T permite estado rampa,
%avanza hasta el tiempo T
%aqui no hay problemas con el mustoff
%porque el peso del arco
%(off,t)-(off,t+1) es cero. genera desde pmin hasta cero!

if t+time(2)<=T
rd=costcra(4,l_a,pd,time(1),time(2),t,T,incd);
[coff(t+time(2)+time(4)+1),k4]=min([coff(t+time(2)+time(4)+1);
                                    con(t+1)+rd+H]);
if t+time(2)~=0
if time(4)~=0
```

```
if (coff(t+time(2)+time(4)+1)~=inf)&(k4==2)
prevoff(:,t+time(2)+time(4))=[1;t];
soff(t+time(2)+time(4)+1)=-time(4);
nsoff(:,t+time(2)+time(4)+1)=[0;0];
end
else
if (coff(t+time(2)+time(4)+1)~=inf)&(k4==2)
prevoff(:,t+time(2)+time(4))=[1;t];
soff(t+time(2)+time(4)+1)=-time(4)-1;
nsoff(:,t+time(2)+time(4)+1)=[0;0];
end
end
end
else
ref=T-t;
ref=time(2)-ref;
rd=costcra(4,l_a,pd,time(1),time(2),t,T,incd,ref);
[coff(T+1),k4]=min([coff(T+1);con(t+1)+rd+H]);

%aqui no contamos el stage off porque la maquina aun
%no esta off pero el costo lo incluimos en coff.
%marcamos stage rampa down

if (coff(T+1)~=inf)&(k4==2)
prevoff(:,T)=[1;t];
nsoff(:,T+1)=[-1;time(2)-ref];
end
end
t=t+1;

clear k1 k2 k3 k4 ru rd
end

g.off=coff;
g.on=con;
g.poff=prevoff;
g.pon=prevon;
g.soff=soff;
```

```
g.son=son;
g.nson=nson;
g.nsoff=nsoff;


function [g]=coste(sini,coff,con,parameter,H,time,T,Fmin,F,
l_a,pu,pd,incu,incd,delta,tempo)

%sini=[state,stage] estado on/off tiempo en ese estado
%coff, con vectores de inicializacion del costo dados en dynpro
%st start-up cost de la unidad
%T es el tiempo final de la discretizacion temporal
%time=[tup;tdn;ton;toff] tiempos de rampa/must de la
%unidad todos son enteros
%P estructura; P.F, P.p son los valores minimos de la funcion
%lagrangeana y la potencia, dados por potmin
%delta el tamanho del paso temporal
%tempo, dato opcional, por si el estage inicial en dynpro
%no satisface min up/down time

%COSTO calcula el costo (minimo) total en los nodos de
%decision desde t=0 a t
%construye las matrices prevon/off que tienen los nodos
%previos a un estado on/off
%construye soff/on es un vector con el stage en un tiempo t en el
%estado off/on

%observacion: esta rutina considera solamente
%los nodos de decision, es
%decir los nodos que cumplen la restriccion
%min up/down times.
%esta rutina no permite el cambio de estado al
%tiempo t si t+tup/tdn>T
%en los nodos que no se pueda avanzar en must-on y
%si rampear, permite el
%cambio y marca el estado,stage como on,0
%no hay problemas con must-off porque no
%necesitamos los multiplicadores
```

```
%************* RUTINA PRINCIPAL DE DYNPRO ***************************

if nargin < 16
tempo=0;
end

%******************** inicializacion de datos ********************

t=tempo;
prevoff=inf*ones(2,T+time(2)+time(4));
prevon=inf*ones(2,T+time(1)+time(3));
soff=inf*ones(1,T+time(2)+time(4)+1);
son=inf*ones(1,T+time(1)+time(3)+1);
nson=inf*ones(2,T+time(1)+time(3));
nsoff=inf*ones(2,T+time(2)+time(4));
%*****************************************************************

if sini(1)==1
son(t+1)=sini(2);
%marcamos el stage inicial en el estado on/off
else
soff(t+1)=sini(2);
end

