ACCELERATING DUAL DYNAMIC PROGRAMMING APPLIED TO HYDROTHERMAL COORDINATION PROBLEMS

Lílian Chaves Brandão dos Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Luidi Gelabert Simonetti
André Luiz Diniz

Rio de Janeiro
Julho de 2018

ACCELERATING DUAL DYNAMIC PROGRAMMING APPLIED TO HYDROTHERMAL COORDINATION PROBLEMS

Lílian Chaves Brandão dos Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. André Luiz Diniz, D.Sc.

Prof. Laura Silvia Bahiense da Silva Leite, D.Sc.

Prof. Alexandre Street de Aguiar, D.Sc.

Dr. Roberto José Pinto, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2018

Santos, Lílian Chaves Brandão dos

Accelerating Dual Dynamic Programming applied to Hydrothermal Coordination problems/Lílian Chaves Brandão dos Santos. – Rio de Janeiro: UFRJ/COPPE, 2018.

XIII, 72 p.: il.; 29, 7cm.

Orientadores: Luidi Gelabert Simonetti

André Luiz Diniz

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2018.

Referência Bibliográfica: p. 67 – 72.

1. Dual Dynamic Programming. 2. Multistage stochastic optimization. 3. Hydrothermal planning. I. Simonetti, Luidi Gelabert *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Para Beatrice - querida,
adorada, morta.

Acknowledgment

I thank my mom for the support and understanding along those years, mainly when I was not present. My dad for the dedication, protection and for every thing he taught me as professor, father and person. André Diniz for the guidance, thrust, cooperation and the beers. The State for the large financial investment in my academic education including Universities of excellence and scholarship abroad. I also thank the CoppeTeXteam for their amazing work, friends and work colleagues for the listening and contributions.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TÉCNICAS PARA ACELERAR A PROGRAMAÇÃO DINÂMICA DUAL APLICADA A PROBLEMAS DE COORDENAÇÃO HIDROTÉRMICA

Lílian Chaves Brandão dos Santos

Julho/2018

Orientadores: Luidi Gelabert Simonetti
André Luiz Diniz

Programa: Engenharia de Sistemas e Computação

A Programação Dinâmica Dual (PDD) é uma estratégia de decomposição capaz de resolver grandes problemas de otimização estocástica multi-estágio, que tem aplicação em diversas áreas de estudo. A PDD é amplamente utilizada no planejamento hidrotérmico de sistemas de energia elétrica, principalmente em sistemas predominantemente hidroelétricos, para definir um despacho de operação de mínimo custo, considerando incertezas em algumas variáveis do problema, notadamente as afluências às usinas hidroelétricas. Quanto maior é o sistema, mais complexo é o modelo que o representa, o que torna mais caro computacionalmente resolver o problema.

Este trabalho apresenta novas estratégias para acelerar o método da PDD, que envolvem um teste de convergência local nas sub-árvores de cenários, assim como uma análise de estabilidade das variáveis de estado, para evitar operações *forward* e *backward* - intrínsecas do método de PDD - desnecessárias e economizar tempo de processamento e memória. Outra forma eficiente de redução de tempo proposta neste trabalho é um algoritmo de processamento paralelo assíncrono para a PDD, e uma variante assíncrona parcialmente paralela. Estas estratégias fazem melhor uso dos recursos disponíveis ao contornar algumas restrições de sincronismo da PDD que podem ser muito prejudiciais ao paralelismo. A eficiência das estratégias propostas é mostrada para problemas de planejamento hidrotérmico.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ACCELERATING DUAL DYNAMIC PROGRAMMING APPLIED TO HYDROTHERMAL COORDINATION PROBLEMS

Lílian Chaves Brandão dos Santos

July/2018

Advisors: Luidi Gelabert Simonetti
André Luiz Diniz

Department: Systems Engineering and Computer Science

Dual Dynamic Programming (DDP) is a decomposition strategy capable of solving high-dimension multistage stochastic optimization problems, which is applied in several fields of study. The DDP method is widely used in hydrothermal coordination planning (HTC) problems for power generation systems - mainly in predominantly hydro power systems, such as in Brazil, Norway and Chile - to define a minimum cost dispatch of power generation, taking into account some uncertainties in the system, such as the natural inflows to the reservoirs. The larger is the system, the more complex is the model, however more expensive is to solve the problem.

This work presents new strategies to accelerate DDP method, which consist in local convergence tests in scenario sub-trees, as well as analysis of the stability in the values of state variables along the nodes, to avoid unnecessary forward and backward passes and therefore saving CPU time and memory requirements. Another efficient way to reduce time proposed in this work is a novel asynchronous parallel scheme based on DDP, as well as a partial-asynchronous variant. Such strategies make a better use of the available resources by overcoming some drawbacks of traditional DDP parallel algorithms, which may be too restrictive depending on the structure of the scenario tree. The efficiency of the proposed strategies is shown for a HTC problem of the real large-scale Brazilian system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most problems involving decision under uncertainty may be modeled as a stochastic programming problem (SPP), which aim to find an optimal decision that minimizes the expected value of a given objective function (possibly, including some risk aversion criterion) under uncertainty [39], [4]. A special class of SPPs - referred to in the literature as stochastic linear programming - considers a linear objective function and linear constrains and represents uncertainty by means of a scenario tree, where the tree levels represent decision stages and branches indicate discrete values for the random variables for each node of the tree. Even with these assumptions, stochastic linear programs may be difficult to be solved, mainly due to their high dimensions.

SPP is widely used for energy planning involving uncertainty [33], [20], [18] and [15]. In particular, the mid-term hydrothermal coordination problem (HTC) in power generation systems aims to determine the weekly/monthly optimum dispatch of hydro and thermal plants. The goal is to meet an energy demand taking into account present and future operation costs, and representing many constraints of the generation and transmission systems, as well as uncertainty on some input data, such as the water inflows to the reservoirs. The HTC problem may be modeled as a multistage SPP and designed as a scenario tree, where the tree levels represent decision stages along time and branches indicate discrete values of the random variables for each node of the tree.

Figure 1.1 illustrates a simplified hydrothemal coordination problem where, in a first stage, a decision on how to generate energy to meet some demand needs to be taken. A second stage may happen under two different possible scenarios, e.g., high or low water inflows to the reservoirs, such that the consequences of the previous decision vary depending on which scenario occurs. Figure 1.2 illustrates a two-stage stochastic programming model for such problem, where the two aforementioned scenarios (high or low inflows) may happen with probabilities p and $1-p$, respectively. The decision on the second stage (consequences) depends on the decision in the first stage, which is made taking into account the probability of the future scenarios.

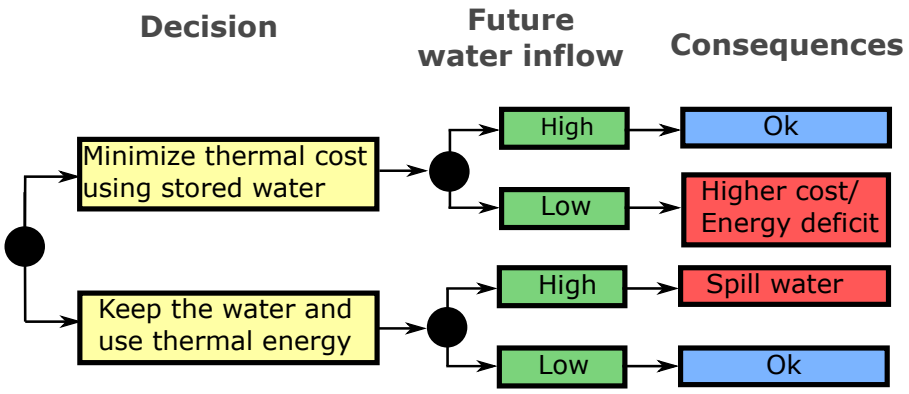


Figure 1.1: Hydrothermal coordination problem with uncertain water inflows

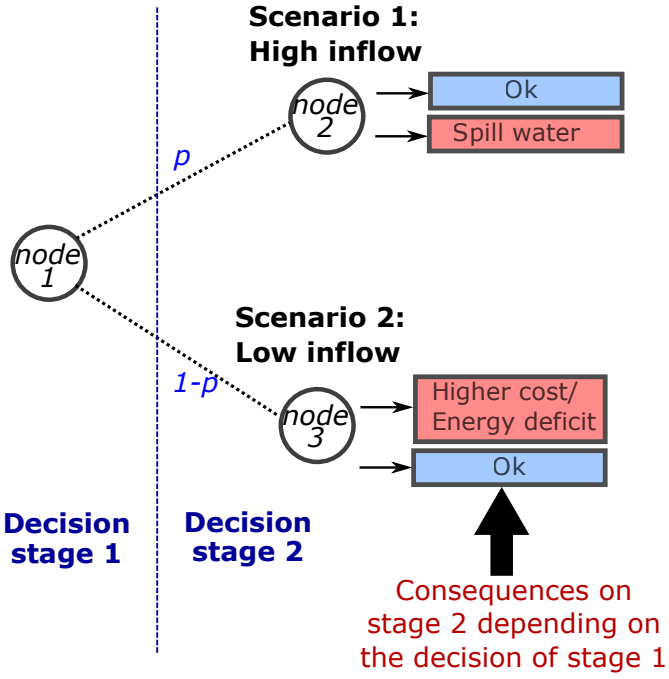


Figure 1.2: Two-stage stochastic programming model of the hydrothermal coordination problem of Figure 1.1.

The HTC problem for real systems has a very large number of stages and corresponding scenarios, leading to a scenario tree with many nodes. Moreover, the so-called "deterministic equivalent formulation" of this problem is a possibly huge linear problem (LP) because it would comprise the constraints and contribution to the objective function related to all scenarios, with their corresponding probabilities. Depending on the problem structure, the size of this LP may become prohibitive. As a consequence, decomposition algorithms have been proposed in the literature as an alternative to solve large problems where the deterministic equivalent approach proved to be inefficient or intractable.

Some of these decomposition methods are based on Lagrangian Relaxation [14],

[23], [16], where the idea is to decompose the problem into subproblems by relaxing some constraints. Depending on the problem formulation, several decomposition perspectives can be applied, such as spatial or scenario decomposition, as well as variable splitting techniques [19]. Other technique commonly applied to these problems is Benders Decomposition [2], where the set of decision variables are explicitly divided according to the stages of the problem and the so-called state variables "connect" these stages, which are solved iteratively until convergence of the global problem. Benders algorithm is the dual form of the Dantzig-Wolfe decomposition [7], where instead of adding columns in the first stage (in the latter), constraints to approximate the recourse function are added in the former method. Initially, Benders Decomposition was applied to solve two stages SPPs, leading to the well known L-shaped method [41]. Later, a nested extension of this algorithm was proposed to solve multistage SPPs, leading to the multistage Benders decomposition approach [3], also known as Dual Dynamic Programming (DDP).

The DDP approach is extensively used to solve linear or convex problems with a discrete scenario tree representation. In the traditional form of this approach, the overall problem is decomposed according to the nodes of the scenario tree. Each "level" of the tree corresponds to the decisions at a given time step and the branches represent the scenarios of each level. Considering this, each node is solved separately while the algorithm evolves along the tree and local information is transmitted back and forward until a global optimal solution is found within a desired tolerance.

However, nested methods for SPPs may require a considerable number of iterations to converge, and is always desirable to solve a stochastic problem as efficient as possible, in order to allow handling large problems without compromising the CPU time for convergence. In this sense, several techniques to improve efficiency of the original algorithms have been proposed in the literature with the purpose of reducing the number of iterations, accelerating the iterations, avoiding unnecessary operations, etc.

1.1 Objectives of the dissertation

Based on a bibliographical survey of existing variants of the DDP approach, the main goal of this work is to propose new techniques that are able to improve the efficiency of this method. The proposed schemes are grouped into two sets:

- The first set aims to increase efficiency by making a better use of the available information, thus avoiding redundant calculations. Two complementary strategies are presented: the local convergence test (LCT) and the state variables stability test (VST), which acts directly in the DDP iterations attempting

to decrease their CPU time (Chapter 5).

- The second set is based on parallel processing. We first describe the traditional DDP parallel algorithm employed in the literature and then propose two different asynchronous parallelization schemes, with the objective of breaking the time dependency of the traditional DDP algorithm and thus increasing its suitability for parallel environments (Chapter 6).

The efficiency of the proposed algorithms are assessed by solving a HTC problem in the official model DECOMP [11] used for mid-term planning and to determine spot prices and weekly dispatch of the Brazilian electrical system. The model determines the optimal dispatch taking into account several physical and operation hydro/thermal constraints and uncertainty on the natural inflows to the reservoirs.

1.2 Events and publications

The research and methods proposed in this work were presented in the conference:

- BRANDAO, LILIAN C., DINIZ, ANDRE L., "Advances in tree traversing strategies and cut sharing for multistage Benders decomposition - application to the stochastic hydrothermal coordination problem in a parallel processing environment". *XIV International Conference on Stochastic Programming (ICSP)*, Buzios - Brazil, June, 2016.

and the following conference papers have been published:

- BRANDAO, LILIAN C., DINIZ, ANDRE L., "Programação dinâmica dual: estratégias eficientes aplicadas a problemas estocásticos de coordenação hidrotérmica". *XXIV Seminário Nacional de Produção e Transmissão de Energia Elétrica (SNPTEE)*, Curitiba - Brazil, October, 2017.

- BRANDAO, LILIAN C., DINIZ, ANDRE L., SIMONETTI, LUIDI G., "Accelerating Dual Dynamic Programming for Stochastic Hydrothermal Coordination Problems". *XX Power Systems Computation Conference (PSCC)*, Dublin - Ireland, June, 2018.

Finally this work was also submitted for publication in the following journal:

- BRANDAO, LILIAN C., DINIZ, ANDRE L., SIMONETTI, LUIDI G., BORGES, CARMEN L., "Asynchronous Dual Dynamic Programming". *European Journal of Operational Research (EJOR)*, February, 2018.

1.3 Dissertation arrangement

The dissertation is organized into 7 chapters, as follows:

- Chapter 2: the state of the art on DDP improving methods is reviewed, and the main techniques proposed in order to increase time efficiency on DDP are described.
- Chapter 3: mid-term HTC problem - which will be used in the dissertation experiments - is modeled in detail as a SSP.
- Chapter 4: the stochastic programming formulation is introduced, and then the DDP method - as traditionally presented in the literature - is described.
- Chapter 5: the two proposed improved strategies for DDP are presented.
- Chapter 6: some parallel processing strategies applied within the DDP approach are presented: first, the traditional parallel approach, followed by the two proposed strategies proposed in this work: an asynchronous and a partial asynchronous approach.
- Chapter 7: the numerical experiments are presented. The proposed strategies are applied to a HTC problem and their performance are evaluated as compared to existing methods on the literature.
- Chapter 8: the main conclusions, results and analysis are summarized.

Chapter 2

Literature Survey: improvements in the DDP Algorithm

Dual Dynamic Programming (DDP) is a decomposition method that is very useful for solving large SPPs, since it is able to split the overall problem, solving it iteratively until a global optimal solution is found. However, as an iterative strategy it may need a large number of iterations to converge and, in addition, each iteration involves solving a large number of subproblems. Some strategies were proposed in the literature attempting to reduce the running time of DDP in two senses: by increasing the convergence rate of the method (i.e., performing a lower number of iterations) and/or by decreasing the time required per iteration (i.e., iterating faster). In this chapter some of the main strategies found in the literature are reviewed.

Multicuts

The classical version of L-shaped method - applied to two-stage problems - computes one linear approximation (cutting plane) per iteration for the expected recourse (future cost) function of the first stage, which is calculated as the average contribution of the second stage nodes. The work [5] proposed an alternative model that considers a linear approximation for the contribution of each scenario separately, thus not aggregating the cuts from the second stage nodes. The idea is to make use of more information in order to increase the convergence rate and thus fewer iteration are needed. Such strategy may lead to smaller CPU times, although it also implies in having more variables and constraints for the first stage, making this subproblem more expensive to be solved. Later, the work [17] extended the multicut version of the L-shaped method for the multistage setting proposed in [3]. A comparative analysis of the two methods [4] concluded that the multicut algorithm is more efficient than the singlecut version when the number of scenarios and the dimension of the vector of decision variables have a similar magnitude. In addition, comparative results presented in [13] for a stochastic HTC problem with 8 time steps and 255 nodes showed a slight superiority of the multicut version.

Cut aggregation

The main drawback of multicut strategies is the large number of constraints included per iteration in the subproblem of each node, due to the addition of one cut for each of its descendant scenarios. In order to reduce this effect, strategies have been developed on the aim of finding an adequate trade-off between the convergence rate and the size of the LPs to be solved for each subproblem. In [42] a general strategy was proposed to aggregate the recourse functions of some descendant nodes, thus allowing an intermediate scheme between the single and multicut strategies, for two-stage problems. A dynamic adjustment of the aggregation level was proposed, where the most aggregated level is the singlecut version and the least is the multicut. Although the choice of the aggregation level is not known *a priori*, the proposed adaptive technique proved to be able to reduce the CPU time as compared to both standard multi and singlecut methods.

More recently, a cut aggregation scheme for a multistage problem was proposed in [46], where the number of cuts is reduced by summing the constraints, even though the number of cost variables is kept. The definition of the DDP iteration in which the cuts are aggregated is based on two measures: (a) the number of consecutive iterations a cut was inactive and (b) a given number of cuts removed per iteration.

Cut selection

Another way of handling the overload caused by large amounts of cuts is selecting some interesting cuts and discarding the others. One of the simplest cut selection strategies is to discard the dominated cuts that will never be active during the LP solution, and thus can be removed without loss of information. Even though this strategy seems to be simple and efficient, it is computationally expensive to determine dominated cuts, specially in problems with a large number of state variables in the recourse function. In [38] an algorithm to remove dominated cuts is presented, where the extra computation cost is offset by proceeding the cut removal algorithm only after a certain number of iterations. In [8] the concept of *level of dominance* is defined, where a cut is considered to be dominated if it is dominated into a set of points of the state variables domain. By applying this technique for a long term hydrothermal planning problem a reduction of up to 10 times in the computational time was achieved.

The work [27] proposed a so called "cut strengthening" method to accelerate Benders convergence for Mixed Integer Problems (MIP). They explore the non-uniqueness of second stage solutions and try to find the best cut among the so called set of "Pareto-optimal" cuts. The idea is to look for nondominated cuts into the set of optimal dual solutions of the second stage problem. Even though an

extra cost is added to perform this procedure, the algorithm may be able to find better results in a faster way and reach convergence more efficiently. Such set of Pareto-optimal cuts was studied in [28] for a problem of aircraft routing, however the reduction in the number of Benders iterations was not compensated by the time spent in those extra calculations. Some strategies were later proposed in order to decrease the cost of finding the nondominated cuts. Approximations, alternatives, adaptations and improvements of the method can be found in [36], [32], [40] and [31].

Instead of searching the bests cuts, the work [47] seeks inexact cuts yielded by non-optimal feasible dual solutions. An early termination of the solving procedure for the second stage subproblem produces a valid cut, and saves the CPU time of finding the optimal solution, specially if the second stage is a large subproblem. The authors have proved convergence in problems with complete recourse.

Bunching

The subproblems of each node in a given time step (level) of the scenario tree are usually very similar. Therefore, the LPs related to these nodes may share the same optimal basis, especially in the particular case where they differ from each other only by the right hand side of some constraints. Based on that, the bunching technique proposes to use the final Simplex basis of a given subproblem as a starting basis to another one, in order to minimize the number of Simplex pivots in the solution of siblings nodes of the tree [43], [44], [17].

Sequence protocols

The sequence protocol, also called tree-traversing strategy, defines how the algorithm will visit the nodes of the scenario tree. Studies have shown that the sequence protocol may have an important impact on the algorithm convergence [17]. Some of the classical protocols are:

- *Fast-forward-fast-backward*: this strategy - proposed by [45] - consists in moving forward through the tree, solving all subproblems from stages 1 to T (where T is the number of stages of the tree) and transferring the values of state variables. Then it goes backward from stages T to 1 passing cost information through the Benders cuts. Therefore, information (states and cuts) are quickly spread through the tree, even if at earlier iterations this information does not correspond to points in the neighborhood of the (yet unknown) optimal values of state variables.

- *Backwards* approach: the idea of this strategy is to always move backward on the tree, except in two situations: when the root node is reached or when no

new information is produced on the backward pass with relation to the previous iterations. The main concern about this approach is that the method may demand too much effort to construct cuts to the earlier stages that represent the recourse function in distant regions to the optimal solution.

- *Forward* approach: in contrast to the previous approach, this strategy always move forward on the tree except in two cases: when a leaf node is reached or when no new information is produced on the forward pass with relation to the previous iterations, [3]. In this case, the main concern is to spend a lot of time trying to reach convergence in the final stages for bad values for the state variables of previous stages.

Alternatively, additional strategies have been proposed in the literature. The work [30] presented a protocol named "$\varepsilon-$strategy", where the idea is to reverse the direction of the algorithm if the local convergence gap on the current stage is smaller than a parameter ε. In [1] a bouncing technique is presented, where the authors first define a "block-stage" $t < T$ and two kinds of iterations: a minor iteration, which goes forward and backward along stages 1 to t; and a major iteration, which consists in a complete forward from stage 1 to T, then a backward pass from stage T to t, followed by another forward pass from stage t to T, and finally a complete backward pass from T to 1. The algorithm proceeds by alternating between major iterations (that traverse the whole tree) and minor iterations, trying to avoid final stages, which may have a large number of scenarios.

More recently, the work [46] proposed a sequence protocol that uses the bouncing method [1] with a dynamic way to choose the block-stage, combined with $\varepsilon-$backward strategies ([30]), but applying a dynamic parameter by computing a discrepancy measure that depends on the size of the convergence gap.