%************************ loop principal ******************************
%aqui calculamos el costo minimo de transicion
%off-->off, off-->on, on-->on, on-->off
%note que para las transiciones hay que tomar en
%cuenta ramp up/down times,
%min up/down times, por eso sumamos los tiempos down/up en las
%transiciones on-->off, off-->on
%marcamos si hay un cambio de estado apuntando
%los nodos previos y calculamos el stage

while (t<T)

%posible transicion off-off
```

```
[coff(t+2),k1]=min([coff(t+2);coff(t+1)]);
if (coff(t+2)~=inf)&(k1==2)
prevoff(:,t+1)=[0;t];
soff(t+2)=soff(t+1)-delta;
end

%posible transicion off-on, aqui prestamos
%atencion al tiempo de rampa
%si t+tup>T no permite el cambio de estado
%si t+tup+1>T no permite must on por los
%multiplicadores(se necesitan los de la proxima iteracion),
%solo permite la rampa: estado es ON
%y el stage 0

%CALCULA EL COSTO DE LA CURVA SI ES POSIBLE START-UP
%SINO ES POSIBLE PONEMOS INF PARA FORZAR QUE NO
%HAGA STARTUP

if  t+time(1)<=T
ru=costcra(2,l_a,pu,time(1),time(2),t,T,incu);
% VAMOS A CALCULAR EL MUSTON COST
muston=0;
if time(3)==0
tmuston=t+time(1);
monstage=0;
if time(1)==0
monstage=1;
end
else
monstage=0;

for i=1:time(3)
if t+time(1)+i<=T
muston=muston+Fmin(t+time(1)+i);
tmuston=t+time(1)+i;
monstage=monstage+1;
end
```

```
end
end
else
ru=inf;
tmuston=t+time(1)+time(3);
end

%posible transicion off--on

%verificamos si el costo de startup es fijo o variable

st=startup([parameter;soff(t+1)]);

[con(tmuston+1),k2]=min([con(tmuston+1);coff(t+1)+muston+st+ru]);

if (con(tmuston+1)~=inf)&(k2==2)
prevon(:,tmuston)=[0;t];
son(tmuston+1)=monstage;
end

%posible transicion on-on
[con(t+2),k3]=min([con(t+2);con(t+1)+Fmin(t+1)]);
if (con(t+2)~=inf)&(k3==2)
prevon(:,t+1)=[1;t];
son(t+2)=son(t+1)+delta;
end

%posible transicion on-off, aqui
%prestamos atencion al tiempo de rampa
%si t+tdn>T no permite el cambio de estado
%aqui no hay problemas con el mustoff
%porque el peso del arco
%(off,t)-(off,t+1) es cero. genera desde pmin hasta cero!

if t+time(2)<=T
rd=costcra(4,l_a,pd,time(1),time(2),t,T,incd);
else
rd=inf;
```

```
end
[coff(t+time(2)+time(4)+1),k4]=min([coff(t+time(2)+time(4)+1);
                                    con(t+1)+rd+H]);
if t+time(2)~=0
if time(4)~=0
if (coff(t+time(2)+time(4)+1)~=inf)&(k4==2)
prevoff(:,t+time(2)+time(4))=[1;t];
soff(t+time(2)+time(4)+1)=-time(4);
end
else
if (coff(t+time(2)+time(4)+1)~=inf)&(k4==2)
prevoff(:,t+time(2)+time(4))=[1;t];
soff(t+time(2)+time(4)+1)=-1;
end
end
end
t=t+1;
clear k1 k2 k3 k4 ru rd
end

g.off=coff;
g.on=con;
g.poff=prevoff;
g.pon=prevon;
g.soff=soff;
g.son=son;
g.nson=nson;
g.nsoff=nsoff;
```

Auxiliary function of Dynpro In this paragraph, we put the main auxiliary functions of dynpro.