Node aggregation

Node aggregation techniques aim to increase the performance of a decomposition method by grouping the nodes of the scenario tree. Nodes that belong to the same group are solved together as deterministic equivalent subproblems, and the groups are taken into account in the cutting plane decomposition scheme as if they were single node subproblems.

Different node aggregation models are proposed and studied in [4], [22], [9], [12], labeled as: complete node decomposition, scenario decomposition, subtree decomposition and complete scenario decomposition, respectively. These methods are very sensitive to the size and shape of the scenario tree, which makes it difficult to evaluate and compare these strategies for general problems.

As the nodes are aggregated in a same group, the corresponding subproblems

become more complex and require more memory and CPU time to be solved. However, the convergence rate tends to increase, since the number of subproblems that need to communicate in order to reach an optimal solution is decreased. On the other hand, when nodes are disaggregated, the subproblems are simpler (and faster to be solved) but the granularity is higher, which leads to a slower convergence rate.

Parallelization

Parallelization is a very powerful and efficient acceleration strategy that can drastically reduce the real time of an algorithm and also allows the introduction of new techniques to increase algorithm efficiency. However, the parallel capability depends not only on the available resources, but also on the suitability of the algorithm for parallelization. As a consequence, the parallelization strategy and efficiency of an algorithm is strongly related to its structure, since dependencies along the algorithmic flows create bottlenecks in the parallel process and may limit the efficiency and scalability of a parallelization scheme, thus reducing its performance.

Parallel computing has been studied and applied in many areas, including stochastic programming, many approaches were proposed and studied in an attempt to parallelize DDP methods. However, the tree structure of multistage stochastic problems causes a high dependency among the different scenario levels: in the traditional DDP tree traversing strategy, a given node of the tree must wait new values of state variables from its ascendant node in the forward passes, as well as new Benders cuts from its descendant nodes in the backward passes.

Even though nodes in the same level are independent and parallelization can be exploited among them, the inter-level dependencies restrain some classical parallel algorithms that have been proposed for DDP problems, as for example [6], that solves independent sub-trees in parallel, and [9], which explores the inter-level independence by solving the same stages in parallel. In both works, several stochastic instances of stochastic programs were considered, and their results showed that speedup and efficiency of the parallelization schemes are very sensitive to the size and shape of the scenario tree. In addition, these authors experienced a considerable drop in running time with a small number of processors, but premature speedup saturation as more processors were employed.

Some smarter techniques attempt to break the time dependency of DDP algorithms. In [24], the authors proposed a partial asynchronism to solve two-stage stochastic programs (asynchronous L-Shaped method), where the first stage subproblem is solved again without waiting all second stage nodes to be solved, thus avoiding idle processors due to unbalance. They showed that the proposed scheme does not harm the method convergence and can be very suitable for heterogeneous parallel environments.

An asynchronous DDP approach was also proposed by [29] for a multistage stochastic program, where the subproblem of a given node is computed whenever a new entry (state or cut) is available. Even though this also reduces idleness along the processors, the convergence criterion of the algorithm is somehow affected and their results are not very conclusive. More recently, [35] proposed an asynchronous version for Deterministic Dual Dynamic Programming (DDDP), where deterministic problems (i.e., that have at most one descendant node for each node) are solved. In those cases, the inter-level dependency of the original DDP algorithm does not allow any level of parallelization. In order to overcome this issue, the authors proposed an asynchronous solving procedure where all nodes are solved in parallel and then information (state variables and Benders cuts) are exchanged. The results showed a significant time reduction as compared to the sequential algorithm for trees of any size, even though the efficiency and speedup of the algorithm were not satisfactory.

Chapter 3

Hydrothermal Coordination Problem

The HTC problem has an important role in operation and expansion planning studies of power generation systems. As the model used to represent the system becomes more detailed and the representation of uncertainties/length of the time horizon are improved, the problem becomes closer to reality but more difficult to solve. This work considers a mid-term HTC problem modeled by a linear objective function and linear or piecewise linear constraints. Some of the main aspects of the model are as follows:

- individual representation of thermal plants, as well as hydro plants in cascade, arranged in several river basins;

- a load duration curve with three load blocks for each weekly/monthly time step. The use of load blocks allows to discretize the demand representation to express the load variability within a stage;

- several physical and operating constraints for hydro and thermal plants;

- a finite number of water income scenarios to the hydro plants in each time step, leading to a multistage scenario tree. The inflow scenarios are synthetically generated by a periodic auto-regressive model (GEVAZP - [21]) where the parameters are calculated based on the inflow historical record. Therefore, the scenario tree has an implicit time correlation, since the past inflows are used by the auto-regressive model in order to generate the inflow scenarios for subsequent stages;

- a multivariate future cost function (FCF) attached at the end of the horizon, to couple mid-term and long term decisions, as presented by [25].

Each node (t, s) of the scenario tree is related to a time period t and an inflow scenario s, with a corresponding linear programming subproblem (LP). The input parameters of this subproblem are the so called "state variables", which are the storages in the reservoirs at the beginning of the scenario. The mathematical formulation of the objective function and constraints of the LP of a given node (t, s) is presented below.

3.1 Objective function

The objective function of each individual LP is composed by the sum of thermal generation costs (present cost) and an estimation of future costs:

$$\min \sum_{i=1}^{NT} \sum_{p=1}^{NL} ct_i gt_i^{p,t,s} + \mathbb{E}[Q(t, s)], \tag{3.1}$$

where NT is the number of thermal plants and NL is the number of load blocks; ct_i is the incremental cost of thermal plant i and $gt_i^{p,t,s}$ is the generation of thermal plant i at load block p, time step t and scenario s. The term $\mathbb{E}[Q(t, s)]$ represents the expected value of future costs, which is given either by the FCF (in the leaf nodes) or by the Benders cuts generated by the DDP strategy (for the remaining nodes). In order to take into account risk aversion, a conditional value-at-risk measure can be considered in the problem formulation, as for example by [37], [34] and [26].

Artificial costs are also included in the objective function, by means of slack variables for constraint violation (with high penalty costs) and very low costs for spillage, in order to avoid unnecessary waste of water.

3.2 Future cost function

For the leaf nodes of time step T, a FCF function provides an expected value of future system operating costs, which depends on the energy stored in the reservoirs at the end of the time horizon. The expression for the future cost is the same for all scenarios and is a function of state variables of a long term planning problem NEWAVE ([25]). Since the hydro plants are not considered individually in such long term problem, the state variables of the FCF are the total energy storage in the equivalent reservoirs (EERs) represented in the NEWAVE model. The cuts of the FCF are modeled as constraints and a variable $FCF^{T,s}$ represents its cost in the objective function:

$$FCF^{T,s} \geq \pi_{0,T}^{(k)} + \sum_{j=1}^{NEER} \pi_{j,T}^{(k)} E_{eer_j}^{T,s}, \qquad k = 1, ..., NCUT_{FCF}; \tag{3.2}$$

$$E_{eer_j}^{T,s} = \sum_{i \in \Omega_j} \rho_i v_i^{T,s}, \tag{3.3}$$

where $NEER$ is the number of equivalent reservoirs and $NCUT_{FCF}$ is the number of FCF cuts provided by the long term model, with corresponding coefficients π. An auxiliary variable $E_{eer_j}^{T,s}$ is used to convert water storage in the individual plants to aggregated energy storage for EER j, where ρ_i is the corresponding conversion coefficient for each plant i in the set Ω_j of hydro plants belonging to EER j.

3.3 Load supply

The load supply constraint for each load block is given by:

$$\sum_{i=1}^{NT} gt_i^{p,t,s} + \sum_{j=1}^{NH} gh_j^{p,t,s} = d^{p,t}, \quad p = 1, ..., NL, \tag{3.4}$$

where NH is the number of hydro plants of system, $gh_j^{p,t,s}$ is the generation of hydro plant j at load block p, time step t and scenario s, and $d^{p,t}$ is the power demand at load block p and time step t.

3.4 Water balance

There is a time-coupling water balance equation for each hydro plant i:

$$v_i^{t,s} + \sum_{p=1}^{NL} (q_i^{p,t,s} + s_i^{p,t,s}) - \sum_{p=1}^{NL} \sum_{j \in \Omega_i^{up}} (q_j^{p,t,s} + s_j^{p,t,s}) = v_i^{t-1,r} + I_i^{t,s} \quad i = 1, ..., NH, \tag{3.5}$$

where $I_i^{t,s}$ is the water inflow and $q_i^{p,t,s}$, $s_i^{p,t,s}$ are the turbined and spillage outflows for hydro plant i; Ω_i^{up} is the set of upstream hydro plants to plant i, and $v_i^{t,s}$ is the final storage level of the reservoir i for the node (t, s). The initial state for the reservoir in this node is the final storage $v_i^{t-1,r}$ of its ascendant node $(t-1, r)$.

3.5 Maximum turbined outflow

A maximum turbined outflow constraint for each hydro plant, time step, scenario and load block is imposed:

$$q_i^{p,t,s} \leq \overline{q_i^{p,t}} \quad \forall i = 1, ..., NH, p = 1, ..., NL, \tag{3.6}$$

where $\overline{q_i^{p,t}}$ is the maximum turbined outflow, calculated by a pre-processing algorithm that considers the limits of the generator and turbine, according to the initial storage levels of the reservoirs. Since the turbined outflow depends on the storage levels, the maximum limit would vary along the DDP iterations, depending on the storage levels in previous nodes, which are decision variables. In order to avoid that, which would violate the convexity principles of DDP, the initial values of storage are used to obtain approximated values for such limits.

3.6 Hydro Production Function

The generation of a hydro plant is a function of the turbined outflow, the net head (which is a function of storage), the spillage and the turbine/generator efficiency factors, which also depend on the net water head and discharge. Since this relation is known to be non-convex, we use a so-called Approximate Hydro Production Function (AHPF), which estimates the power generation as a concave piecewise linear function of the storage in the reservoir, the turbined outflow and the spillage ([10]). This piecewise function is computed a priori and added to the model as a set of time-coupling inequalities for each load block, hydro plant and time step:

$$\begin{aligned} gh_i^{p,t,s} \leq \gamma_{0,i,t,s}^{(k)} + \gamma_{v,i,t,s}^{(k)}(v_i^{t-1,r} + v_i^{t,s})/2 \\ +\gamma_{q,i,t,s}^{(k)} q_i^{p,t,s} + \gamma_{s,i,t,s}^{(k)} s_i^{p,t,s} \\ k = 1, ..., NCUT_{FPHA}^{i,t}; i = 1, ..., NH; p = 1, ..., NL, \end{aligned} \tag{3.7}$$

where γ are the cut coefficients related to each variable (indicated in the first lower index) of the AHPF.

Finally, there are additional constraints related to storage/generations limits for all hydro plants and power capacity for the thermal plants.

Chapter 4

Dual Dynamic Programming

Dual Dynamic Programming (DDP) is a decomposition method capable of solving multistage stochastic optimization problems. The algorithm builds iteratively a set of linear approximations of the recourse (or future cost function) at each node of the scenario tree. This function will tend to converge to the exact future cost function, specially in the neighborhood of the optimal solution, after a fair number of iterations. This chapter introduces the stochastic programming formulation and notation used in this work, as well as the DDP algorithm.

4.1 Stochastic programming formulation

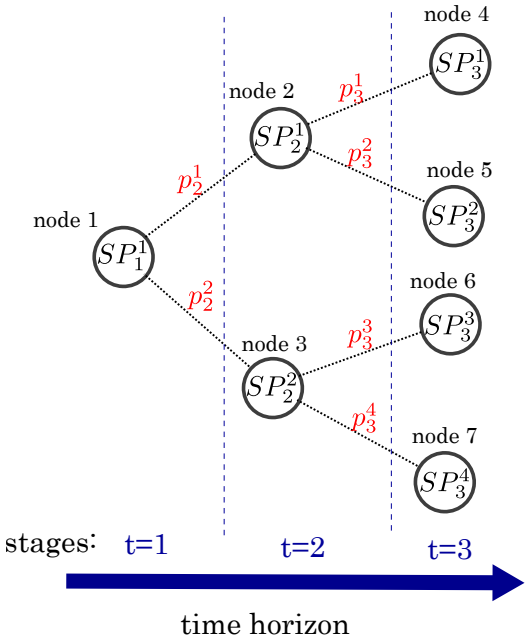
A multistage optimization problem with a discrete scenario representation is a rather general type of a stochastic programming problem. We consider a problem with stagtes (time steps) $\mathcal{T} = \{1, ..., T\}$, where each stage $t \in \mathcal{T}$ is composed by a set of nodes (or subproblems) denoted by SP_t^s, where $s \in \mathbb{Z}$ is a scenario of stage t. By defining Ω_{t-1}^u as the set of descendant scenarios of the subproblem SP_{t-1}^u, then $s \in \Omega_{t-1}^u$ if and only if the SP_t^s is descendant of the subproblem SP_{t-1}^u. We denote $p_t^s \in \mathbb{R}_+$ the conditional probability of scenario s, given that scenario u has occurred, and $s \in \Omega_{t-1}^u$. By probability laws we have that:

$$\sum_{s \in \Omega_{t-1}^u} p_t^s = 1, \quad \forall t \in \{2, ..., T\}$$

The total probability $p_{total_t}^s$ of a node is defined by the probability that a specific scenario s in time period t will happen, and is the product of the conditioned probabilities of the branches along the path from the root node to this node:

$$p_{total_t}^s = p_t^s * p_{total_{t-1}}^u, s \in \Omega_{t-1}^u, \forall t \in \{3, ..., T\}$$

$$p_{total_2}^j = p_2^j, j \in \Omega_1$$



$$\Omega_1 = \{1, 2\}$$
$$\Omega_2^1 = \{1, 2\}$$
$$\Omega_2^2 = \{3, 4\}$$
$$\Omega_3^1 = \Omega_3^2 = \Omega_3^3 = \Omega_3^4 = \emptyset$$

Figure 4.1: Three stages scenario tree example

Figure 4.1 illustrates an example of a scenario tree with the notation for probabilities, subproblems and set of scenarios described previously. The tree contains three stages ($T = 3$), and each node is associated with two possible descendant scenarios ($|\Omega_t^s| = 2, \forall t < 3$), with a total of seven nodes.

The dynamic programming formulation of the nested decomposition applied to the overall problem is as follows:

Stage $t = 1$:

$$z = \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s \in \Omega_1} [Q_2^s(x_1)] \tag{4.1}$$

Stage $t \in [2, T-1]$:

$$Q_t^u(x_{t-1}^j) = \min_{x_t^u \in \mathcal{X}_t^u(x_{t-1}^j)} f_t(x_t^u) + \mathop{\mathbb{E}}_{s \in \Omega_t^u} [Q_{t+1}^s(x_t^u)], \forall u \in \Omega_{t-1}^j.$$

Stage $t = T$:

$$Q_T^u(x_{T-1}^j) = \min_{x_T^u \in \mathcal{X}_T^u(x_{T-1}^j)} f_T(x_T^u), \forall u \in \Omega_{T-1}^j,$$

where:

$x_t^s \in \mathbb{R}^n$:	decision variables from subproblem SP_t^s (stage t and scenario s);
$\mathcal{X}_t^s \subset \mathbb{R}^n$:	feasible set for x_t^s, which depends on the state variable x_{t-1}^j coming from ascendant the scenario;
$f_t : \mathbb{R}^n \to \mathbb{R}$:	present cost function of stage t;
$Q_{t+1}^s : \mathbb{R}^n \to \mathbb{R}$:	is the cost-to-go function of a future stage $t+1$ and scenario s;

We note that, x_1, $\mathcal{X}_1$ and Ω_1 are not indexed by scenario, since stage $t = 1$ has only one subproblem. The cost-to-go function Q_{t+1}^s represents the future cost incurred by a decision x_t^u on a stage t given the occurrence of a scenario s in time step $t+1$. Therefore, the goal is to make a decision at each stage u that minimizes the present cost $f_t(x_t^u)$ plus the expected value of the cost-to-go function given by the set of possible scenarios. Such expected cost function is the "recourse function" $\mathcal{Q}_{t+1}^u$ of the scenario, given by:

$$\mathcal{Q}_{t+1}^u(x_t^u) = \mathop{\mathbb{E}}_{s \in \Omega_t^u}[Q_{t+1}^s(x_t^u)].$$

In order to solve problem 4.1 and obtain the minimal cost z, some strategy must be adopted. The most intuitive approach is to aggregate the problem into a single huge minimization problem that comprises the constraints and contributions of all scenarios - with their corresponding probabilities - to the objective function. This so called "deterministic equivalent problem" is formulated as follows:

$$\min f_1(x_1) + \sum_{u \in \Omega_1} p_2^u [f_2(x_2^u) + \sum_{s \in \Omega_2^u} p_3^s [f_3(x_3^s) + ...[f_{T-1}(x_{T-1}^q) + \sum_{j \in \Omega_{T-1}^q} p_T^j f_T(x_T^j)]...]] \tag{4.2}$$

$s.t.$

$$x_1 \in \mathcal{X}_1(x_0)$$

$$x_t^s \in \mathcal{X}_t^s(x_{t-1}^j), \forall s \in \bigcup_j \Omega_{t-1}^j, \forall t \in [2, ..., T],$$

where the set of decision variables comprises all stages and scenarios. If we consider linear cost functions and polyhedral feasible sets, the optimization problem as presented in 4.2 is a linear program that can be solved with standard linear programming methods such as Simplex algorithm. However, due to the stochastic nature of the problem and the exponential growth in the number of scenarios of the tree, it rapidly becomes prohibitive to solve such problem.. In this sense, decomposition algorithms have been proposed in the literature, one well known strategy that explores the formulation 4.1 is Nested Benders Decomposition, also known as Dual

Dynamic Programming (DDP). This strategy - used in this work and described in the next section - uses a natural decomposition in scenarios and applies an iterative solving procedure that uses a cutting plane strategy.

4.2 Dual Dynamic Programming

Dual Dynamic Programming or Nested Benders Decomposition is a cutting plane strategy that solves multistage problem 4.1 in an iterative way. The idea is to obtain one feasible solution for the subproblem of each node per iteration and build a linear approximation of the corresponding cost-to-go functions at the states (state variables values) related to these solutions. As the iterative process evolves and several points are visited, piecewise linear models of the cost-to-go functions are progressively constructed. The model $M^k_{t+1,s}$ of the cost-to-go function for scenario s of stage $t+1$ in iteration k can be represented as follows.

$$M^k_{t+1,s}(x^u_t) = \max\{\alpha^i_{t+1,s} + \beta^i_{t+1,s}x^u_t, \quad i = 1, ..., k\},$$

where coefficients α and β are obtained by a first order Taylor approximation of the cost-to-go function around a feasible point $x^{k,u}_t$ at iteration k:

$$\alpha^k_{t+1,s} + \beta^k_{t+1,s}x = M^{k-1}_{t+1}(x^{k,u}_t) + \frac{\partial M^{k-1}_{t+1}(x^{k,u}_t)}{\partial x^{k,u}_t}(x - x^{k,u}_t).$$

As the model accumulates the linear approximations computed in all previous iterations, the following property holds:

$$M^{k-1}_{t+1,s}(x^u_t) \leq M^k_{t+1,s}(x^u_t) \quad \forall k \in \mathbb{Z}, \forall x^u_t \in \mathbb{R}. \tag{4.3}$$

It is shown that if the subproblem SP^s_t is convex, the cost-to-go function Q^s_t is also convex [4]. Due to this convexity property, any linear approximation built as a first order Taylor expansion of the cost-to-go function is a lower estimation of this function along its domain, i.e:

$$M^k_{t+1,s}(x^u_t) \leq Q^s_{t+1}(x^u_t), \quad s \in \Omega^u_t, \ \forall x^u_t \in \mathbb{R}^n. \tag{4.4}$$

The dynamic programming formulation of the DDP method then becomes:
Stage $t = 1$:

$$\underline{z^k} = \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s \in \Omega_1}[M^k_{2,s}(x_1)]$$

Stage $t \in [2, T-1]$:

$$M^k_{t,u}(x^j_{t-1}) = \min_{x^u_t \in \mathcal{X}^u_t(x^j_{t-1})} f_t(x^u_t) + \mathop{\mathbb{E}}_{s \in \Omega^u_t}[M^k_{t+1,s}(x^u_t)], \forall u \in \Omega^j_{t-1}.$$

Stage $t = T$:

$$M^k_{T,u}(x^j_{T-1}) = \min_{x^u_T \in \mathcal{X}^u_T(x_{T-1})} f_T(x^u_T), \forall u \in \Omega^j_{T-1}$$

We note that for the last stage ($t = T$) the model matches the real function in those points around which the function was approximated. An iteration of the traditional tree traversing protocol of DDP method comprises two phases:

- *Forward pass*: consists in solving the nodes of the tree from stage 1 to T, nesting the decision variables of the ascendants nodes with the state variables for the descendants nodes. In other words, the solution x^j_t of subproblem SP^j_t is used as input in the subproblems $SP^s_{t+1}, \forall s \in \Omega^j_t$, for all subproblems of the scenario tree, as presented in Figure 4.2.

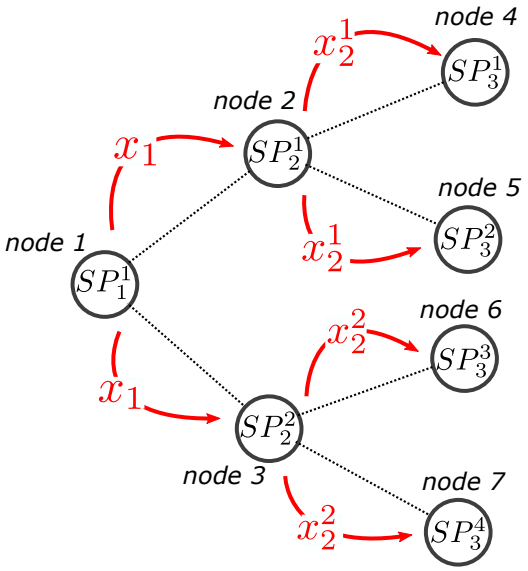


Figure 4.2: Forward pass representation

- *Backward pass*: consists in solving the nodes of the tree from stage T to 1, where the solution of nodes on stage T are the ones obtained in the previous forward pass (i.e., nodes of the last stage are solved once per iteration), and the solution of the root node ($t = 1$) is the used for next forward pass. The solution of each subproblem SP^s_t in the backward pass is used to build a Benders cut for the model $M^k_{t,s}$, as presented in Figure 4.3.

During the course of the DDP algorithm, several forward and backward passes are consecutively performed. As a consequence, several values of states at each node

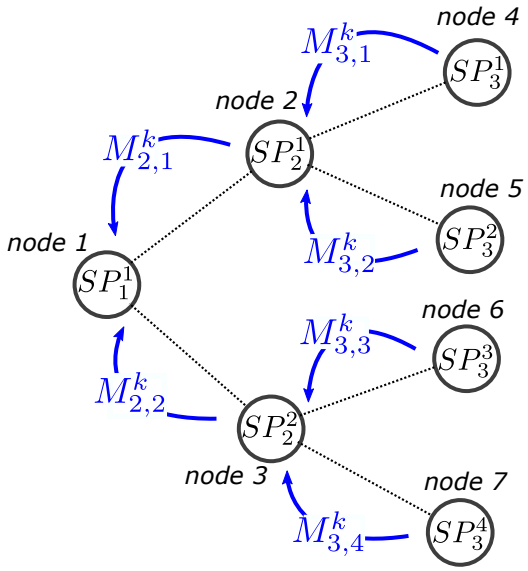


Figure 4.3: Backward pass representation

of the scenario tree are visited and the model of the future cost in each node is built and updated. After a fair number of iterations, the model may have a reasonable number of cuts in order to be a good approximation of the real future cost function and then archive a optimum solution within the desired tolerance. The next section details the convergence criterion of the DDP strategy.

4.3 Convergence of Dual Dynamic Programming

A global optimal solution for the multistage stochastic programming problem is one that yields the minimal cost and is feasible for all nodes of the scenario tree. Considering the dynamic formulation, an optimum solution for the first stage problem is given by:

$$x_1^* \in \mathcal{X}_1(x_0) \text{ is an optimum solution} \iff$$

$$f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1^*)] \leq f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1)], \forall x_1 \in \mathcal{X}_1(x_0).$$

Therefore, the minimum cost solution is given by:

$$z^* = f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1^*)].$$

The stopping criterion of the DDP solving strategy is based on the proximity of lower and upper bounds for the value z^* of the optimum solution, as explained below.

Lower bound calculation:

The DDP lower bound $\underline{z^k}$, at iteration k, is given by the optimal value (sum of

present and future costs) of the solution of the first node in this iteration:

$$\underline{z^k} = \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathbb{E}[M^k_{2,s}(x_1)],$$

where $\underline{z^k}$ has the following properties:

- by definition, it is always the minimum (optimal) value of the subproblem related to the first node;
- the future cost function, at any iteration k, is a lower approximation of the real future cost function, due to the convexity property (equation 4.4).

Because of these two properties we conclude that $\underline{z^k}$ is always a lower bound for z^*.

Proof. Suppose $\exists x_1^* \in \mathcal{X}_1(x_0)$ such that:

$$z^* = f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1^*)] = \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1)].$$

From 4.4 we also have:

$$M^k_{2,s}(x_1) \leq Q_2^s(x_1), \forall x_1 \in \mathcal{X}_1(x_0), \forall k \in \mathbb{Z}.$$

So, considering a fixed point x_1^* and the expectation of the second stage costs, the following inequality also holds:

$$f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[M^k_{2,s}(x_1^*)] \leq f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1^*)], \forall k \in \mathbb{Z}.$$

Considering the minimum of the functions we have:

$$\begin{aligned}\min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[M^k_{2,s}(x_1)] &\leq f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[M^k_{2,s}(x_1^*)] \\ &\leq f_1(x_1^*) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1^*)] = \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1)], \forall k \in \mathbb{Z}\end{aligned}$$

and:

$$\min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[M^k_{2,s}(x_1)] \leq \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s\in\Omega_1}[Q_2^s(x_1)], \forall k \in \mathbb{Z},$$

which leads finally to:

$$\underline{z^k} \leq z^*, \forall k \in \mathbb{Z}.$$

□

Additionally, as the first node subproblem differs between iterations only by an

extra cut (constraint), we have that:

$$M_{2,s}^{k}(x_1) \leq M_{2,s}^{k+1}(x_1), \forall x_1 \in \mathcal{X}_1(x_0), \forall k \in \mathbb{Z},$$

with a similar proof we can conclude that:

$$\min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s \in \Omega_1}[M_{2,s}^{k}(x_1)] \leq \min_{x_1 \in \mathcal{X}_1(x_0)} f_1(x_1) + \mathop{\mathbb{E}}_{s \in \Omega_1}[M_{2,s}^{k+1}(x_1)], \forall k \in \mathbb{Z}$$

and, finally:

$$\underline{z^k} \leq \underline{z^{k+1}}, \forall k \in \mathbb{Z}.$$

Considering this, the lower bound is a monotonically increasing function along the iterations.

Upper bound calculation:

The upper bound is calculated based on the total cost of a global solution obtained at the end of a forward pass, taking into account all nodes and their corresponding probabilities. Such solution is feasible, since it came from a nested forward pass, i.e., the initial condition of a given node is the final condition of its ascendant node; thus satisfying the time linking constraints of the problem. The cost z^k of the forward solution $x^k = \{x_1^k, x_2^{s,k}, x_3^{u,k}, ..., x_T^{j,k}\}, \forall s \in \Omega_1, u \in \Omega_2^s, ..., j \in \Omega_{T-1}^q$, at iteration $k \in \mathbb{Z}$ is given by:

$$z^k = f_1(x_1^k) + \sum_{s \in \Omega_1} p_2^s [f_2(x_2^{s,k}) + \sum_{u \in \Omega_2^s} p_3^u [f_3(x_3^{u,k}) + ... [f_{T-1}(x_{T-1}^{q,k}) + \sum_{j \in \Omega_{T-1}^q} p_T^j f_T(x_T^{j,k})]...]].$$

Since any feasible solution in a minimization problem is an upper bound of its optimal value, we have the following property:

$$z^k \geq z^*, \forall k \in \mathbb{Z}.$$

However, there is no guarantee that the cost of a feasible solution will be monotonically decreasing along the iterations, because depending on the decision made on earlier stages, the sum of costs in all nodes may be higher than a solution computed in a previous iteration. In this sense, the upper bound $\overline{z^k}$ is given by:

$$\overline{z^k} = \min(\overline{z^{k-1}}, z^k)$$

Convergence test:

The optimality gap gap^k at a given iteration k is defined as the relative difference of the bounds:

$$gap^k = \frac{\overline{z^k} - \underline{z^k}}{\overline{z^k}},$$

and the convergence test, which determines the stopping criterion is:

$$gap^k < \epsilon,$$

where $\epsilon \in \mathbb{R}$ is the accepted tolerance for the value of the optimal solution. It has been shown in the literature that DDP method finitely converges to the global optimum solution after a (possibly) large enough number of iterations [4].

Chapter 5

Smart strategies on DDP

This chapter presents two complementary strategies proposed in this work that are directly applied to the iterative process of the DDP. The goal is to improve DDP efficiency by avoiding unnecessary calculations and use of memory that would not provide useful additional information to the solution process, without loss of the convergence properties of the algorithm.

5.1 Local convergence test (LCT)

Along the DDP iteration process, state variables are transmitted forward on the tree, i.e., the optimal final conditions on the current stage are the initial conditions for the next stage. Similarly, Benders cuts are transmitted backward on the tree, in order to improve the model of the cost function of the current stage for the subproblem of the previous stage, around the current value of the state variables. As explained in the previous chapter, the convergence test of the method is based on a comparison of the actual costs obtained in the forward passes (upper bound) with costs based on a lower convex approximation of future costs, which is built during the backward iterations (lower bound). When these two values are close enough the solution is considered as optimal.

The concept of the Local Convergence Test (LCT) uses a similar idea to the DDP global convergence, but applied to each subtree instead of only in the complete scenario tree. The goal is to avoid calculations and memory that would not append useful information to the problem in the solution process.

Subtrees are defined as a subset of nodes of the scenario tree which can be considered as tree itself. We consider one subtree for each node on the complete tree where it is composed by the node itself and all its descendants, therefore, the number of subtrees considered for LCT is equal to the number of nodes on the tree. However, there are two special kinds of subtrees and for neither of them we perform the LCT test:

- Trivial subtrees: formed by the leaf nodes, where the subtree is the node itself. The LCT in that case is also trivial and always true;

- complete tree: the subtree which contains the root node is the whole scenario tree itself. The LCT test in this case is already performed in the DDP solving procedure.

The LCT strategy is performed at each DDP iteration, and consists in calculating local upper and lower bounds for the considered subtrees and then proceed a local convergence test, as detailed below:

- *Local lower bound* ($\underline{z^k_{t,u}}$): for iteration k, subproblem SP^u_t, given a state $x^s_{t-1}, u \in \Omega^s_{t-1}$, the local lower bound is given by:

$$\underline{z^k_{t,u}} = \min_{x^u_t \in \mathcal{X}^u_t(x^s_{t-1})} f_t(x^u_t) + \mathop{\mathbb{E}}_{v \in \Omega^u_t}[M^k_{t,v}(x^u_t)].$$

- *Local upper bound* ($\overline{z^k_{t,u}}$): for iteration k, subproblem SP^u_t, given a state $x^s_{t-1}, u \in \Omega^s_{t-1}$, we take the forward solution $\{x^{u,k}_t, x^{v,k}_{t+1}, ..., x^{j,k}_T\}$ on the subtree $v \in \Omega^u_t, ..., j \in \Omega^q_{T-1}$ spanned from subproblem SP^u_t to compute the upper bound:

$$\overline{z^k_{t,u}} = f^u_t(x^{u,k}_t) + \sum_{v \in \Omega^u_t} p^v_{t+1}[f_{t+1}(x^{v,k}_{t+1}) + ... + \sum_{j \in \Omega^q_{T-1}} p^j_T f_T(x^{j,k}_T)...]$$

- *Local convergence test*: local convergence gap $gap^k_{t,u}$ for iteration k and subproblem SP^u_t is given by:

$$gap^k_{t,u} = \frac{\overline{z^k_{t,u}} - \underline{z^k_{t,u}}}{\overline{z^k_{t,u}}}$$

and the convergence test, which determines the stopping criterion is:

$$gap^k_{t,u} < \epsilon,$$

where $\epsilon \in \mathbb{R}$ is the desired tolerance for the optimality of the solution.

The test is proceeded before the backward pass: if the LCT is successful in a given node, it means that the future cost approximation is already accurate (given a convergence criterion) for the current value of the state variables. Therefore, it is not necessary to build more linear cuts for this node, which avoids additional CPU, memory and the computational burden of adding more cuts. More importantly, it may avoid several LP computations because there is no need to compute the

backward pass for all nodes on this subtree. We note that, differently from the global convergence test, the LCT depends on the values of the state variables, i.e., if a subtree has converged in a certain iteration, it may not necessarily converge on the next one, because the values of the corresponding state variables for the root node on this subtree may change.

5.1.1 Global and local convergence

In order to determine the convergence gap to be used in the local convergence test we compare the relation between the local and global gaps. Considering this, we show that if the subtrees corresponding to all descendant nodes of the root node converges with the tolerance ϵ then the global tree converges with the same tolerance.

Considering $u \in \Omega_1$ the descendants of the root node, the gaps and the tolerance test of these nodes are given by:

$$gap_{2,u} = \frac{\overline{z_{2,u}} - \underline{z_{2,u}}}{\overline{z_{2,u}}} < \epsilon$$

We also can write that:

$$\overline{z_{2,u}} - \underline{z_{2,u}} < \epsilon * \overline{z_{2,u}}, \forall u \in \Omega_1$$

Which means that all descendant nodes of the root node have converged with ϵ tolerance. We also can write:

$$\sum_{u \in \Omega_1} \overline{z_{2,u}} - \sum_{u \in \Omega_1} \underline{z_{2,u}} < \epsilon * \sum_{u \in \Omega_1} \overline{z_{2,u}}$$

The upper bound of the root node with relation of its descendants can be write as:

$$\overline{z_1} = f_1(x_1) + \sum_{u \in \Omega_1} \overline{z_{2,u}}$$

So we have:

$$\overline{z_1} - f_1(x_1) - \sum_{u \in \Omega_1} \underline{z_{2,u}} < \epsilon * \sum_{u \in \Omega_1} \overline{z_{2,u}}$$

$$\overline{z_1} - (f_1(x_1) + \sum_{u \in \Omega_1} \underline{z_{2,u}}) < \epsilon * \sum_{u \in \Omega_1} \overline{z_{2,u}}$$

Considering that the lower bound is always a lower approximation of a convex function, the lower bound of the root node is less than the :

$$\underline{z_1} = f_1(x_1) + \mathbb{E}[M_{2,s}^k(x_1)] < f_1(x_1) + \sum_{u \in \Omega_1} \underline{z_{2,u}}$$

So:

$$\overline{z_1} - (\overline{z_1}) < \epsilon * \sum_{u \in \Omega_1} \overline{z_{2,u}}$$

And, the sum of the upper bounds are also below of the root node upper bound:

$$\overline{z_1} - (\overline{z_1}) < \epsilon * \sum_{u \in \Omega_1} \overline{z_{2,u}} < \epsilon * \overline{z_1}$$

Leading to:

$$\overline{z_1} - \overline{z_1} < \epsilon * \overline{z_1}$$
$$\frac{\overline{z_1} - \overline{z_1}}{\overline{z_1}} < \epsilon$$

In conclusion, if the subtrees have converged with tolerance ϵ the global tree also converges with the same tolerance.

5.2 State Variables Stability Test (VST)

The State Variables Stability Test (VST) is a complementary strategy to the LCT, and aims to avoid the resolution of LPs in the forward iteration. The VST test makes use of the concept that a subtree will always converge at the current DDP iteration, if it has already reached convergence at a past iteration for the same values (given a numerical tolerance) of the state variables for the root node of this subtree. In this sense, we save all values of state variables for which the subtree has converged in past iterations and avoid the forward pass on that subtree when such values appear again (given a numerical tolerance). We note that this procedure is very effective in saving CPU time, because eliminates the entire forward and backward passes for the subtree that has become stable in a given iteration

Figure 5.1 exemplifies the convergence tests defined in this chapter, as described below:

- *Global convergence*: the input state variables for node 1 (x_0) are constant. If this node yields a solution x_1^k with approximated cost $\underline{z^k}$ close enough to the real cost $\mathcal{Q}^2(x_1^k)$ for the subtree spanned from that node (which is the complete scenario tree of the problem), global convergence has been reached.

- *Local convergence* (node 2): for the same cost function approximation, a certain state reach local convergence and another may not:

  - *case A*: for a certain set of input state variables (x_1^k) an output state x_2^k is obtained. This output state gives an approximated future cost of $\underline{z_{2,1}^k}$ and local convergence is not detected because in this region the approximated

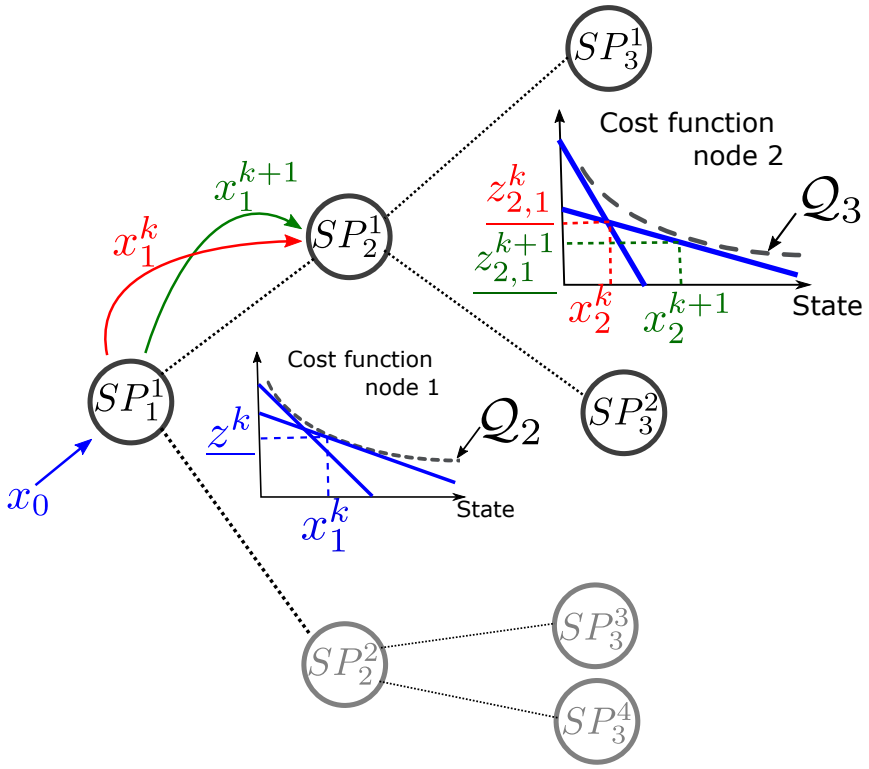


Figure 5.1: Example of global convergence, local convergence and state variables stability tests.

value of future cost given by the model is much smaller than the real cost $\mathcal{Q}^3(x_2^k)$;

- *case B*: for another set x_1^{k+1} of input state variables, an output state x_2^{k+1} is obtained. This output state gives an approximated future cost of $\underline{z_{2,1}^{k+1}}$, which is an accurate of approximation of the real cost. Therefore, we may say that node 2 has converged for an input state x_1^k and there is no need to proceed a backward operation for this node.

- *State variables stability* (node 2): suppose that, at iteration $k+1$, local convergence was detected for an input state x_1^{k+1}, yielding a lower bound that had been considered converged, as shown in Figure 5.1. Since the lower bound is monotonically increasing (for a constant input state), if in any further iteration $n > k+1$ the input state x_1^{k+1} occurs again (i.e., $x_1^n = x_1^{k+1}$), we neither need to recalculate the costs nor to build Benders cuts, because the algorithm has already performed such procedures in the previous iteration $k+1$.

Figure 5.2 shows the flow diagram of one iteration with respect to the subproblem of a given node of the tree. The node receives an input state from its ascendant node, and the VST is performed. If the test succeeds, the process for this node is terminated, otherwise the forward pass for the subtree spanned from this node

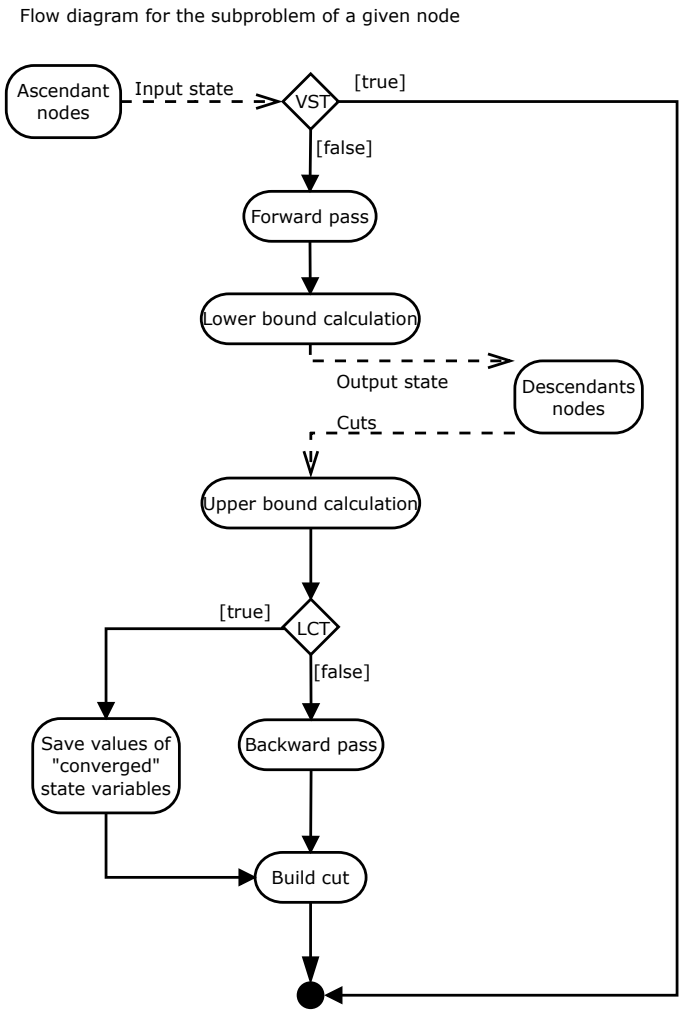


Figure 5.2: Flow diagram of one iteration with respect to the subproblem of a given node of the tree.

is performed. At the end of this procedure, local lower and upper bounds for this node are tested and the backward pass for this subtree is performed only if local convergence has not been detected. Finally, after the backward pass reaches this node, a Benders cut is built and sent to the ascendant node.

We expect that, as the iterative process evolves and the approximations of the future costs become more accurate, several subtrees will reach local convergence and stabilize the values of state variables along their nodes. This drastically reduces the number of solved subproblems in each forward and backward pass and, as a consequence, the CPU time of each DDP iteration.