```
function [S]=pathcurve(mod,path,ini,son,soff,nson,nsoff)

m=length(path);
S(:,1)=[ini(1);0;ini(2)];
if((mod==1)|(mod==2))
for i=2:m
```

```
if path(1,i)==1
S(:,i)=[path(:,i);son(path(2,i)+1)];
else
S(:,i)=[path(:,i);soff(path(2,i)+1)];
end
end
elseif((mod==3)|(mod==4))
for i=2:m
if path(1,i)==1
if nson(1,path(2,i)+1)==2
S(:,i)=[2;path(2,i);nson(2,path(2,i)+1)];
else
S(:,i)=[path(:,i);son(path(2,i)+1)];
end
else
if nsoff(1,path(2,i)+1)==-1
S(:,i)=[-1;path(2,i);nsoff(2,path(2,i)+1)];
else
S(:,i)=[path(:,i);soff(path(2,i)+1)];
end
end
end
end
%este funcion pone el stage del
nodo en path y pone nodos up/dn
```

Dynplo

```
function[D]=dynplo(mode,ini,time,mult,F,pot,zu,zd,H,
                   parameter,yext,potini)

T=length(mult);
sz=size(mult);
lambda=[];
if sz(2)==1
lambda(:,1)=zeros(sz(1),1);
lambda(:,2)=mult;
else
```

```
lambda=mult;
end

sy=size(yext);
y=[];
if sy(2)==1
y(:,1)=zeros(sy(1),1);
y(:,2)=yext;
else
y=yext;
end

xu=zu(:,1);
yu=zu(:,2);
xd=zd(:,1);
yd=zd(:,2);

ru=curves(2,xu,yu,time(1),time(2),F);
rd=curves(4,xd,yd,time(1),time(2),F);
pu=ru.p;
pd=rd.p;
incu=ru.inc;
incd=rd.inc;

P=potmin(F,pot,lambda);
Q=potmin(F,pot,y);

if mode==2
Fmin=extend(P.F,Q.F);
l_a=extend(lambda(:,2),y(:,2));
K=length(y);
potencias=extend(P.p,Q.p);
elseif mode==1
l_a=lambda(:,2);
Fmin=P.F;
potencias=P.p;
K=0;
else
```

```
    error('Unknown monde')
end

S=setgraph(ini,time,T+K);
G=S.g;
timeini=S.time;
newini=[];
newini(1)=ini(1);
newini(2)=S.nin;
sinkis=S.sink;
dims=S.dim;
UpDn=S.ud;

sizec=size(G);
m=sizec(1); n=sizec(2);
C=inf*ones(m,n);
Prev=zeros(m,n-1);

if newini(2)==1
   nin=[0;1];
else
   nin=[1;0];
end

nin=extension(nin,n-4);
newini=[newini(1) newini(2) nin];
% deberia tener dimension n-1

source=1; sink=1;

Prev(:,1)=ones(m,1);
path=[];

if ini(1)==1
path(:,1)=[1;0;ini(2)];
dp=1;
if timeini>0
Cup=0;
```

```
for i=1:timeini
Cup=Cup+Fmin(i); % costo por estar on de 0-timeini
end
C(source,1)=Cup;
path(:,2)=[1;timeini;ini(2)+timeini];
dp=dp+1;
else
C(source,1)=0;
end
elseif ini(1)==0
path(:,1)=[0;0;ini(2)];
dp=1;
C(source,1)=0;
if timeini>0
path(:,2)=[0;timeini;ini(2)-timeini];
dp=dp+1;
end
newstage=ini(2);

elseif ini(1)== -1
path(:,1)=[-1;0;ini(2)];
if ini(2)==0
error('In shutdown process ini(2) can not be zero');
elseif ini(2)<0
error('In shutdown process ini(2) must be positive');
end
tgo=time(2)-ini(2);
if tgo<0
error('In shutdown process ini(2)
must be lower than tdn');
else
cdn=0;
for i=1:tgo
cdn=cdn+l_a(i)*pd(ini(2)+i);
end
cdn=cdn+incd(time(2))-incd(ini(2));
coff=0; % costo mustoff
C(source,1)=cdn+coff; % costo
```