Chapter 6

Parallel Dual Dynamic Programming

This chapter explores the parallel execution of the DDP algorithm. Firstly, a traditional parallelization scheme for this algorithm is introduced. Then, an asynchronous version of the DDP method proposed in this paper is presented. The asynchronism introduced in the DDP method makes it more suitable for parallel environments, although the serial execution of this algorithm may be harmed. Finally, another approach which is partially asynchronous is also proposed with the goal of increasing the convergence rate of the method.

6.1 Traditional Parallel Dual Dynamic Programming Algorithm

A classical way of parallelizing a DDP method was proposed by [9], which explores independence of the scenarios at the same level of the tree. The main idea is to apply the traditional DDP algorithm, with forward and backward passes, solving the nodes at the same stage in parallel. Because the subproblem of a node requires either the state of its ascendant node (in the forward pass) or the Benders cuts of its descendants nodes (in the backward pass), there exists an inter-level dependency that creates a synchronization point in each time level.

These synchronization points are a major drawback of the method. Besides creating a dependence between the levels in both backward and forward passes, it also limits the granularity of the algorithm. The maximum granularity of a time level is given by the number of scenarios at this level. Therefore, since each time level has a different granularity, idleness along the traditional DDP solving procedure is expected, specially in the prior levels of the scenario tree. It is also important to note that parallel performance, granularity and dependencies are strongly related to

the size and shape of the scenario tree.

In order to describe the classical DDP parallel scheme we use the example tree of Figure 6.1, which shows how the subproblems (nodes) are distributed among the processors. Since the tree has a total number of four scenarios at the last time level, the maximum granularity is four processors, being represented by $p0$, $p1$, $p2$ and $p3$, .

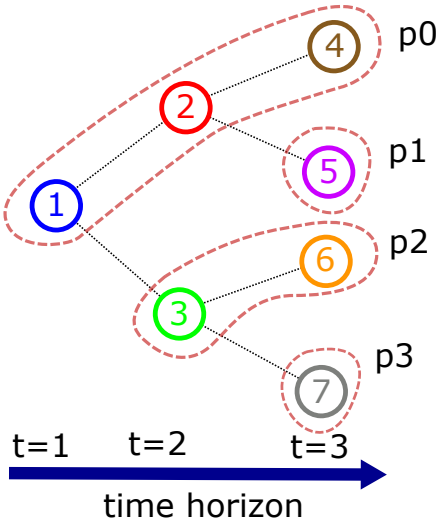


Figure 6.1: Example tree and the distribution of the sub-trees among the processors for the DDP traditional parallel method.

Figure 6.2 presents the Unified Modeling Language (UML) sequence diagram of the iteration loop on the classical parallel DDP algorithm. The diagram represent the lifelines of the four processors, $p0$, $p1$, $p2$ and $p3$, and the information to be exchanged among them, which are the values of state variables in the forward pass and the Benders cuts in the backward pass. Firstly, $node1$ is solved by processor $p0$, while the other processors are waiting; the state of $node1$ is transmitted for processor $p2$, which will solve $node3$, at the same time processor $p0$ solves $node2$. Then, the output states of nodes 2 and 3 are transmitted to processors $p1$ and $p3$ in order to solve the subproblems for the last time level (nodes 4, 5, 6 and 7), by all processors working in parallel. After that, the backward iteration begins with the transmission of the cuts to the descendant nodes on the tree. Only processors $p0$ and $p2$ proceed this pass by solving nodes 2 and 3, respectively.

It is possible to note that idle times are an important issue for all processors (except for $p0$), since they do not perform any operation in a portion of the iteration time line. Such idleness is critical for processors $p1$ and $p3$, which are responsible for the solution of only one node. At the end of each iteration a convergence test is performed, for which a special care must be taken in transmitting the present costs of all nodes to processor $p0$, which will compute the upper bound. We note that

the convergence process of the parallel version of the traditional DPP algorithm is exactly the same as its non-parallel (serial) version, since the parallelization is applied without changing the algorithm.

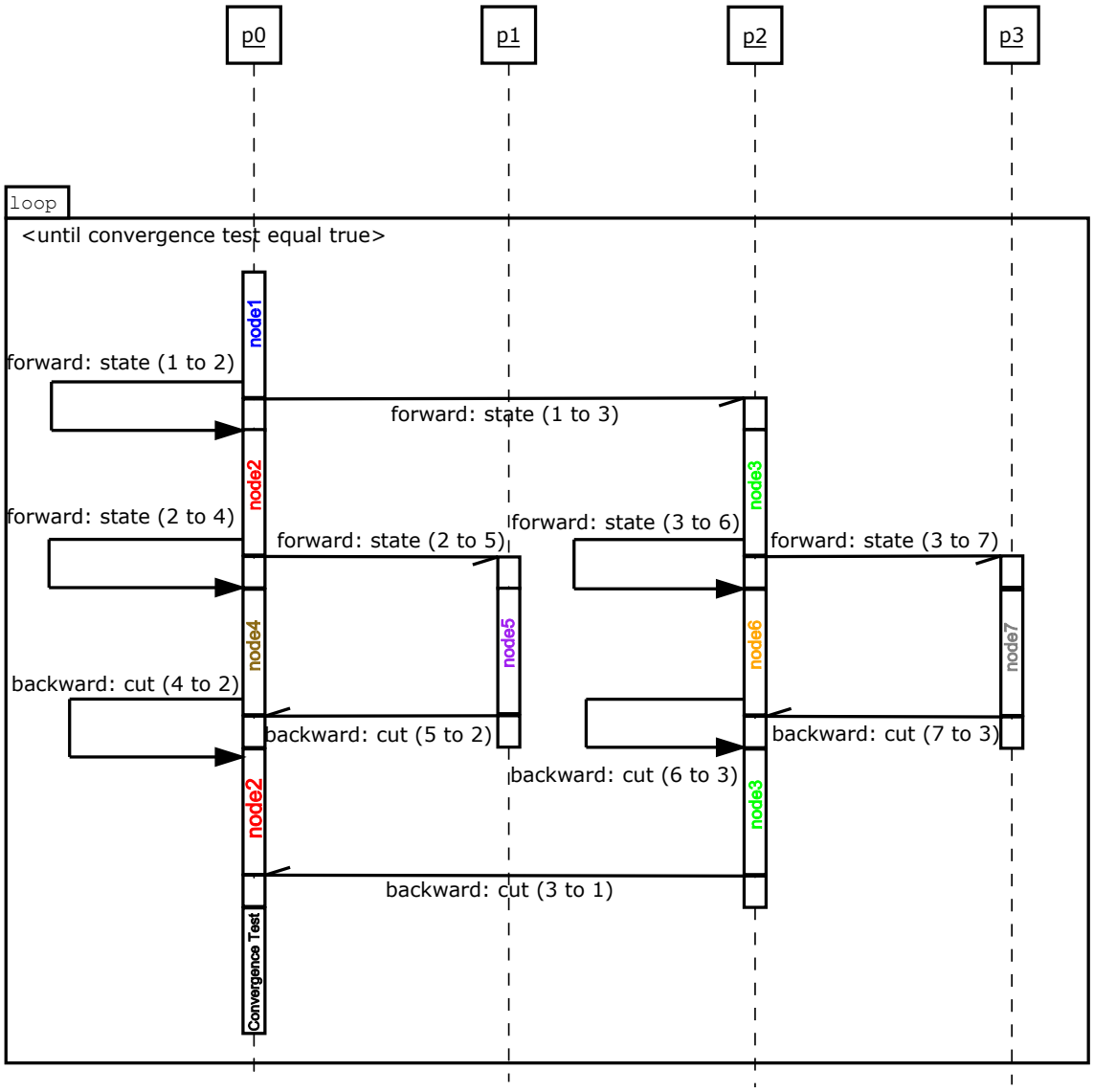


Figure 6.2: UML sequence diagram of DDP parallel algorithm

6.2 An Asynchronous Dual Dynamic Programming Algorithm

The inherent time level dependence in the forward and backward passes of the DDP algorithm can be very restrictive to parallelization schemes. In order to overcome this limitation, an asynchronous version of the DDP algorithm called ADDP is formally proposed here and can be more naturally adapted to parallel environments.

A similar asynchronous idea was proposed for deterministic problems in [35], to allow the use of several processors simultaneously, since in the deterministic case the application of traditional DDP algorithm turned out to be inherently serial. We adapted their idea to the stochastic framework, with a set of improvements in order to better proceed the DDP algorithm in a parallel environment for stochastic problems.

The ADDP algorithm has neither a forward pass nor a backward pass; thus it does not iterate in the same way as the traditional DDP approach. Instead, ADDP iterates by "steps", where each step is defined as an independent resolution of all subproblems of the scenario tree. The exchange of information occurs simultaneously along the tree in between steps: state variables for the descendant stages and Benders cuts for the ascendant stage of each given subproblem. Considering this, the nodes are totally independent from each other and can be solved simultaneously within a step. However, in between steps there is a synchronization point.

The same scenario tree as before is used to describe the ADDP scheme, and Figure 6.3 shows the distribution of duties among the processors for this tree. Since the maximum granularity of the ADDP method corresponds to the number of nodes on the tree, seven processors can be used in this case: $p0$ to $p6$.

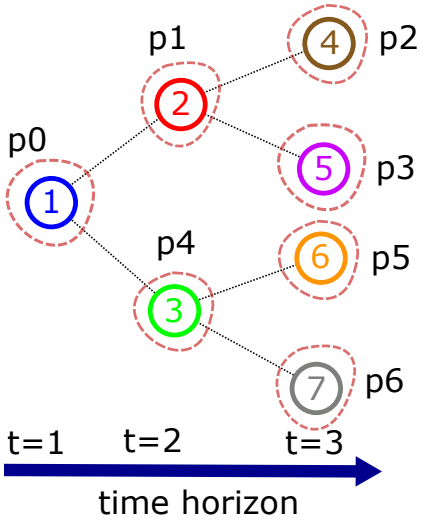


Figure 6.3: Example tree and the distribution of the subtrees among the processors for the ADDP strategy.

Figure 6.4 represents the UML sequence diagram of the iterative loop of steps for the ADDP algorithm. Each step is composed by two well defined phases: the first one consists in solving the LP subproblem of each node; therefore, within a step, all nodes of the tree can be solved independently. The second phase consists in exchange of information: state variables and cuts are transmitted to the ascendant

and descendants nodes, respectively, and cost information is transmitted to the master noted in order to perform the convergence test.

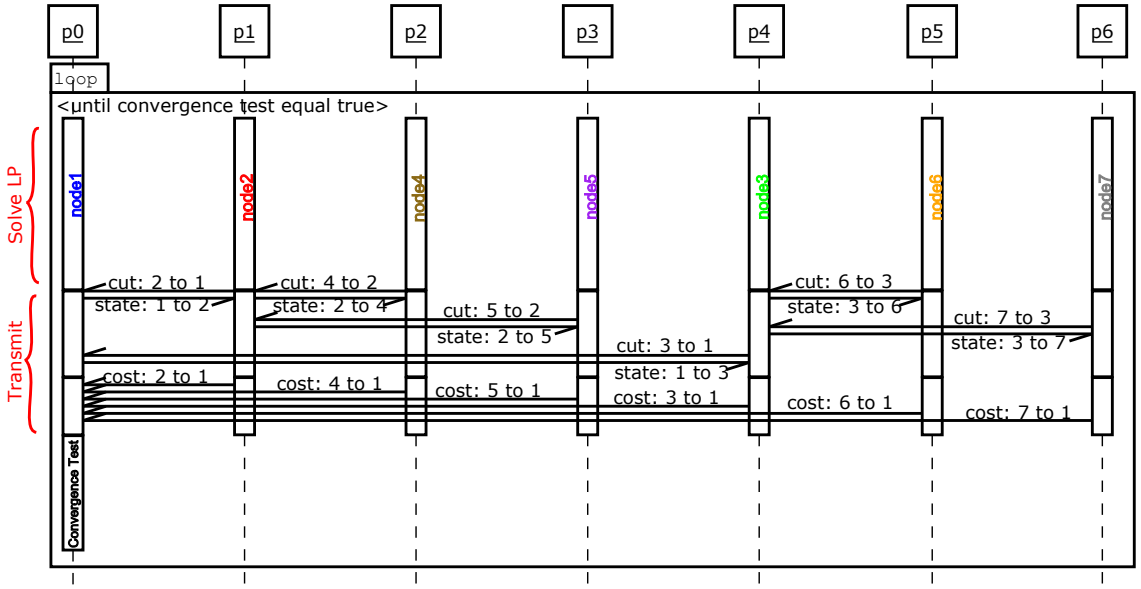


Figure 6.4: UML sequence diagram of the ADDP parallel algorithm.

We also show in Figure 6.5 a flow diagram of processes in the slave processors ($p1$ to $p6$) and in the master processor ($p0$). Firstly, the LP subproblem of the corresponding node is initialized in each processor. Then the LP is updated with new information (cuts and states), solved and new values of local state variables and cuts are computed. We proceed by sending state variables to descendants nodes and receiving cuts from the ascendants nodes. In the same time, a new cut that has just been built is sent backwards and the new state values are received from the ascendant node. Moreover, cost information is transmitted to the master node by all nodes, and such master node computes the convergence and broadcasts the convergence result (i.e., a stop signal if the stopping criterion has been met). Otherwise, all processors repeat the same process of updating the LP and a new step of the ADDP algorithm is performed.

At the fist step, there are no state values available for the subproblems, except for the root node, where the state is given as an input data for the algorithm. Therefore, as the algorithm needs to be initialized with state values to start the iterative process, in the first step we applied the initial state from the first node to all nodes.

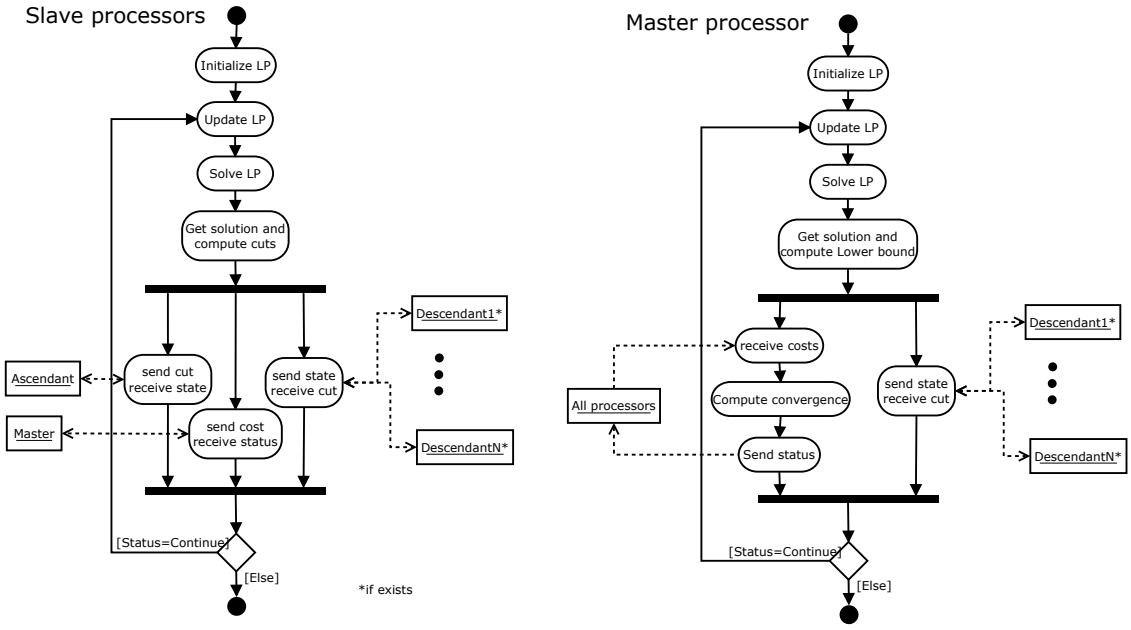


Figure 6.5: Flow diagram of ADDP process for the slave and master processors

6.2.1 Convergence of the ADDP algorithm

Convergence of the ADDP method carries the same concepts and properties of the traditional DDP convergence, as described in section 4.3. The lower bound is the value of the optimal solution of the first (root) node. The upper bound calculation, on the other hand, becomes more complex since the method has no forward passes. However, it is possible to identify an implicit forward pass along the ADDP steps: as the state variables are transmitted to descendant nodes between steps, a sequence of T steps - where T is the number of time levels - comprises a complete propagation of a forward pass from the root node to the leaf nodes.

Figure 6.6 illustrates the implicit forward pass from step k to step $k+2$ of the ADDP algorithm for the previous example. The convergence test is calculated as follows: the lower bound $\underline{z^k}$ is the solution of first node in step k, similarly to the traditional DDP lower bound. On the other hand, the upper bound $\overline{z^k}$ is composed by the present costs of the nodes in steps k to $k+2$, following the forward propagation. In this sense, the convergence test on step k needs the information on step $k+2$ in order to be computed.

Algorithm 1 presents the pseudo code of the convergence test in the proposed ADDP method. After computing the upper and lower bounds, a convergence gap is obtained by the relative difference between the two bounds. The algorithm reaches convergence in the current step if the gap is less than a fixed tolerance. As required for a reliable stopping criterion, once convergence is detected in a given step, the algorithm will be always convergent for furthers steps because of the monotonicity

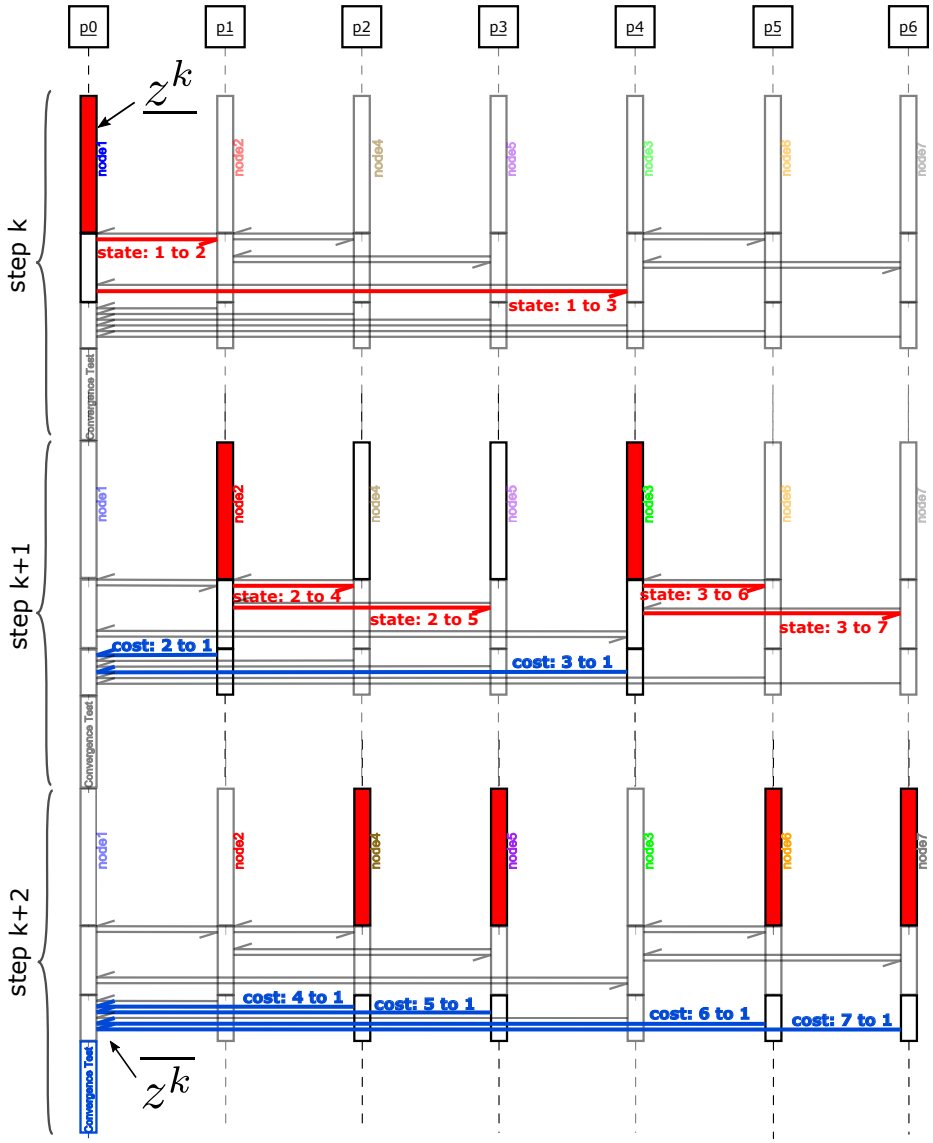


Figure 6.6: Convergence process of the ADDP algorithm.

property of the bounds.

6.3 Partial ADDP algorithm

The asynchronous DDP method is capable of solving all nodes on the scenario tree in a completely independent mode; thus allowing full node parallelization as compared to the traditional DDP algorithm. Nevertheless, when the number of available processors is lower than the number of nodes in the tree, a certain number of nodes will share a same processor, and their subproblems will be solved sequentially but asynchronously.