```
end
newstage=-time(4);
path(:,2)=[0;tgo;0];
path(:,3)=[0;tgo+time(4);newstage];
dp=3;

elseif ini(1)==2
path(:,1)=[2;0;ini(2)];
tgo=time(1)-ini(2);
if ini(2)>=time(1)
error('In startup process ini(2) must be lower than tup');
elseif ini(2)==0
error('In startup process ini(2) can not be zero');
elseif ini(2)<0
error('In startup process ini(2) must be positive');
else
cup=0;
for i=1:tgo
cup=cup+l_a(i)*pu(ini(2)+i);
end
cup=cup+incu(time(1))-incu(ini(2));
%costo must on aqui seria por estar on tgo-ton
con=0;
for i=1:time(3)
con=con+Fmin(tgo+i);
end
C(source,1)=cup+con;
end
newstage=time(3);
path(:,2)=[1;tgo;0];
path(:,3)=[1;tgo+time(3);newstage];
dp=3;
end

if newini(2) == 1
for i=1:dims(2)
timeon=G(i,2)-time(2)-timeini;
con=0;
```

```
% aqui se debe calcular el costo por estar on de
% timeini+1->timeon
for k=1:timeon
con=con+Fmin(k+timeini);
end
if G(i,2)<=T+K
tstep=G(i,2)-time(2);
cdn=costcra(4,l_a,pd,time(1),time(2),tstep,T+K,incd);
C(i,2)=C(source,1)+con+cdn+H;
elseif G(i,2)>T+K
if G(i,2)~=sinkis(2)
ref=G(i,2)-T-K;
tstep=G(i,2)-time(2);
cdn=costcra(4,l_a,pd,time(1),time(2),tstep,T+K,incd,ref);
C(i,2)=C(source,1)+con+cdn+H;
else
C(i,2)=C(source,1)+sum(Fmin(timeini+1:T+K));
% costo on en todo el horizonte
end
end
end
elseif newini(2) == 0
coff=0;
for i=1:dims(2)
tstep=G(i,2)-time(1);
timoff=-tstep+newstage;
if G(i,2)<=T+K
cup=costcra(2,l_a,pu,time(1),time(2),tstep,T+K,incu);
C(i,2)=C(source,1)+cup+coff+startup([parameter;timoff]);
elseif G(i,2)>T+K
if G(i,2)~=sinkis(2)
ref=G(i,2)-T-K;
cup=costcra(2,l_a,pu,time(1),time(2),tstep,T+K,incu,ref);
C(i,2)=C(source,1)+cup+coff+startup([parameter;timoff]);
else
C(i,2)=C(source,1)+coff;
end
end
```

```
end
end

for j=3:n-1

if newini(j) == 1
for i=1:dims(j)
if G(i,j)<=T+K
for p=1:i
timeon=G(i,j)-time(2)-G(p,j-1); % timeon en cada (i,j)
con=0; % costo on en (i,j) de timeini+1->timeon
for l=1:timeon
con=con+Fmin(l+G(p,j-1)); % costo on en cada (i,j)
end
tstep=G(i,j)-time(2);
cdn=costcra(4,l_a,pd,time(1),time(2),tstep,T+K,incd);
mincad(p)=C(p,j-1)+con+cdn+H;
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos
elseif G(i,j)>T+K
if G(i,j)~=sinkis(j)
for p=1:i
timeon=G(i,j)-time(2)-G(p,j-1); % timeon en cada (i,j)
con=0; % costo on en (i,j) de timeini+1->timeon
for l=1:timeon
con=con+Fmin(l+G(p,j-1)); % costo on en cada (i,j)
end
ref=G(i,j)-T-K;
tstep=G(i,j)-time(2);
cdn=costcra(4,l_a,pd,time(1),time(2),tstep,T+K,incd,ref);
mincad(p)=C(p,j-1)+con+cdn+H;
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos
else % costo por estar on el resto del horizonte
```