For this reason, in order to improve the convergence rate of the ADDP method, a Partial Asynchronous DDP approach (PADDP) is also proposed, where we in-

Algorithm 1 The ADDP method convergence calculation

▷ Lower bound computation

1: $\underline{z^k} = f(x_1^{*^k}) + \mathbb{E}_{u \in \Omega_1}[M_{2,u}^k(x_1^{*^k})]$

▷ Upper bound computation

2: $\overline{z^k} = f(x_1^{*^k})$
3: **for** $i = 1, ..., T-1$ **do** ▷ collecting costs from second to last
4: $period = i + 1$ ▷ periods consulting the corresponding step
5: **for** $s \in \cup_{\forall u} \Omega_{period}^u$ **do**
6: $\overline{z^k} = \overline{z^k} + p_{acc_{period}}^s \times f(x_{period}^{*^{s,k+i}})$
7: **end for**
8: **end for**
9: $\overline{z^k} = \min(\overline{z^{k-1}}, \overline{z^k})$

▷ Convergence gap

10: $gap^k = (\overline{z^k} - \underline{z^k})/\underline{z^k}$
11: $convergence^k = false$
12: **if** $gap^k \leq \epsilon$ **then** ▷ Convergence test
13: $convergence^k = true$
14: **end if**

tentionally introduce an inter-level synchronism for nodes at the same processor to allow one node to use information computed by other nodes during the same ADDP step. Therefore, the idea of PADDP is to pass information on the state variables from one node to its descendant node at the same ADDP step if both nodes are solved at the same processor. It is important to note that, for a problem comprising a scenario tree with N nodes, if there are N available processors the PADDP and ADDP methods are identical. Conceptually, the PADDP approach gradually leads to the ADDP method as the number of processors tends to the number of nodes.

The upper bound computation for convergence test varies on how the implicit forward pass is split among the PADDP steps, depending on the number of processors. For example, in one limiting case, the PADDP with a single processor has an entire forward pass within one step (which is identical to the DDP forward pass), and each step yields an independent convergence test. If two processors are employed, a complete forward pass will be performed in two consecutive steps. On the other extreme case, the PADDP approach with N processors needs N steps to produce an upper bound.

Figure 6.7 is the UML sequence diagram representing the PADDP method of the example tree when there are two available processors. In such case the first processor will solve nodes 1, 2, 4 and 5, the second processor will solve nodes 3, 6 and 7, and the state variables between them are shared at the same step.

For the PADDP algorithm it is necessary to define how the nodes will be spread among the processors, since this may have an effect in the algorithm progress. A static distribution is made before the solution of the problem as a preprocessing

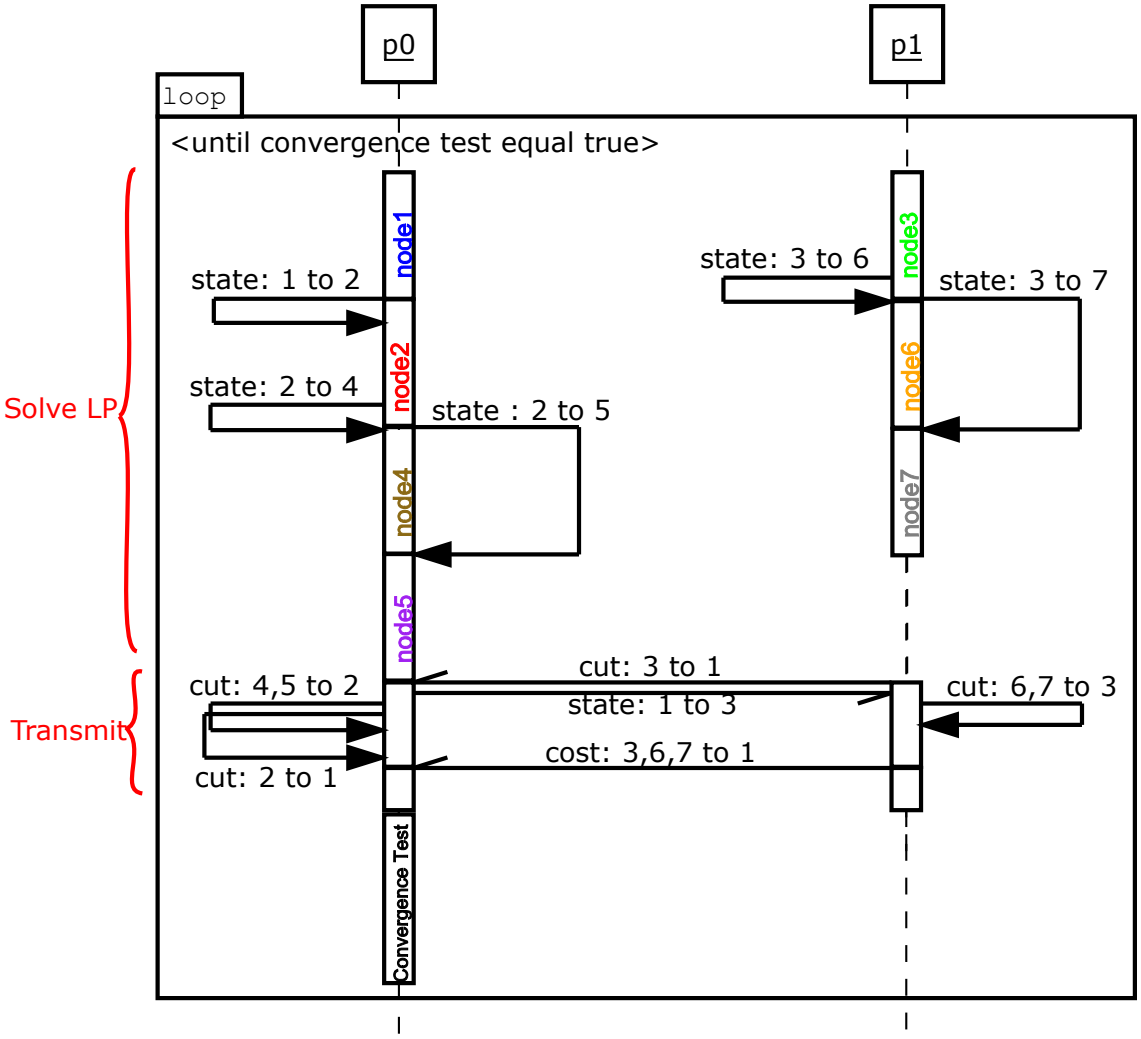


Figure 6.7: UML sequence diagram of an iteration of the PADDP method with 2 processors

phase. We allocate an uniform number of nodes in each processor to pursue a good load balance, two different strategies were experiment to proceed this distribution:

- **Node approach:** which divides the nodes among the processors according to their numbers, as presented in Figure 6.3, enumerated by their position in the time horizon.
- **Scenario approach:** which attempts to allocate in the same processor the nodes that are located in a same path.

We note that, despite having the same forward process with one processor, the PADDP algorithm differs from the traditional DDP approach in the backward pass, which is implicit among the steps in the PADDP method.

Chapter 7

Results

This chapter presents the results of studies made to evaluate the efficiency of the proposed schemes in multistage linear stochastic optimization problems solved using DDP decomposition with a multi-cut approach [5]. All the algorithms, the proposed ones and the originals were implemented by the author with the same resources in order to be comparable. The LCT and VST strategies proposed in this work were compared to the standard DDP strategy in terms of number of operations that were avoided and time taken until convergence. In addition, the proposed ADDP and PADDP strategies were compared to a parallel scheme that is traditionally used for the DDP algorithm (section 6.1) where time, efficiency and speedup were assessed for the three parallel schemes with different amounts of cores.

7.1 Study cases

We considered a stochastic HTC problem for part of the large-scale Brazilian interconnected system, as described in chapter 3. The model comprises 84 hydro plants, from which 44 are reservoirs with regularization capacity, and 46 thermal plants. To evaluate the algorithms more carefully, we generated different scenario trees by varying the length of the time horizon and the stochastic scenarios of water inflows. Four different study cases were constructed with the scenario trees described in Tables 7.1, 7.2, 7.3 and 7.4.

Table 7.1: Structure of the scenario tree for study case 1

Total number of nodes:	127						
Time periods:	1	2	3	4	5	6	7
# Scenarios per period:	1	2	2	2	2	2	2

Each LP subproblem in a node has around 1200 variables and 2500 constraints, besides the Benders cuts that are appended to the LP along the iterative process.

Table 7.2: Structure of the scenario tree for study case 2

Total number of nodes:	781				
Time periods:	1	2	3	4	5
# Scenarios per period:	1	5	5	5	5

Table 7.3: Structure of the scenario tree for study case 3

Total number of nodes:	306						
Time periods:	1	2	3	4	5	6	7
# Scenarios per period:	1	1	1	1	1	1	300

Table 7.4: Structure of the scenario tree for study case 4

Total number of nodes:	221											
Time periods:	1	2	3	4	5	6	7	8	9	10	11	12
# Scenarios per period:	1	20	1	1	1	1	1	1	1	1	1	1

Three load blocks are represented in each time period. We considered monthly periods, except for case 3, where periods 1 to 6 correspond to weeks, in order to emulate the official studies for the definition of spot prices for the Brazilian market ([11]).

7.2 Hardware and software features

We developed the algorithms in Fortran language and Linux environment. Since the applied parallelism technique was suitable for a distributed memory environment based on message passing, Message Passing Interface (MPI) was used to implement the parallel algorithms. The study cases were executed in a cluster with several nodes containing 2 AMD processors, 6 cores and 2.6GHz each, 2MBytes of cache memory and 96 GBytes of RAM memory.

The study cases were executed five times each and the time computed in average excluding any outliers execution.

7.3 Assessment of LCT and VST strategies

The performances of the strategies presented in Chapter 5 were evaluated in a multistage linear stochastic optimization problem through the study cases described previously. We compare the number of LPs solved in each iteration, in the forward and backward passes, as well as the time taken until convergence (within a given tolerance) when using 1 processor and 8 processors (with traditional parallel implementation on DDP - section 6.1), for the following methods:

- traditional DDP method, as a benchmark (section 4.2);
- DDP method with the proposed LCT strategy (section 5.1);
- DDP method with the proposed VST strategy, which also includes the LCT approach by construction (section 5.2).

The results are exposed and discussed for each study case in the following.

7.3.1 Results and analysis - Study case 1

Table 7.5 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds in the three methods assessed:

Table 7.5: Study case 1 - lower and upper bounds and convergence gap of the executions.

Method	Upper bound	Lower bound	Gap
DDP	11900728.6	11900828.5	0.0008389382
LCT	11900728.2	11900810.4	0.0006908652
VST	11900738.7	11900842.2	0.0008702884

It is important to notice that for the three methods the solution bounds are consistent with the fact that the problems been solved are the same as well as the solution.

Figures 7.1 and 7.2 show the number of LPs solved in the forward pass and in the backward pass, respectively, for each one of the 18 iterations before the global convergence. For the classical DDP algorithm, the number of LPs solved is constant along iterations, since each forward pass (with 127 LPs) and backward pass (with 63 LPs) traverses the whole scenario tree. On the other hand, the LCT strategy avoids solving LPs in the backward passes, so the total number of solved LPs tends to be smaller (Figure 7.2). Such difference is more evident as the process approaches the optimal solution, since more and more sub-trees reach convergence.

The VST strategy avoids LPs in the backward and in the forward passes of the DDP algorithm, thus yielding an even greater reduction in the number of solved LPs, since the forward pass has twice the number of LPs than the backward pass. However, we cannot expect the VST strategy to always provide a lower number of LPs as compared to LCT, because there is a somehow random effect caused by multiple optimal solutions for the subproblem of a given node, which may cause different behaviors in further stages.

We also note that VST had a small impact in the second iteration and LCT did not. The reason is that, since the future cost function of the leaf nodes are

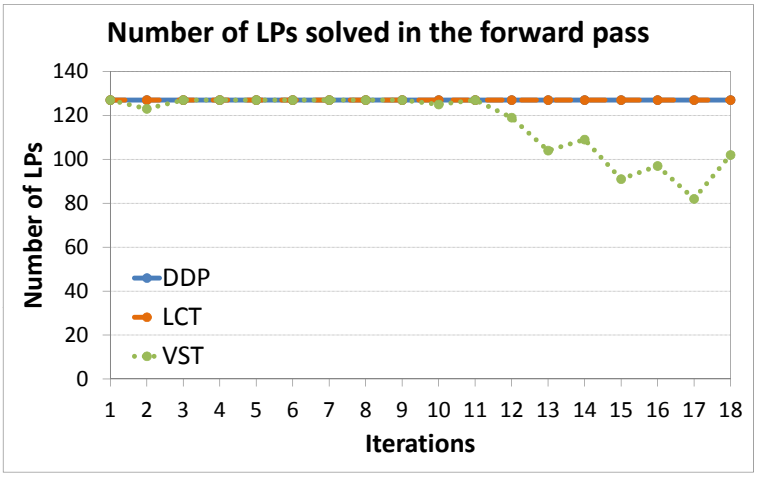


Figure 7.1: Number of LPs solved per iteration in the forward pass for the study case 1.

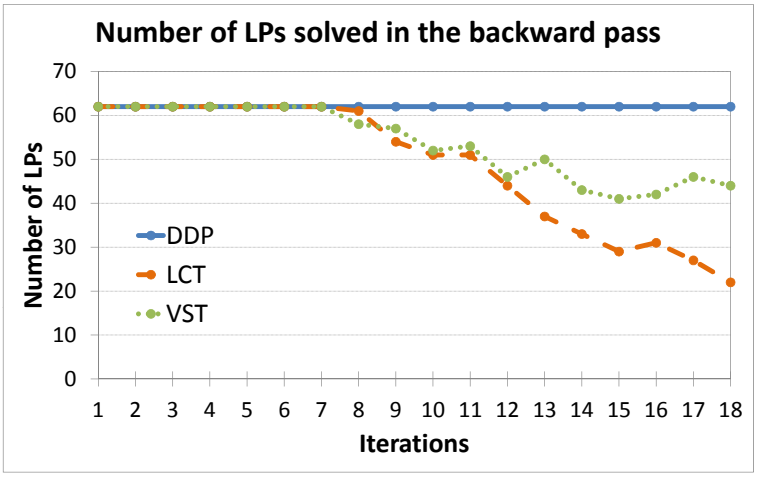


Figure 7.2: Number of LPs solved per iteration in the backward pass for the study case 1.

fixed, they do not perform backward passes, therefore LCT is not effective on them. However, the VST strategy may avoid forward passes in the leaf nodes if the values of state variable are stable from one iteration to another.

Figure 7.3 shows the time each of the 18 iterations took to proceed, both in a serial execution and in a parallel execution with 8 processors. As a consequence of

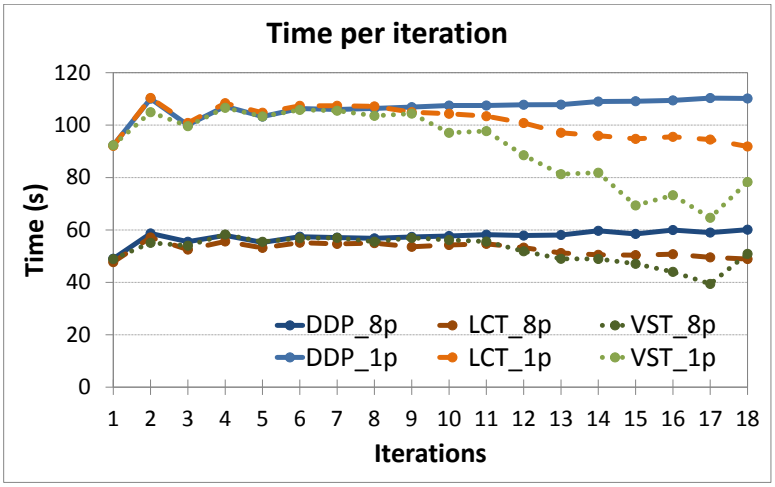


Figure 7.3: Time per iteration in study case 1 by using one and eight processors.

the reduction on the number of LPs solved, we observe the lower execution times for LCT and VST strategies as compared to the traditional version of DDP method, specially in the final iterations. We note that for this case VST presents the best (lower) time consumption, performing some iterations 40% faster than DDP (with one processor). In the parallel case the gain in terms of time is lower as compared to the single-processor case. The reason is that part of the reduction is obtained by not solving subproblems in a processor which later would be idle anyway because of the DDP time dependency.

7.3.2 Results and analysis - Study case 2

Table 7.6 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds in the three methods assessed:

Table 7.6: Study case 2 - lower and upper bounds and convergence gap of the executions.

Method	Upper bound	Lower bound	Gap
DDP	11765793.1	11765899.3	0.0009027470
LCT	11765780.5	11765889.1	0.0009236680
VST	11765790.3	11765897.0	0.0009076835

As well as the previous case, the bounds are consistent reinforcing the fact that the methods lead to the same global solution.

Figures 7.4 and 7.5 show the number of LPs solved in the forward pass and in the backward pass for each of the 16 iterations before the global convergence of the

study case 2.

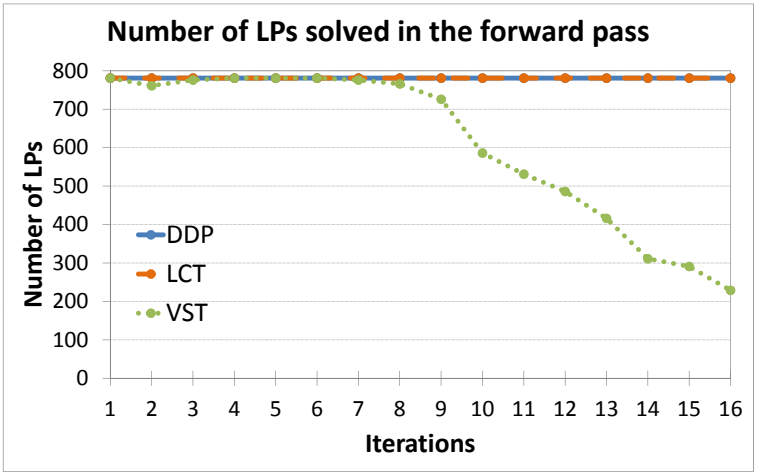


Figure 7.4: Number of LPs solved per iteration in the forward pass for the study case 2.

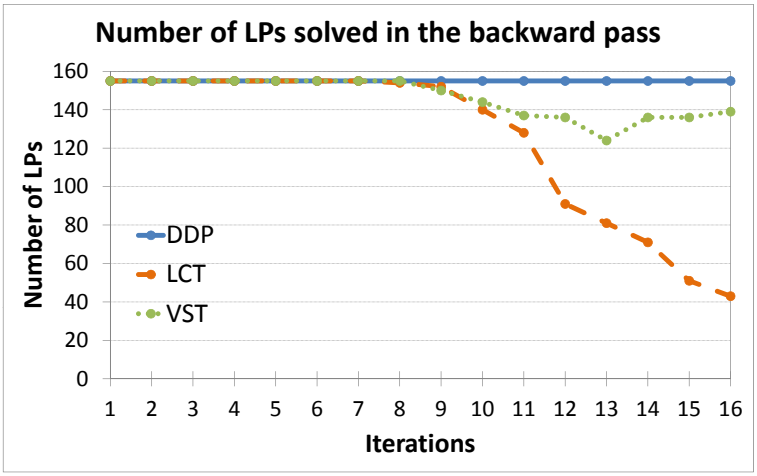


Figure 7.5: Number of LPs solved per iteration in the backward pass for the study case 2

In this case, the number of LPs in the forward pass is 781 and in the backward pass is 155, therefore the classical DDP algorithm will solve all these LPs per iteration since it traverses the whole scenario tree. On the other hand, the LCT strategy

provides a reduction on the number of LPs in the backward passes, which can be more than 50% in the final iterations.

The impact observed on VST in the backward pass was lower than the LCT approach, however in the forward pass the reduction in the number of LPs solved was significant. Since in this study case the forward pass is proportionally more costly than the backward pass, the impact of VST on the execution time was more evident. Figure 7.6 shows the time taken for each method using one and eight processors. Even though we observe that the impact of LCT in the execution time is small, on the other hand, the impact of VST was significant, in both experiments, being more evident with a single processor.

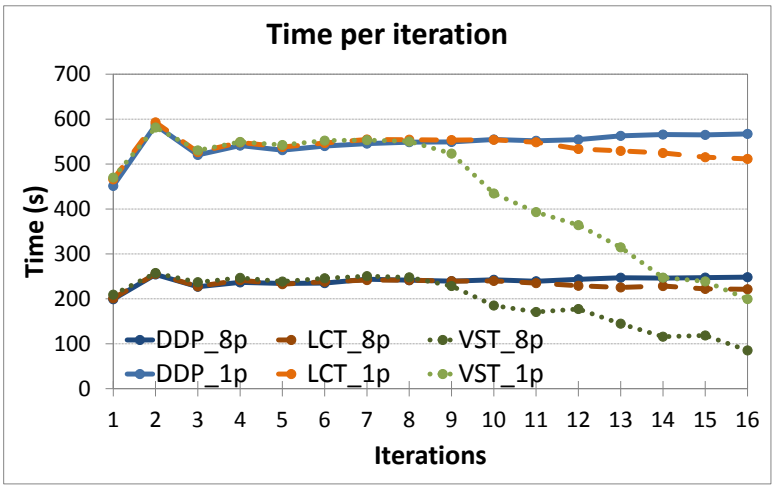


Figure 7.6: Time per iteration in study case 2 by using one and eight processors.

7.3.3 Results and analysis - Study case 3

Table 7.7 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds in the three methods assessed:

Table 7.7: Study case 3 - lower and upper bounds and convergence gap of the executions.

Method	Upper bound	Lower bound	Gap
DDP	12812793.4	12812918.6	0.0009769356
LCT	12812793.4	12812918.6	0.0009769356
VST	12812793.4	12812918.6	0.0009769356

In this case of study the bounds and consequently the gap are the same in the three executions. The shape of the scenario tree presents six deterministic time steps

followed by a 7th time step with 300 scenarios. Therefore, 306 nodes perform the forward passes and 5 nodes perform the backward passes. The local convergence test on these 5 nodes means that almost the whole tree has converged, and no cut will be built, so the whole tree is convergent. In conclusion, the local convergence test does not apply to this type of tree. Concerning the VST, the state variable stability on a deterministic tree also means global convergence, since no new information will be appended to the problem. Therefore, VST is also not applicable to this type of scenario tree. Figures 7.7 and 7.8 show the number of LPs solved in the study case 3, reinforcing the identical execution of the three methods.