```
for p=1:dims(j-1)
if G(p,j-1)<T+K
timeon=T+K-G(p,j-1);
con=0;
for l=1:timeon
con=con+Fmin(l+G(p,j-1));
end
mincad(p)=C(p,j-1)+con;
else
transink=0;
mincad(p)=C(p,j-1)+transink;
end
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos
end
end
end
elseif newini(j) == 0
coff=0;
for i=1:dims(j)
if G(i,j)<=T+K
for p=1:i
timeoff=G(i,j)-G(p,j-1)-time(1);
tstep=G(i,j)-time(1);
cup=costcra(2,l_a,pu,time(1),time(2),tstep,T+K,incu);
mincad(p)=C(p,j-1)+cup+coff+startup([parameter;-timeoff]);
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos
elseif G(i,j)>T+K
if G(i,j)~=sinkis(j)
for p=1:i
ref=G(i,j)-T-K;
tstep=G(i,j)-time(2);
timeoff=G(i,j)-G(p,j-1)-time(1);
```

```
cup=costcra(2,l_a,pu,time(1),time(2),tstep,T+K,incu,ref);
mincad(p)=C(p,j-1)+cup+coff+startup([parameter;-timeoff]);
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos;
else
for p=1:dims(j-1)
if G(p,j-1)<T+K
mincad(p)=C(p,j-1)+coff;
else
transink=0;
mincad(p)=C(p,j-1)+transink;
end
end
[C(i,j),pos]=min(mincad);
Prev(i,j-1)=pos;
clear mincad pos
end
end
end
end
end

minfin=C(:,n-1);
[C(1,n),pos]=min(minfin(1:dims(n-1)));
Prev(1,n-1)=pos;

%el costo del camino esta en el nodo sink
cost=C(1,n);

nodes=builthpath(Prev,G,timeini)
%nodes(1) marca tiempo inicial para poder hacer el loop path
finp=fin(nodes,[sinkis(2);sinkis(3)]);
%la posicion finp+1 hay un nodo = sink1|sink2

%construir path
lp=dp;
```

```
if finp==1;
if newini(2)==0
for i=dp+1:T+K+1
path(:,i)=[0;i-1;path(3,dp)-i+dp];
lp=lp+1;
end
elseif newini(2)==1
for i=dp+1:T+K+1
path(:,i)=[1;i-1;path(3,dp)+i-dp];
lp=lp+1;
end
end
else
for i=2:finp
if nodes(i)<=T+K
if newini(i)==1
son=nodes(i)-nodes(i-1)-time(2);
if son >0
path(:,i+dp-1)=[1;nodes(i-1)+son;path(3,dp)+son];
path(:,i+dp)=[0;nodes(i);0];
lp=lp+2;
else
path(:,i+dp-1)=[0;nodes(i);0];
lp=lp+1;
end
elseif newini(i)==0
soff=nodes(i)-nodes(i-1)-time(1);
if soff>0
path(:,i+dp-1)=[0;nodes(i-1)+soff;path(3,dp)-soff];
path(:,i+dp)=[1;nodes(i);0];
lp=lp+2;
else
path(:,i+dp-1)=[1;nodes(i);0];
lp=lp+1;
end
end
elseif nodes(i)>T+K
if newini(i)==1
```

```
son=nodes(i)-nodes(i-1)-time(2);
ref=time(2)-nodes(i)-T-K;
path(:,i+dp-1)=[1;nodes(i-1)+son;path(3,dp)+son];
path(:,i+dp)=[-1;T+K;ref];
lp=lp+2;
elseif newini(i)==0
soff=nodes(i)-nodes(i-1)-time(1);
ref=time(1)-nodes(i)-T-K;
path(:,i+dp-1)=[0;nodes(i-1)+soff;path(3,dp)-soff];
path(:,i+dp)=[2;T+K;ref];
lp=lp+2;
end
end
end

%marcando la ultima transicion esto es cuando llega a sink
%si estamos en un nodo curva llegamos al fin del horizonte
%en ese caso la transicion sink indica que el nodo last es el
%de transicion curva si no estamos en un nodo curva transicion
%es mantener el estado hasta el fin del horizonte.

if path(2,lp)<T+K
if newini(finp+1)==0
soff=nodes(finp+1)-nodes(finp)-time(1);
path(:,lp+1)=[0;T+K;path(3,lp)-soff];
elseif newini(finp+1)==1
son=nodes(finp+1)-nodes(finp)-time(2);
path(:,lp+1)=[1;T+K;path(3,lp)+son];
end
end
end
if mode==2
D.ext=path;
D.costext=cost;
[log , post]=search(nodes,T);
% busca si T esta en los nodos
if log==1
i=T-UpDn(post);
```

```
D.cost=C(i+1,post);
[last,prev,npos]=nodof(path,T);
for i=1:npos
N(:,i)=path(:,i);
end
D.path=N;
clear last prev npos
elseif log==0
if finp==1
% esto es que estuvo on/off todo el horizonte
[last,prev,npos]=nodof(path,T);
for i=1:npos
N(:,i)=path(:,i);
end
D.path=N;
clear last,prev,npos
if newini(2)==1
% restar el costo por estar on K periodos T->T+K
con=0;
for i=1:K
con=con+Fmin(T+i);
end
costf=cost-con;
D.cost=costf;
elseif newini(2)==0
D.cost=costf;
end
else
[last,prev,pos]=nodof(path,T);
[nstage, npos, k]=nodofstage(prev,last,pos,path,T);
mini=Fmin(prev(2)+1:last(2)); %puede pasar que prev(2)=0
[newcost,newlast]=getcost(dp,last,prev,pos,nstage,npos,k,
UpDn,time,T,K,C,F,pu,pd,incu,incd,l_a,mini,parameter,H);
for i=1:pos-1
N(:,i)=path(:,i);
end
N(:,pos)=newlast;
D.path=N;
```

```
D.cost=newcost;
end
end
elseif mode==1
D.p=path;
D.cost=cost;
end

D.C=C;
D.sink=sinkis;
D.node=nodes;
D.nin=newini;
D.G=G;
D.prev=Prev;
D.fin=finp;
D.ud=UpDn;

function [S]=setgraph(ini,time,T)

source=1;
sink=1;
sink1=T+time(1);
sink2=T+time(2);

if  ini(1)==1
if ini(2)>=0
dif=time(3)-ini(2);
if dif>0
timeini=dif;
else
timeini=0;
end
else
error('stage on must be positive')
end
newini=ini(1);

elseif ini(1)==0
```

```
if ini(2)<=0
dif=time(4)+ini(2);
if dif>0
timeini=dif;
else
timeini=0;
end
else
error('stage off must be negative')
end
newini=ini(1);
elseif ini(1)==-1
if ini(2)>0
if ini(2)<time(2)
timeini=time(2)-ini(2)+time(4);
else
error('curve stage must be less than tdn')
end
else
error('curve stage must be positive')
end

newini=0;

elseif ini(1)==2
if ini(2)>0
if ini(2)<time(1)
timeini=time(1)-ini(2)+time(3);
else
error('curve stage must be less than tup')
end
else
error('curve stage must be positive')
end
newini=1;
end

UpDn=[];
```

```
UpDn(1)=source;

if newini==0
tempos=[time(1);time(3);time(2);time(4)];
tempos=extension(tempos,50);
else
tempos=[time(2);time(4);time(1);time(3)];
tempos=extension(tempos,50);
end

i=1;
K=tempos(i);
j=0;

while ((timeini+K<=sink1)|(timeini+K<=sink2))
UpDn(i+1)=timeini+K;
K=K+tempos(i+1+j)+tempos(i+2+j);
i=i+1;
j=j+1;
end

if newini==0
sinkis=[sink1;sink2];
else
sinkis=[sink2;sink1];
end

n=length(UpDn);
UpDn(n+1)=sink;
sinkis=extension(sinkis,n-3);
sinkis=[source sinkis source];
dimax=sinkis(2)-UpDn(2)+1;
for i=1:n
temp =sinkis(i)-UpDn(i);
if temp>0
dims(i)=temp+1;
else
dims(i)=1;
```