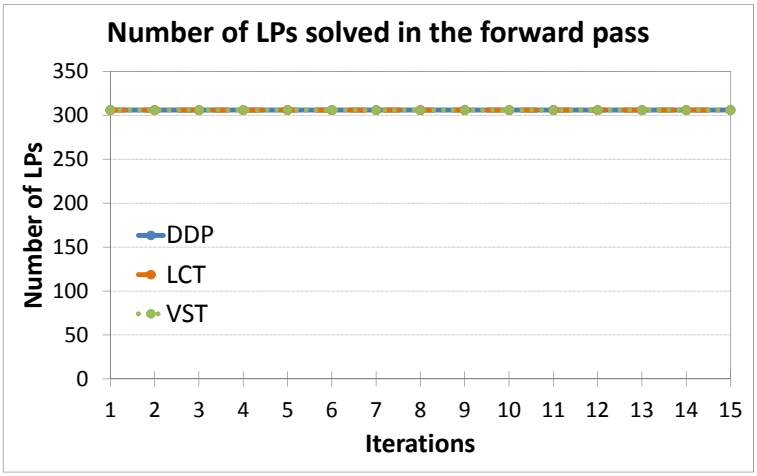


Figure 7.7: Number of LPs solved per iteration in the forward pass for the study case 3.

Figure 7.9 shows the time to proceed each iteration until convergence. Since the LCT and VST have no effect on the number of LPs solved, we observe that both strategies have a higher time than DDP, which can be explained by the extra calculations in order to perform the local testes. In the case with 8 processors, the time of the three strategies are similar.

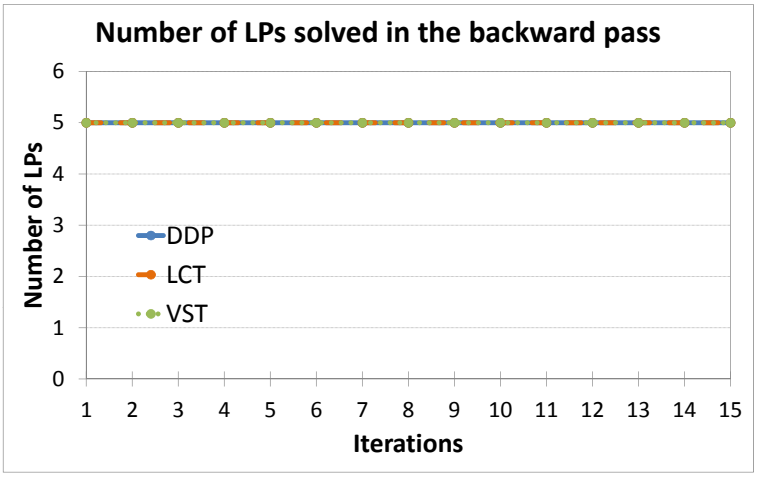


Figure 7.8: Number of LPs solved per iteration in the backward pass for the study case 3.

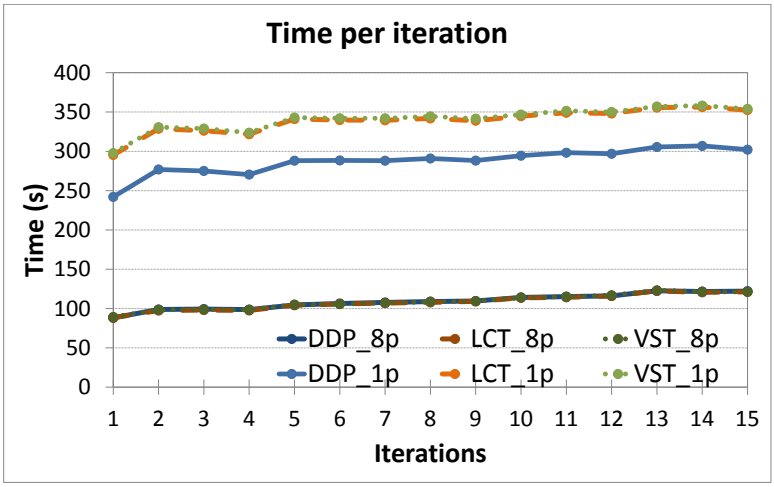


Figure 7.9: Time per iteration in study case 3 by using one and eight processors.

7.3.4 Results and analysis - Study case 4

Table 7.8 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds in the three methods assessed.

Noticing the the bounds among the methods showed to be consistent.

In the scenario tree of study case 4, the second stage is stochastic (with 20 scenarios) and the further stages are deterministic. Therefore there are 221 nodes

Table 7.8: Study case 4 - lower and upper bounds and convergence gap of the executions.

Method	Upper bound	Lower bound	Gap
DDP	12837714.5	12837830.8	0.0009054650
LCT	12837704.7	12837832.9	0.0009986051
VST	12837718.3	12837815.3	0.0007555267

performing forward passes and 200 performing backward passes.

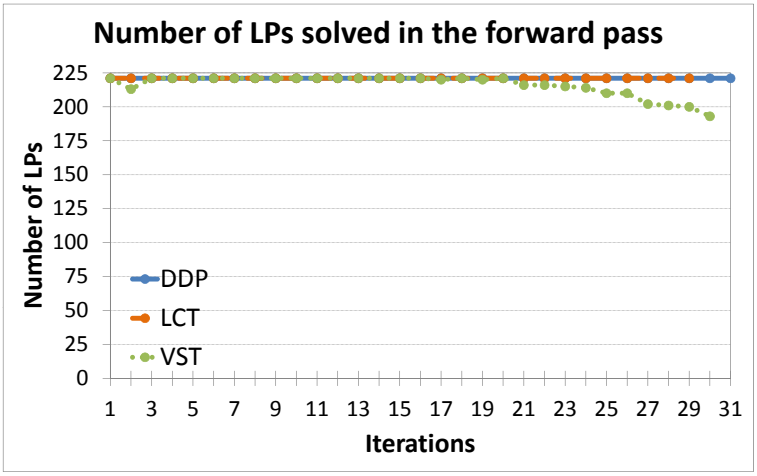


Figure 7.10: Number of LPs solved per iteration in the forward pass for the study case 4.

Figures 7.10 and 7.11 show the number of LPs solved in the forward and backward passes, respectively. We observe that LCT has an important impact on the reduction of the number of LPs solved in the backward pass. On the other hand, the impact caused by VST is less evident, even with the reduction being observed in both forward and backward passes. The consequence of these reductions can be noted in Figure 7.12, which shows the time of each iteration of the DDP algorithm.

We observe that the methods took different number of iterations to converge, which can be explained by the different paths in the convergence process of each approach. It is important to note that this is a random effect; thus the reduction in the number of iterations is not related directly to some strategy.

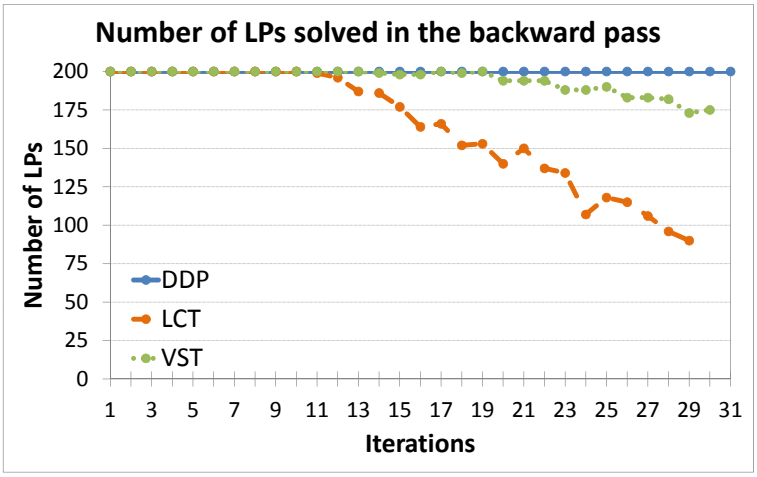


Figure 7.11: Number of LPs solved per iteration in the backward pass for the study case 4.

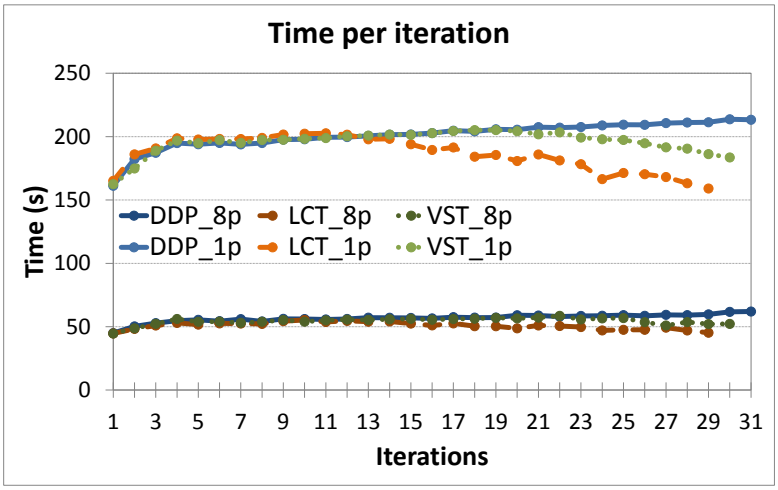


Figure 7.12: Time per iteration in study case 4 by using one and eight processors.

7.4 Assessment of parallelization: traditional and asynchronous DDP approaches

The performance of the presented algorithms was evaluated in a multistage linear stochastic optimization problem. We assess the parallel CPU time, speedup and efficiency using several amounts of cores for the following algorithms:

- traditional DDP parallelization (section 6.1);

- the proposed ADDP approach, as described in section 6.2;
- the alternative PADDP framework also proposed (6.3). The results showed correpond to the scenario approach once we verified that the efficiency are not too different compared to the node approach, we also observed that depending on the number of processors and the shape of the tree one approach is slightly more or less efficient than other.

7.4.1 Results and analysis - Study case 1

Table 7.9 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds of the DDP and ADDP, where the bounds do not vary with the number of processors and all bounds of the parallel executions of PADDP method.

Table 7.9: Study case 1 - lower and upper bounds and convergence gap of the parallel methods.

Method		**Upper bound**	**Lower bound**	**Gap**
PADDP	1p	11900724.0	11900815.5	0.0007686771
	12p	11900714.7	11900817.8	0.0008668367
	24p	11900707.2	11900806.0	0.0008300325
	36p	11900717.1	11900833.1	0.0009749642
	48p	11900703.3	11900815.1	0.0009393943
	60p	11900719.5	11900834.4	0.0009653238
	72p	11900719.5	11900834.4	0.0009653238
	84p	11900719.5	11900834.4	0.0009653238
	96p	11900719.5	11900834.4	0.0009653238
	108p	11900719.5	11900834.4	0.0009653238
	120p	11900719.5	11900834.4	0.0009653238
ADDP		11900707.0	11900819.5	0.0009452812
DDP		11900728.6	11900828.5	0.0008389382

It is important to notice that for the three methods the solution bounds are consistent with the fact that the problems been solved are the same as well as the solution.

Table 7.10 shows the necessary CPU time to solve case 1 by the three strategies and varying the number of processors. Table 7.11 shows the number of steps for convergence of the ADDP and PADDP approaches, as compared to the number of iterations for the traditional DDP algorithm. We note that a PADDP or ADDP step consists in solving all nodes a single time and a DDP iteration consists in performing both a forward pass (solving all nodes once) and a backward pass, solving again all nodes except the leaf ones.

Based on the CPU time and number of iterations/steps, we can make the following comments:

Table 7.10: Study case 1 - time consumption (in seconds) varying the number of processors.

Number of Processors										
1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP t(s):										
1897	337	249	177	110	85	84	73	65	71	70
ADDP t(s):										
3563	408	302	175	147	135	103	104	101	93	89
DDP t(s):										
1845	655	517	347	358	317	321	321	326	323	327

Table 7.11: Study case 1 - number of steps/iterations until convergence

	Number of Processors										
	1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP steps:	24	30	35	34	35	38	38	38	35	39	40
ADDP steps:	**44 in all cases**										
DDP iterations:	**18 in all cases**										

- The DDP algorithm has a serial (1 processor) CPU time of 1845 seconds (approximately 30 minutes), which is reduced to approximately 5 minutes with the increase in the number of processors. However, we note a saturation of the CPU time curve when the number of processors is greater than 60. This saturation was expected because the scenario tree has a maximum number of 64 independent leaf nodes.

- The time of the serial execution of the ADDP (around 60 minutes) was twice the time of the traditional DDP approach. Nevertheless, in the parallel environment with 12 processors, the CPU time of ADDP is 60% of the total DDP time. Moreover, the time of ADDP drops to 89 seconds (1 minute and half) with 120 processors.

- On the other hand, PADDP has comparable CPU time with DDP with one processor, and smaller CPU times with 12 or more processors. Moreover, this method had a better performance than the ADDP algorithm.

- The DDP approach takes 18 iterations until convergence, while the ADDP takes 44 steps, which justifies the greater CPU time of the latter in the serial run. Nevertheless, even proceeding 44 steps, the parallel CPU time for the ADDP approach is smaller, since the asynchronous approach is more suitable for parallel environments.

- The number of steps needed to perform a complete DDP iteration (forward and backward passes) depends on the size of the tree. Since this case has 7 periods, a complete iteration takes 12 steps because the initial state requires 7

steps to propagate the state variables information until the end of the scenario tree, plus 5 more steps to propagate cuts until the first node. As a result, 44 steps of the ADDP algorithm represented only about 4 DDP iterations. This shows that the extra information appended in the steps indeed contributes for faster convergence of the method.

- Regarding the ADDP method, the number of steps until convergence in Table 7.11 decreases when a smaller number of processors is used, which is expected since the convergence rate increases as the nodes share the same processor. As a consequence, for identical input data, the PADDP convergence results may differ with the number of processors, which may be an issue in official uses of this approach for market pricing, since the Independent System Operator (ISO) may require that exactly the same results should be obtained regardless of the number of processors.

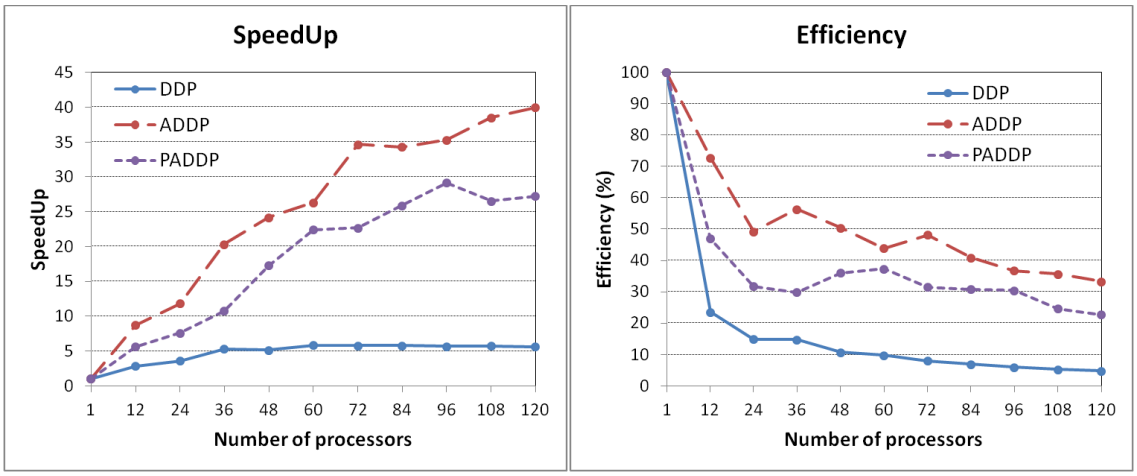


Figure 7.13: Study case 1 - speedup and efficiency varying the number of processors

The speedup of a given parallel algorithm is defined by the ratio between the time speed with 1 and N processors ($speedup(N) = t_1/t_N$), and the ideal speedup occurs when $t_N = t_1/N$ or $speedup(N) = N$. The parallel efficiency of the algorithm may be defined as the speedup divided by the number of processors that were employed:

$$\eta_{\%}(N) = 100 \times \frac{t_1}{N \times t_N}.$$

Figure 7.13 shows the computed speedup and efficiency for study case 1. We observe that both the ADDP and PADDP approaches have better speedup than DDP for any number of processors. Since the number of steps of the PADDP method increases with the number of processors, the speedup is reduced. This reduction causes the ADDP to have better speedup than the PADDP, even if the PADDP spends less computational time.

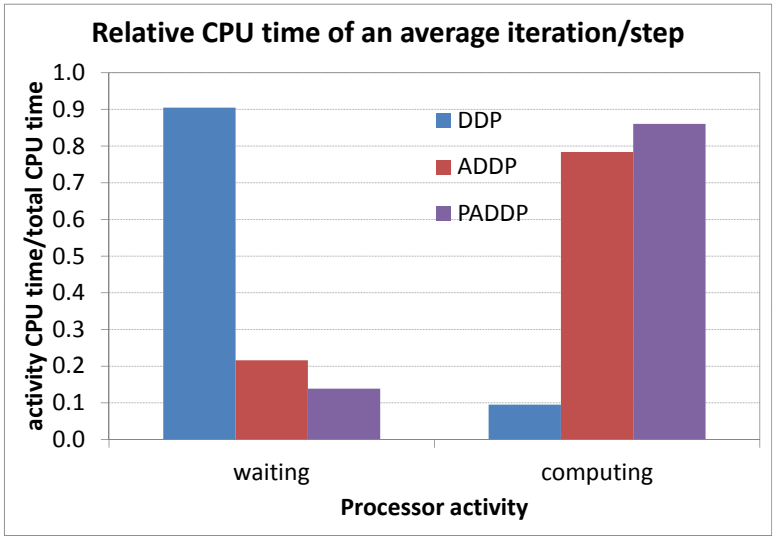


Figure 7.14: Study case 1 - average CPU time of the processor activities per iteration/step.

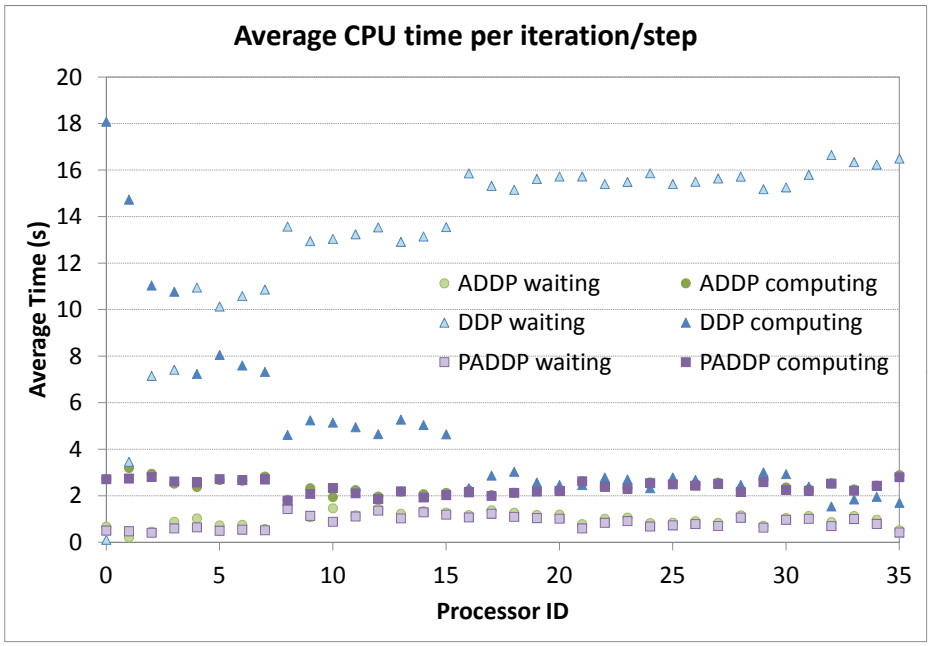


Figure 7.15: Study case 1 - average CPU time per iteration/step in each processor.

In order to measure the CPU times spent by each processor during the solution process, we divide the processor work into two main activities:

- *Waiting*: is the CPU time a processor spends either sending/receiving mes-

sages from other processors or waiting into a synchronization point;

- *Computing*: is the CPU time spent in all calculations of the algorithm flow, including data processing, solving or updating of LPs, as well as computation of costs or convergence tests. We note that 99% of the computing time is spent solving LPs.

We process the study case 1 with 36 processors and Figure 7.14 shows the ratio between the waiting activity and the computing activity per-unity of the total CPU time, in average among all processors. We observe that in DDP method the processors spend 90% of the time in average waiting and only 10% computing, in contrast, the processors using the ADDP and PADDP methods spend around 80% proceeding calculations. Figure 7.15 shows the average CPU time per iteration for each processor individually. As expected, the average iteration time of DDP method is higher than the average step time of the ADDP or PADDP method since the steps are faster to process than the iterations. We note that DDP algorithm has an unbalanced distribution where some processors spend a larger time computing while others spend a larger time waiting, which occurs because of the stage dependencies of the method. On the other hand, the processors work more homogeneously in the ADDP and PADDP algorithms, we also note that the balance in this two methods are quite similar.

7.4.2 Results and analysis - Study case 2

Table 7.12 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds of the DDP and ADDP, where the bounds do not vary with the number of processors and all bounds of the parallel executions of PADDP method.

It is important to notice that for the three methods the solution bounds are consistent with the fact that the problems been solved are the same as well as the solution.

Table 7.13 shows the CPU time to solve study case 2 and Table 7.14 shows the number of steps/iterations for convergence of the ADDP, PADDP and DDP methods. Based on these results, we note that:

- The DDP algorithm has a serial CPU time of 9536 seconds (approximately 2 hours and 40 minutes), which is decreased to about 8 minutes with the increase in the number of processors.