```
end
end
dims(n+1)=sink;
G=zeros(dimax,n+1);

G(1,1)=source;
G(1,n+1)=sink;

for j=2:n
temp=sinkis(j)-UpDn(j);
if temp>=0
for i=0:temp
G(i+1,j)=UpDn(j)+i;
end
else
G(1,j)=sinkis(j);
UpDn(j)=sinkis(j);
end
end

S.g=G;
S.ud=UpDn;
S.sink=sinkis;
S.dim=dims;
S.time=timeini;
S.nin=newini;
```

Bibliography

[1] A. Belloni, A. Diniz, M. Maceira, and C. Sagastizábal. Bundle Relaxation and Primal Recovery in Unit Commitment Problems. The Brazilian case. *Annals of Operations Research*, (120), 2003. pp. 21-44.

[2] J. Birge, S. Takriti, and E. Long. Intelligent unified control of unit commitment and generation allocation. Technical report, Department of Industrial and Operations Engineering. The University of Chicago. pp. 4-7,34-39, `http://iems.northwestern.edu/~jrbirge/`.

[3] A. Diniz. Implementação do unit commitment das usinas térmicas no modelo dessem. Technical report, CEPEL. Centro de Pesquisas de Energia Elétrica. pp. 3-13, 29-34.

[4] A. Diniz. *Uma Estratégia de Decomposição por Relaxação Lagrangeana para Otimização da Programação Diária da Operação de Sistemas Hidrotérmicos com Modelagem Detalhada da Rede Elétrica- Aplicação ao Sistema Brasileiro*. PhD thesis, Universidade Federal do Rio de Janeiro. pp. 65-69, 112-119, 2007.

[5] A. Diniz, C. Sagastizábal, and M. Maceira. Assessment of lagrangian relaxation with variable splitting for hydrothermal scheduling. In *2007 IEEEPES General Meeting*, 2007, pp. 1-3.

[6] W. Fan, X. Guan, and Q. Zhai. A new method for Unit Commitment with Ramping Constraints. *Electric Power Systems Research*, (62):215–224, 2002.

[7] S. Feltenmark. *On Optimization of Power Production*. PhD thesis, Royal Institute of Technology, 1997. pp. 35-42.

[8] A. Frangioni and C. Gentile. Solving Nonlinear Single Unit Commitment Problems with Ramping Constraints. *Operations Research*, 54(4):767–770, 2006.

[9] L. Lasdon. Large scale nonlinear programming. In *Studies in Manangement Science and System*, pages 50–53. North-Holland Publishing Company, 1982.

[10] C. Lemaréchal, F. Pellegrino, A. Renaud, and C. Sagastizábal. Bundle methods applied to the unit-commitment problem. In J. Doležal and J. Fidler, editors, *System Modelling and Optimization*, pages 395–402. Chapman and Hall, 1996.

[11] M. Maceira, P. Terry, and L. Costa. Chain of Optimization Models for Setting the Energy Dispatch and Spot Price in the Brazilian System. *Proceedings of the Power System Computation Conference-PSCC'02*, pp. 1-4. 2002.

[12] N. Nowak. *Stochastic Lagrangian Relaxation in Power Scheduling of Hydro-Thermal System Under Uncertainty*. PhD thesis, Institut für Mathematik, Humboldt-Univ. Berlin, 1999. pp. 78-83.

[13] C. Wang and S. Shahidehpour. Effects of Ramp-Rate on Unit Commitment and Economical Dispatch. *IEEE Transactions on Power Systems*, 8, 1996. pp. 1341-1343.