- The CPU time of the serial run of the ADDP algorithm (5 hours and 18 minutes) is twice the CPU time of the DDP method. Nevertheless, the CPU

Table 7.12: Study case 2 - lower and upper bounds and convergence gap of the parallel methods.

Method		Upper bound	Lower bound	Gap
PADDP	1p	11765779.9	11765887.3	0.0009127211
	12p	11765772.1	11765880.8	0.0009241311
	24p	11765794.8	11765892.0	0.0008266657
	36p	11765799.7	11765910.2	0.0009386515
	48p	11765777.4	11765883.5	0.0009012106
	60p	11765788.9	11765903.3	0.0009724866
	72p	11765796.1	11765898.1	0.0008666923
	84p	11765816.2	11765924.5	0.0009210862
	96p	11765780.7	11765895.2	0.0009730531
	108p	11765780.3	11765892.4	0.0009529356
	120p	11765789.8	11765885.6	0.0008142677
ADDP		11765789.2	11765889.3	0.0008510270
DDP		11765793.1	11765899.3	0.0009027470

Table 7.13: Study case 2 - time consumption (in seconds) varying the number of processors.

Number of Processors										
1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP t(s):										
9944	1432	825	533	431	373	313	268	243	233	236
ADDP t(s):										
19042	2017	1102	752	578	470	408	363	310	290	272
DDP t(s):										
9536	2727	1676	1164	925	817	590	526	514	519	504

Table 7.14: Study case 2 - number of steps/iterations until convergence

	Number of Processors									
	1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP steps:	20	26	29	28	29	31	31	29	30	31	33
ADDP steps:	**37 in all cases**										
DDP iterations:	**16 in all cases**										

time of the ADDP, with 12 processors, is 70% of the CPU time for the DDP approach. Also, the CPU time of the ADDP with 120 processors drops to 272 seconds (4 minute and half), which is 50% of the CPU time of DDP. We note that such drop happens faster in the ADDP scheme because the CPU time with 60 processors is already smaller than the CPU time of the DDP with 120 processors.

- As in case 1, the serial PADDP algorithm has comparable time with the DDP approach and, for 12 or more processors, the CPU time of PADDP is always less than both DDP and ADDP algorithms.

- While the DDP takes 16 iterations until convergence, the ADDP takes 37

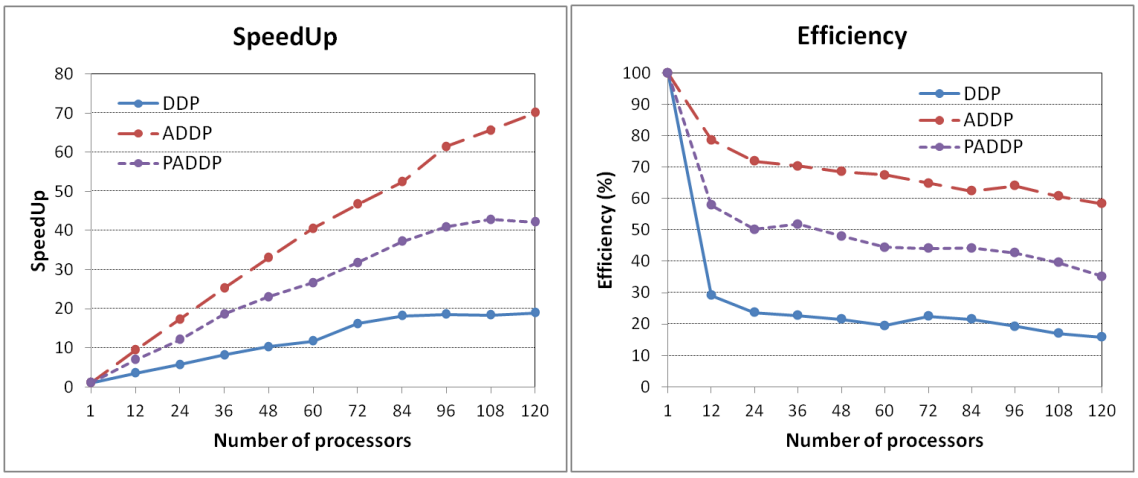


Figure 7.16: Study case 2 - speedup and efficiency with different numbers of processors.

steps. This explains the difference in the serial CPU time and reinforces the parallel suitability of the asynchronous algorithm.

- We note that the number of steps until convergence for the ADDP method tends to increase with the number of processors. However, we note that from 24 to 36 processors and from 72 to 84 processors, the number of steps decreases. This may occur due to the distribution of the nodes among the processors: the convergence rate may differ depending on the nodes that share the same processor.

Figure 7.16 shows the computed speedup and efficiency for study case 2. We observe that both ADDP and PADDP approaches have a more satisfactory speedup than the DDP method for all numbers of processors. While the DDP efficiency is lower than 30%, the ADDP efficiency is at least 60%.

We process the study case 2 with 36 processors and Figure 7.17 shows the ratio between the waiting activity and the computing activity per-unity of the total CPU time, in average among the processors. In this case the ratios were better than the previous study case for the three methods, although the DDP method still has the waiting CPU times higher than the computing CPU times. Figure 7.18 shows the average CPU time of each processor.

7.4.3 Results and analysis - Study case 3

Table 7.15 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds of the DDP and ADDP, where the bounds do not vary with the number of processors and all bounds of the parallel executions of PADDP method.

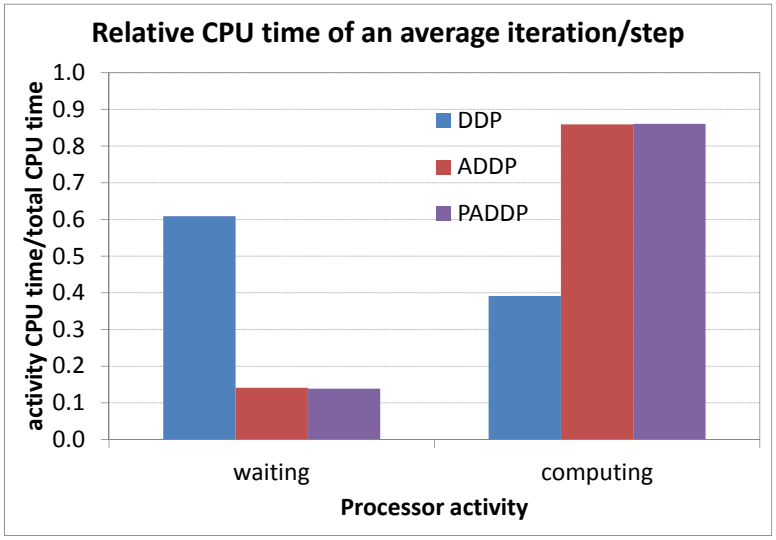


Figure 7.17: Study case 2 - average CPU time of the processor activities per iteration/step.

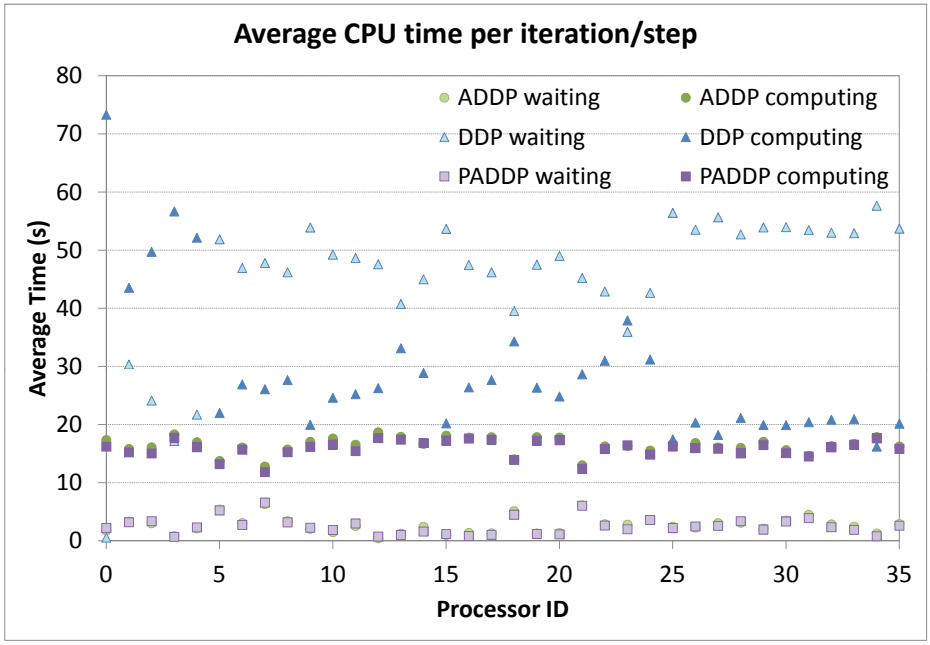


Figure 7.18: Study case 2 - average CPU time per iteration/step in each processor.

It is important to notice that for the three methods the solution bounds are consistent with the fact that the problems been solved are the same as well as the solution.

Table 7.15: Study case 3 - lower and upper bounds and convergence gap of the parallel methods.

Method		**Upper bound**	**Lower bound**	**Gap**
PADDP	1p	12812796.3	12812908.2	0.0008731928
	12p	12812778.7	12812904.4	0.0009816727
	24p	12812772.8	12812882.1	0.0008532467
	36p	12812793.1	12812885.5	0.0007214608
	48p	12812782.6	12812897.7	0.0008978335
	60p	12812770.2	12812896.5	0.0009856747
	72p	12812766.7	12812894.2	0.0009952550
	84p	12812772.3	12812899.9	0.0009962923
	96p	12812772.3	12812899.9	0.0009962923
	108p	12812775.1	12812901.8	0.0009885657
	120p	12812775.1	12812901.8	0.0009885657
ADDP		12812793.0	12812888.9	0.0007486135
DDP		12812793.4	12812918.6	0.0009769356

Table 7.16 shows the CPU time for study case 3 and Table 7.17 shows the number of steps/iterations for convergence of the ADDP, PADDP and DDP methods. Figures 7.19a and 7.19b show the computed speedup and efficiency for study case 3.

Table 7.16: Study case 3 - time consumption (in seconds) varying the number of processors.

Number of Processors										
1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP t(s):										
6266	998	724	473	576	429	706	492	682	466	475
ADDP t(s):										
13075	2053	1397	1293	1222	1175	1116	1088	1054	1060	1037
DDP t(s):										
5132	1211	804	643	562	506	509	478	476	452	428

Table 7.17: Study case 3 - number of steps/iterations until convergence

	Number of Processors									
	1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP steps:	24	28	30	26	30	27	37	30	36	30	30
ADDP steps:	**45 in all cases**										
DDP iterations:	**15 in all cases**										

We note that:

- The DDP algorithm has a serial CPU time of 5132 seconds (approximately 1 hours and half), which is decreased to about 7 minutes with the increase in the number of processors.

- The serial CPU time of the ADDP algorithm (around 3 hours and 40 minutes minutes) is more than twice the CPU time of DDP, since the number of ADDP steps is three times greater than the number of DDP iterations. Although the speedup of ADDP is still better than DDP, it is not sufficient to compensate the global CPU time for a parallel environment.

- By contrast, the serial run of the PADDP algorithm has comparable time with the DDP one, since the number of steps until convergence is smaller.

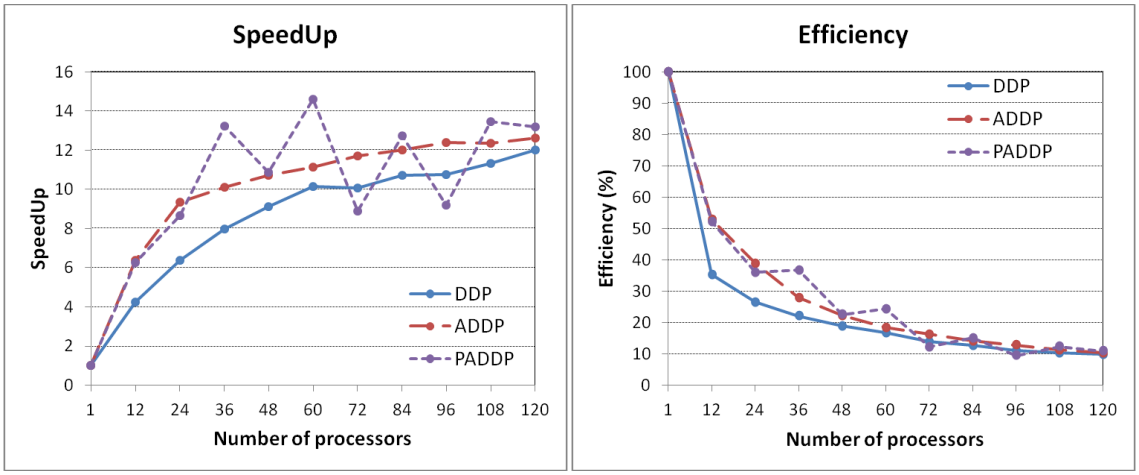


Figure 7.19: Study case 3 - speedup and efficiency with different numbers of processors.

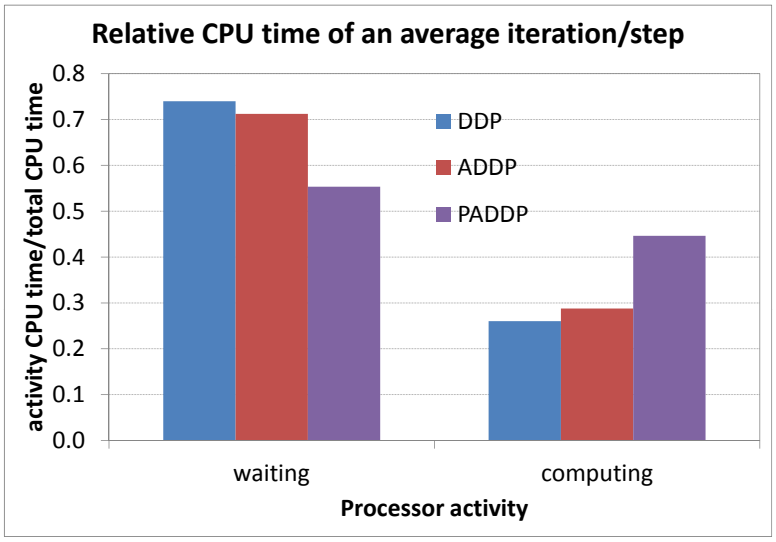


Figure 7.20: Study case 3 - average CPU time of the processor activities per iteration/step.

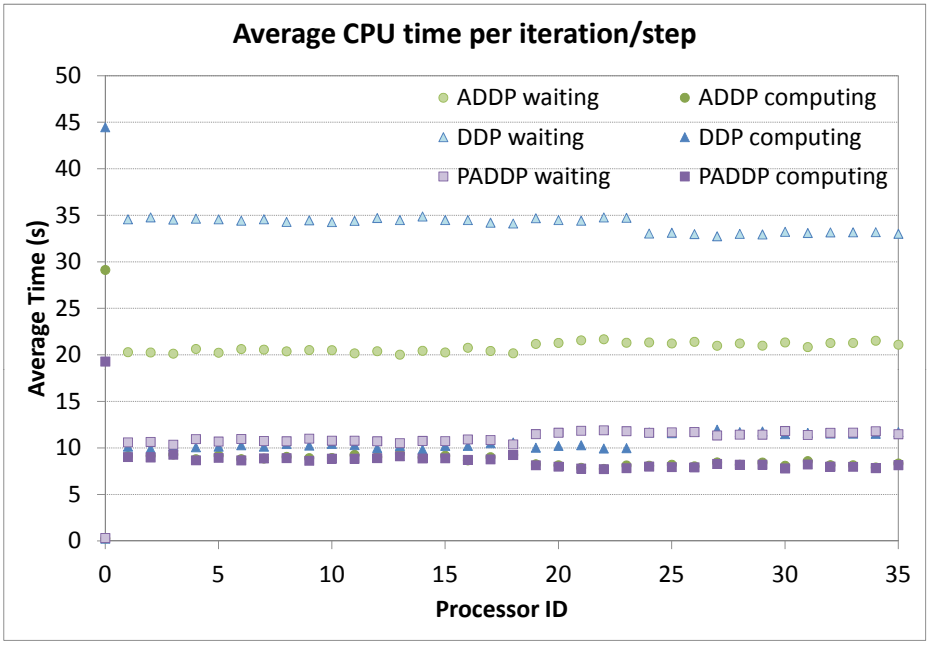


Figure 7.21: Study case 3 - average CPU time per iteration/step in each processor.

Figure 7.19 shows the computed speedup and efficiency for study case 3. The PADDP presented irregular behavior because of the changes on the number of steps to convergence. For both measures the ADDP presented slightly more satisfactory numbers than DDP.

Figure 7.20 and Figure 7.21 show the waiting and computing activities CPU times of processors, in average and individually. Due to the deterministic part of the scenario tree, which causes the nodes to be very dependent from each other, we observe that waiting CPU times are greater than computing in all methods. It is possible to note the overload of the processor that performs that handles this part of the tree (id 0) while the others are idle most part of the time.

7.4.4 Results and analysis - Study case 4

Table 7.18 shows the final bounds on the convergence process and the convergence gap calculated as described in section 4.3. We show the bounds of the DDP and ADDP, where the bounds do not vary with the number of processors and all bounds of the parallel executions of PADDP method.

It is important to notice that for the three methods the solution bounds are consistent with the fact that the problems been solved are the same as well as the solution.

Table 7.18: Study case 4 - lower and upper bounds and convergence gap of the parallel methods.

Method		Upper bound	Lower bound	Gap
PADDP	1p	12837698.5	12837823.1	0.0009704447
	12p	12837707.6	12837818.0	0.0008599612
	24p	12837700.2	12837825.0	0.0009725472
	36p	12837699.5	12837818.1	0.0009236725
	48p	12837706.0	12837821.1	0.0008968712
	60p	12837712.6	12837810.1	0.0007594435
	72p	12837701.0	12837826.3	0.0009762634
	84p	12837700.7	12837827.9	0.0009901938
	96p	12837704.3	12837829.2	0.0009732802
	108p	12837705.8	12837824.8	0.0009265179
	120p	12837696.8	12837819.3	0.0009539291
ADDP		12837693.5	12837814.9	0.0009455434
DDP		12837714.5	12837830.8	0.0009054650

Table 7.19 shows the CPU time for study case 4 and Table 7.20 shows the number of steps/iterations for convergence of the ADDP, PADDP and DDP methods. For this case, we note that:

Table 7.19: Study case 4 - time consumption (in seconds) varying the number of processors.

Number of Processors										
1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP t(s):										
6566	641	400	320	270	274	247	208	221	202	171
ADDP t(s):										
10652	1143	806	581	480	437	409	350	357	348	282
DDP t(s):										
6179	1024	725	722	748	730	715	735	725	736	735

Table 7.20: Study case 4 - number of steps/iterations until convergence

	Number of Processors										
	1	**12**	**24**	**36**	**48**	**60**	**72**	**84**	**96**	**108**	**120**
PADDP steps:	50	51	52	54	57	57	60	60	64	64	64
ADDP steps:	**82 in all cases**										
DDP iterations:	**31 in all cases**										

- The CPU time of the serial run of the DDP algorithm is 6179 seconds (approximately 1 hours and 40 minutes) and decreases to about 12 minutes with the increase in the number of processors. However, we observe a saturation of the CPU time curve with 24 processors, which is expected since the shape of the tree allows only 20 independent nodes to be solved at the same time.

- In contrast, the ADDP serial time is approximately 3 hours. Nevertheless, in the parallel environment this difference soon vanishes, and the CPU time of ADDP and DDP algorithms are similar with 12 processors. The CPU time of ADDP with 120 processors drops to 282 second (5 minutes), which is 40% of the DDP CPU time for the same number of processors.

- The PADDP algorithm has a serial time slightly greater than the DDP method. However, the CPU time of PADDP using 12 processors is smaller that the CPU time of DDP, even when the latter uses 120 processors.

- PADDP and ADDP algorithms are shown to be much more suitable for parallel environments as compared to DDP approach, since they lead to much smaller CPU times even when performing a much larger number of steps than DDP iterations.

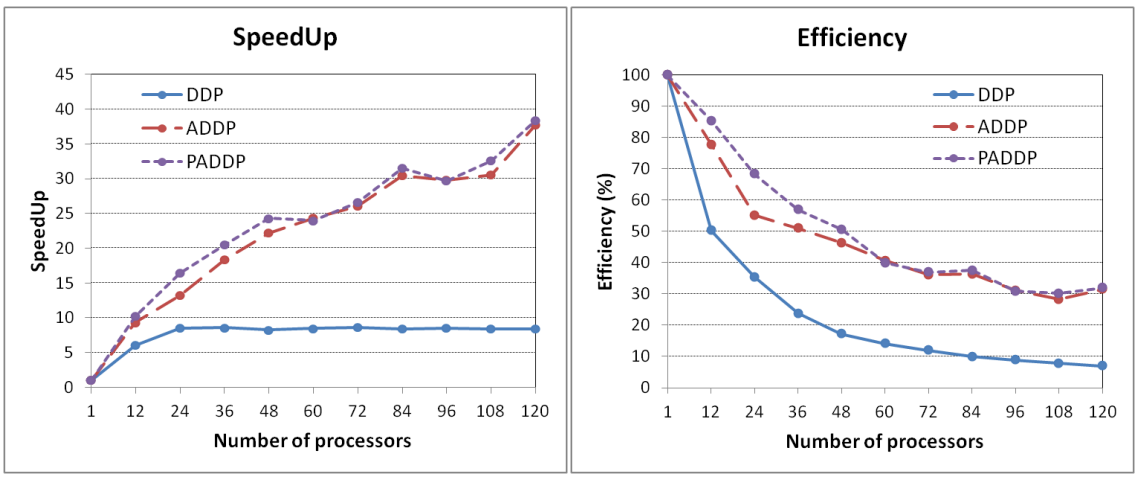


Figure 7.22: Study case 4 - speedup and efficiency with different numbers of processors.

Figure 7.22 shows the computed speedup and efficiency for study case 4. Differently from the other cases, the PADDP presented a similar performance to the ADDP, considering speedup and efficiency.

Figure 7.23 and Figure 7.24 show the waiting and computing activities CPU times of processors, in average and individually. The individual activities show that DDP traditional parallel scheme could not use all 36 processors since the maximum number of scenarios are 20 for this study case, also we note that processor 0 is overloaded because of the synchronization point in the beginning of the scenario tree. On the other hand, the PADDP and ADDP algorithms were able to better distribute the effort among all processors.

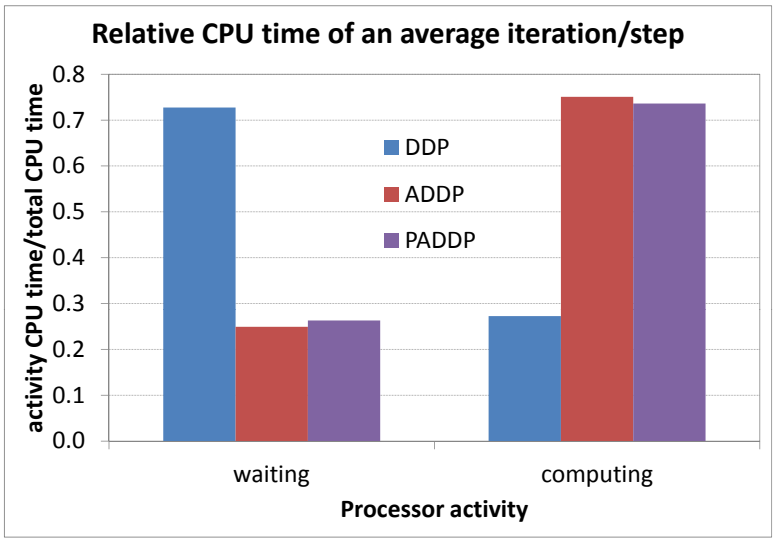


Figure 7.23: Study case 4 - average CPU time of the processor activities per iteration/step.

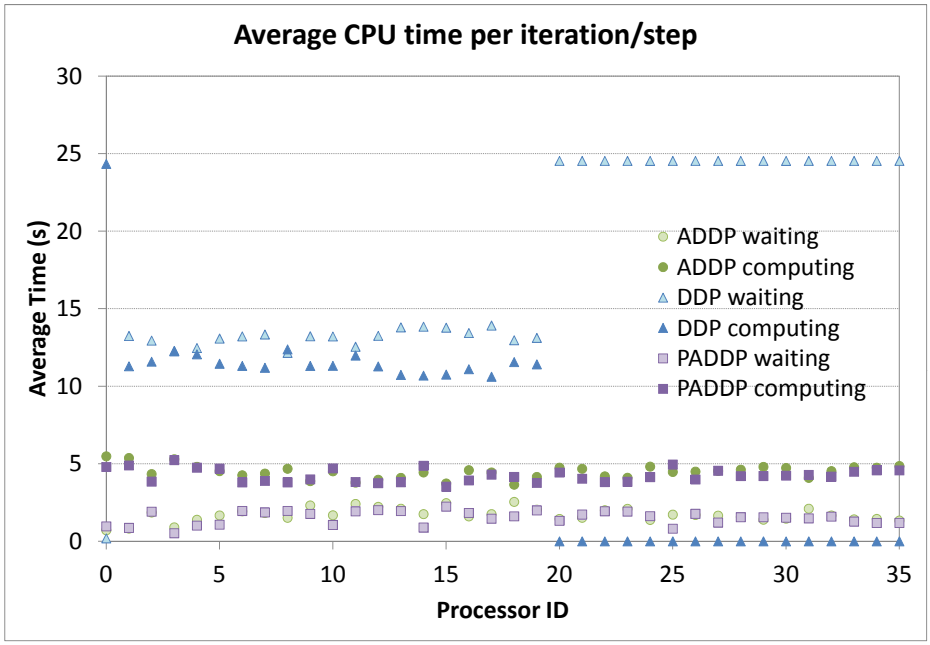


Figure 7.24: Study case 4 - average CPU time per iteration/step in each processor.

Chapter 8

Conclusion

The optimal power energy planning and dispatch of large hydrothermal systems is a challenging task in both modeling and solving aspects. When using a complex model, it is possible to represent more precisely the many features of the system, however this poses difficulties to solve the problem. Decomposition methods are commonly used to solve large problems, since they are able to find global solution by handling smaller subproblems in an iterative way.

Dual Dynamic Programming (DDP) is a method for solving multistage stochastic optimization problems that is widely used in the literature for hydrothermal coordination problems. However, depending on the size of the scenario tree and on the complexity of the optimization subproblems, the DDP algorithm may take a large time until reach convergence, which encourages the development of strategies to accelerate it.

In this dissertation we review some strategies that have been proposed in the literature with the objective of reducing the time of the DDP method. We also propose alternative strategies to accelerate the DDP solution process and evaluate these algorithms by solving four study cases related to the large scale hydrothermal coordination (HTC) problem of the Brazilian system, with different features of the scenario tree. The proposed strategies are divided in two approaches:

- **Reducing the amount of calculation of the DDP algorithm:** We propose two strategies - called local convergence test (LCT) and state variables stability test (VST) - which are capable of reducing the iteration time of the solution process by avoiding unnecessary operations. We show that the application of both strategies yields a reduction in the computation burden of the DDP algorithm, mainly in the final stages of the convergece process. However, these strategies can be more or less effective depending on the size and shape of the scenario tree.

- **Parallel approach applied to the DDP algorithm:** We propose two asyn-

chronous versions of the Dual Dynamic Programming (DDP) algorithm, which are much more suitable for parallel environments as compared to the traditional algorithm, without losing its convergence properties. The main idea is to perform totally asynchronous steps, with respect to solving the linear programming subproblems at each node, instead of using the classical iterations with forward and backward passes. The first asynchronous DDP algorithm (ADDP) allows homogeneous granularity and full node-wise parallelization within each step. In the partial version of this algorithm (PADDP) a certain synchronism is introduced, as convenient, in order to obtain a better convergence rate. Both ADDP and PADDP approaches presented better speedup and efficiency as compared to the traditional DDP algorithm.

Table 8 summarizes the proposed strategies by showing the state of the art background and the main features of each method.

Table 8.1: Summary of the proposed strategies

	Background	**What is new?**	**Effect**
LCT	[45]	Local convergence test of subtrees	- Reduces CPU time per iteration - Avoids backward passes in subtrees
VST	[45]	State variables stability test	- Reduces CPU time per iteration - Avoids forward passes in subtrees
ADDP	[35]	Asynchronous way of performing DDP	Reduces wall time in parallel environments
PADDP	[35]	ADDP with convenient synchronism	Reduces wall time in parallel environments

As future works, we aim to continue investigating the benefits of using ADDP instead of DDP method. Possible improvements in the new parallel approach can also be studied, such as: a better division of nodes between the processors, the use of a dynamic parallel alocation of the available processors, a more accurate analysis on the best way to traverse the tree, and also investigating more suitable initial conditions at each node in the ADDP approach, in order to increase the convergence rate. We can also combine ADDP with a cut selection technique, in order to avoid an excessive number of cuts in the linear programming subproblems (LPs). Finally, the proposed ADDP approach can also be extended to the sampling version of the DDP algorithm, known as stochastic dual dynamic programming (SDDP).

Bibliography

[1] ALTENSTEDT, F., 2003, *Aspects on asset liability management via stochastic programming*. Tese de Doutorado, Chalmers University of Technology and Gotebourg University.

[2] BENDERS, J. F., 1962, "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik*, v. 4, n. 1, pp. 238–252. ISSN: 0945-3245. doi: 10.1007/BF01386316. Disponível em: <http://dx.doi.org/10.1007/BF01386316>.

[3] BIRGE, J. R., 1985, "Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs", *Operations Research*, v. 33, n. 5, pp. 989–1007. doi: 10.1287/opre.33.5.989.

[4] BIRGE, J. R., LOUVEAUX, F. V., 1997, *Introduction to Stochastic Programming*. New York, Springer-Verlag. ISBN: 978-0-387-22618-7.

[5] BIRGE, J. R., LOUVEAUX, F. V., 1988, "A multicut algorithm for two-stage stochastic linear programs", *European Journal of Operational Research*, v. 34, n. 3, pp. 384 – 392. ISSN: 0377-2217. doi: http://dx.doi.org/10.1016/0377-2217(88)90159-2.

[6] BIRGE, J. R., DONOHUE, C. J., HOLMES, D. F., et al., 1996, "A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs", *Mathematical Programming*, v. 75, n. 2, pp. 327–352. ISSN: 1436-4646. doi: 10.1007/BF02592158. Disponível em: <http://dx.doi.org/10.1007/BF02592158>.

[7] DANTZIG, G. B., WOLFE, P., 1960, "Decomposition Principle for Linear Programs", *Oper. Res.*, v. 8, n. 1 (fev.), pp. 101–111. ISSN: 0030-364X. doi: 10.1287/opre.8.1.101. Disponível em: <http://dx.doi.org/10.1287/opre.8.1.101>.

[8] DE MATOS, V. L., PHILPOTT, A. B., FINARDI, E. C., 2015, "Improving the performance of Stochastic Dual Dynamic Programming", *Journal of Computational and Applied Mathematics*, v. 290, pp. 196 –

208. ISSN: 0377-0427. doi: http://dx.doi.org/10.1016/j.cam.2015.04.048. Disponível em: <//www.sciencedirect.com/science/article/pii/S0377042715002794>.

[9] DEMPSTER, M. A. H., THOMPSON, R. T., 1998, "Parallelization and aggregation of nested Benders decomposition", *Annals of Operations Research*, v. 81, n. 0, pp. 163–188. ISSN: 1572-9338. doi: 10.1023/A:1018996821817. Disponível em: <http://dx.doi.org/10.1023/A:1018996821817>.

[10] DINIZ, A. L., MACEIRA, M. E. P., 2008, "A Four-Dimensional Model of Hydro Generation for the Short-Term Hydrothermal Dispatch Problem Considering Head and Spillage Effects", *IEEE Transactions on Power Systems*, v. 23, n. 3 (Aug), pp. 1298–1308. ISSN: 0885-8950. doi: 10.1109/TPWRS.2008.922253.

[11] DINIZ, A. L., SANTOS, T. N., CABRAL, R., et al., 2018, "Short/mid-term hydrothermal dispatch and spot pricing for large-scale systems - the case of Brazil", Accepted for publication at the 20th Power System Computation Conference.

[12] DOS SANTOS, T. N., DINIZ, A. L., 2009, "A New Multiperiod Stage Definition for the Multistage Benders Decomposition Approach Applied to Hydrothermal Scheduling", *IEEE Transactions on Power Systems*, v. 24, n. 3 (Aug), pp. 1383–1392. ISSN: 0885-8950. doi: 10.1109/TPWRS.2009.2023265.

[13] ENNES, M. I. A., CABRAL, R. N., DINIZ, A. L., 2012, "Modelagem linear por partes dinâmica para a estratégia de programação dinâmica dual aplicada ao problema de planejamento hidrotérmico não linear estocástico". In: *Proc. of the XII Symposium of Specialists in Electric Operational and Expansion Planning*, p. 6, May.

[14] FISHER, M. L., 2004, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Manage. Sci.*, v. 50, n. 12 Supplement (dez.), pp. 1861–1871. ISSN: 0025-1909. doi: 10.1287/mnsc.1040.0263. Disponível em: <http://dx.doi.org/10.1287/mnsc.1040.0263>.

[15] FOSSO, O., GJELSVIK, A., HAUGSTAD, A., et al., 1999, "Generation scheduling in a deregulated system. The Norwegian case - Discussion", v. 14 (03), pp. 75 – 81.

[16] FRANGIONI, A., 2005, "About Lagrangian Methods in Integer Optimization", *Annals of Operations Research*, v. 139, n. 1, pp. 163–193. ISSN: 1572-9338.

doi: 10.1007/s10479-005-3447-9. Disponível em: <http://dx.doi.org/10.1007/s10479-005-3447-9>.

[17] GASSMANN, H. I., 1990, "Mslip: A computer code for the multistage stochastic linear programming problem", *Mathematical Programming*, v. 47, n. 1, pp. 407–423. ISSN: 1436-4646. doi: 10.1007/BF01580872. Disponível em: <http://dx.doi.org/10.1007/BF01580872>.

[18] GRÖWE, N., RÖMISCH, W., SCHULTZ, R., 1995, "A simple recourse model for power dispatch under uncertain demand", *Annals of Operations Research*, v. 59, n. 1 (Dec), pp. 135–164. ISSN: 1572-9338. doi: 10.1007/BF02031746. Disponível em: <https://doi.org/10.1007/BF02031746>.

[19] GUIGNARD, M., KIM, S., 1987, "Lagrangean decomposition: A model yielding stronger lagrangean bounds", *Mathematical Programming*, v. 39, n. 2, pp. 215–228. ISSN: 1436-4646. doi: 10.1007/BF02592954. Disponível em: <http://dx.doi.org/10.1007/BF02592954>.

[20] JACOBS, J., FREEMAN, G., GRYGIER, J., et al., 1995, "SOCRATES: A system for scheduling hydroelectric generation under uncertainty", *Annals of Operations Research*, v. 59, n. 1 (Dec), pp. 99–133. ISSN: 1572-9338. doi: 10.1007/BF02031745. Disponível em: <https://doi.org/10.1007/BF02031745>.

[21] JARDIM, D. L. D. D., MACEIRA, M. E. P., FALCAO, D. M., 2001, "Stochastic streamflow model for hydroelectric systems using clustering techniques". In: *2001 IEEE Porto Power Tech Proceedings (Cat. No.01EX502)*, v. 3, p. 6. doi: 10.1109/PTC.2001.964916.

[22] LATORRE, J. M., CERISOLA, S., RAMOS, A., et al., 2008, "Analysis of stochastic problem decomposition algorithms in computational grids", *Annals of Operations Research*, v. 166, n. 1, pp. 355–373. ISSN: 1572-9338. doi: 10.1007/s10479-008-0476-1. Disponível em: <http://dx.doi.org/10.1007/s10479-008-0476-1>.

[23] LEMARÉCHAL, C., 2001, "Lagrangian Relaxation". In: Jünger, M., Naddef, D. (Eds.), *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions*, pp. 112–156, Berlin, Heidelberg, Springer Berlin Heidelberg. ISBN: 978-3-540-45586-8. doi: 10.1007/3-540-45586-8_4. Disponível em: <http://dx.doi.org/10.1007/3-540-45586-8_4>.

[24] LINDEROTH, J., WRIGHT, S., 2003, "Decomposition Algorithms for Stochastic Programming on a Computational Grid", *Computational Optimization and Applications*, v. 24, n. 2, pp. 207–250. ISSN: 1573-2894. doi: 10.1023/A:1021858008222. Disponível em: <http://dx.doi.org/10.1023/A:1021858008222>.

[25] MACEIRA, M., DUARTE, V., PENNA, D., et al., 2008, "Ten years of application of stochastic dual dynamic programming in official and agent studies in Brazil - Description of the NEWAVE program". In: *16th Power System Computation Conference*.

[26] MACEIRA, M., MARZANO, L., PENNA, D., et al., 2015, "Application of CVaR risk aversion approach in the expansion and operation planning and for setting the spot price in the Brazilian hydrothermal interconnected system", *International Journal of Electrical Power & Energy Systems*, v. 72, pp. 126 – 135. ISSN: 0142-0615. doi: https://doi.org/10.1016/j.ijepes.2015.02.025. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0142061515001088>. The Special Issue for 18th Power Systems Computation Conference.

[27] MAGNANTI, T. L., WONG, R. T., 1981, "Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria", *Operations Research*, v. 29, pp. 464–484.

[28] MERCIER, A., SOUMIS, F., 2007, "An integrated aircraft routing, crew scheduling and flight retiming model", *Computers and Operations Research*, v. 34, n. 8, pp. 2251 – 2265. ISSN: 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2005.09.001. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054805002911>.

[29] MORITSCH, H. W., PFLUG, G. C., SIOMAK, M., 2001, "Asynchronous nested optimization algorithms and their parallel implementation", *Wuhan University Journal of Natural Sciences*, v. 6, n. 1 (Mar), pp. 560–567. ISSN: 1993-4998. doi: 10.1007/BF03160302. Disponível em: <https://doi.org/10.1007/BF03160302>.

[30] MORTON, D. P., 1996, "An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling", *Annals of Operations Research*, v. 64, n. 1, pp. 211–235. ISSN: 1572-9338. doi: 10.1007/BF02187647. Disponível em: <http://dx.doi.org/10.1007/BF02187647>.

[31] OLIVEIRA, F., GROSSMANN, I., HAMACHER, S., 2014, "Accelerating Benders stochastic decomposition for the optimization under uncertainty of

the petroleum product supply chain", *Computers and Operations Research*, v. 49, pp. 47 – 58. ISSN: 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2014.03.021. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0305054814000768>.

[32] PAPADAKOS, N., 2008, "Practical enhancements to the Magnanti–Wong method", *Operations Research Letters*, v. 36, n. 4, pp. 444 – 449. ISSN: 0167-6377. doi: http://dx.doi.org/10.1016/j.orl.2008.01.005. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167637708000102>.

[33] PEREIRA, M. V. F., PINTO, L. M. V. G., 1991, "Multi-stage stochastic optimization applied to energy planning", *Mathematical Programming*, v. 52, n. 1, pp. 359–375. ISSN: 1436-4646. doi: 10.1007/BF01582895. Disponível em: <http://dx.doi.org/10.1007/BF01582895>.

[34] PHILPOTT, A., DE MATOS, V., FINARDI, E., 2013, "On solving multistage stochastic programs with coherent risk measures", *Operations Research*, v. 61, n. 4 (Feb), pp. 957–970.

[35] SANTOS, T. N., DINIZ, A. L., BORGES, C. L. T., 2017, "A New Nested Benders Decomposition Strategy for Parallel Processing Applied to the Hydrothermal Scheduling Problem", *IEEE Transactions on Smart Grid*, v. 8, n. 3 (May), pp. 1504–1512. ISSN: 1949-3053. doi: 10.1109/TSG.2016.2593402.

[36] SANTOSO, T., AHMED, S., GOETSCHALCKX, M., et al., 2005, "A stochastic programming approach for supply chain network design under uncertainty", *European Journal of Operational Research*, v. 167, n. 1, pp. 96 – 115. ISSN: 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2004.01.046. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221704002292>.

[37] SHAPIRO, A., TEKAYA, W., COSTA, J., et al., 2013, "Risk neutral and risk averse Stochastic Dual Dynamic Programming method", *European journal of operational research*, v. 224, n. 2 (Jan), pp. 375–391.

[38] SHAPIRO, A., TEKAYA, W., SOARES, M. P., et al., 2013, "Worst-Case-Expectation Approach to Optimization Under Uncertainty", *Operations Research*, v. 61, n. 6, pp. 1435–1449. doi: 10.1287/opre.2013.1229.

[39] SHAPIRO, A., DENTCHEVA, D., RUSZCZYNSKI, A., 2014, *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Philadel-

phia, PA, USA, Society for Industrial and Applied Mathematics. ISBN: 1611973422, 9781611973426.

[40] SHERALI, H. D., LUNDAY, B. J., 2013, "On generating maximal nondominated Benders cuts", *Annals of Operations Research*, v. 210, pp. 57–72.

[41] SLYKE, R. M. V., WETS, R., 1969, "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming", *SIAM Journal on Applied Mathematics*, v. 17, n. 4, pp. 638–663. doi: 10.1137/0117061.

[42] TRUKHANOV, S., NTAIMO, L., SCHAEFER, A., 2010, "Adaptive multicut aggregation for two-stage stochastic linear programs with recourse", *European Journal of Operational Research*, v. 206, n. 2, pp. 395 – 406. ISSN: 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.02.025. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221710001566>.

[43] WALKUP, D. W., WETS, R. J.-B., 1969, "Lifting projections of convex polyhedra." *Pacific J. Math.*, v. 28, n. 2, pp. 465–475. Disponível em: <http://projecteuclid.org/euclid.pjm/1102983467>.

[44] WETS, R. J.-B., 1984, "Large scale linear programming techniques in stochastic programming", *IIASA Working Paper*.

[45] WITTROCK, R. J., 1985, "Dual nested decomposition of staircase linear programs". In: Cottle, R. W. (Ed.), *Mathematical Programming Essays in Honor of George B. Dantzig Part I*, pp. 65–86, Berlin, Heidelberg, Springer Berlin Heidelberg. ISBN: 978-3-642-00919-8. doi: 10.1007/BFb0121043. Disponível em: <http://dx.doi.org/10.1007/BFb0121043>.

[46] WOLF, C., KOBERSTEIN, A., 2013, "Dynamic sequencing and cut consolidation for the parallel hybrid-cut nested L-shaped method", *European Journal of Operational Research*, v. 230, n. 1, pp. 143 – 156. ISSN: 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2013.04.017. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221713003159>.

[47] ZAKERI, G., PHILPOTT, A. B., RYAN, D. M., 1999, "Inexact Cuts in Benders Decomposition", *SIAM Journal on Optimization*, v. 10, n. 3, pp. 643–657. doi: 10.1137/S1052623497318700. Disponível em: <http://dx.doi.org/10.1137/S1052623497318700>.