Analysis of the Performance of Benders-based Distributed Decomposition Methods for Linear Stochastic Programming

Jesus M. Latorre, Rafael Palacios, Andres Ramos
Institute for Research in Technology
ICAI School of Engineering
Comillas Pontifical University
Alberto Aguilera 23, 28015 Madrid, Spain
(Jesus.Latorre@iit.upcomillas.es)

In stochastic programming, the consideration of uncertainty might lead to large scale problems. In these problems, computational resources might fall short of the requirements for solving these problems, especially concerning memory capacity. Decomposition techniques help to clear this obstacle, by producing a set of smaller subproblems at the cost of additional computation time for coordination due to the iterative of the resolution algorithm. However, one way to mitigate the effect of the increased computational time derived from the decomposition is to solve the subproblems in parallel or distributed systems. This paper compares the performance of two different Benders decomposition techniques for linear stochastic problems when executed in two different computational grids (one using single-core computers and the other using dual-core computers). Decomposition methods that create less task-dependency take better advantage of computer resources available in the grid, thus compensating a small increase in the overall computational weight. In particular, the proposed method called "complete-scenario decomposition" requires a larger total CPU time (the sum of all individual CPU times in the grid) compared to traditional Benders decomposition; however, the computational overhead is compensated by a better grid performance, so the proposed method yields shorter execution times as measured by the user.

Keywords: linear stochastic programming; grid computing; Benders decomposition

1 Introduction

The representation of uncertainty in stochastic programming can be done through the use of scenario trees. These scenario trees present a discretization of the underlying probability of the uncertain data of the problem. The explicit consideration of the full scenario tree in the problem is usually not practical, due to the large size of the resulting problem, the so-called deterministic equivalent.

Generally, the solution of linear programming problems can be improved by taking advantage of the structure found in the constraints matrix. Problems are usually formulated linking subsets of variables, resulting in a block structure like the one shown in figure 1. This figure depicts part of the equation matrix of a multiperiod deterministic hydrothermal scheduling problem (in fact, the first 8 periods of it), which is the deterministic version of the problem employed in the results section of this paper. Individual blocks (framed in the picture) represent the coefficients of the equations relating variables within each period, whereas the top block is the objective function that links all periods. These period blocks are scarcely linked among themselves by a small amount of equations that consider two consecutive periods (see the coefficients below boxes that link variables from one period with variables from the next period). This characteristic structure (called staircase or L-shaped) can be used by decomposition methods to solve each block separately. Furthermore, since scenario trees in stochastic programming involve replicating the matrix structure for each node of the tree, decomposition methods have been widely applied in this field with positive results.

Decomposition methods break a large scale problem into a set of smaller subproblems that may fit into the available resources, e.g. the memory of the computer employed. This decomposition comes at the cost of an iterative solution algorithm needed to coordinate the subproblem solutions in order to obtain the solution for the original complete problem.

In this point, distributed computing can be used to reduce total execution time experienced by the user. By spreading the computation of subproblems over a set of computers, the effect of the increased computation time can be mitigated. However, there are several reasons that entail limitations to the efficiency of the distributed execution, that need to be carefully examined when designing and implementing distributed algorithms:

- The most important limitations are task dependencies that imply unavoidable restrictions in the way in which subproblems can be distributed. The most extreme situation

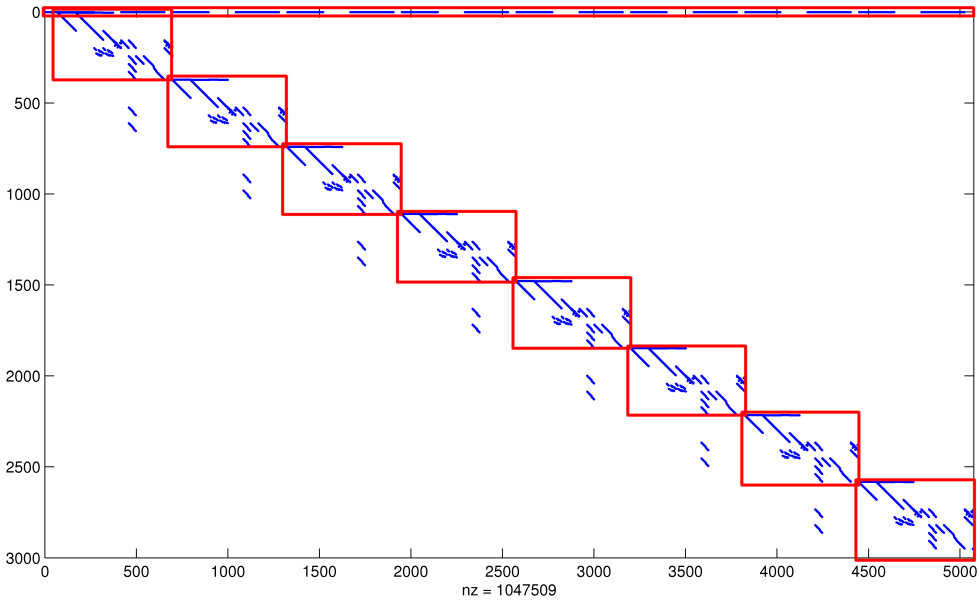

Figure 1: Structure of the equation matrix of a deterministic hydrothermal coordination problem.

would be a chain of dependencies that imposes sequential execution, regardless of the number of computers available in the grid system.

- Task distribution also produces some overhead time due to sending input data and retrieving results. Network latency, bandwidth, and the balance between data volume and computing time, are characteristics that play a role in this aspect.

- Finally, non-deterministic execution times, typical in optimization algorithms, produce load unbalancing in a more important and unpredictable fashion that unbalances produced by heterogeneous computer-power characteristic in the elements of grid systems.

Grid environments are Multiple Instruction Multiple Data (MIMD) distributed systems comprising large amounts of heterogeneous resources that act as a virtual system providing high computing power and storage capacity, not fully controlled by the user (Foster and Kesselman 1998). This paper focuses on employing this kind of systems for the solution of stochastic programming problems.

This paper compares two decomposition methods when solved in a grid environment. The partial-scenario decomposition is a nested Benders decomposition, in which stage ag-

gregation is applied to increase the grain and obtain better distributed performance. The second decomposition method, namely the complete-scenario decomposition, is proposed in this paper and it is based on Benders decomposition break subproblem linkages at all the stages. This makes the subproblems independently solvable without the need of the serial solution of a master problem, as in the usual Benders decomposition and Lagrangian relaxation methods. Complete-scenario decomposition results in subproblems of homogeneous size, and by eliminating subproblem dependencies, it increases the performance of the grid environment. Since this method also means a higher computational load, it is not worthwhile for application on a single computer. Nevertheless, due to its increased distributed performance, it might compensate the increased overall CPU time, as it will be shown later.

This paper is organized as follows: next section presents a review of previous works found in the literature concerning stochastic programming and parallel computing; then, in section 3 the two decomposition methods used in this paper are described, along with their formulation; follows section 4, where a real case study is used to evaluate the performance of both methods running on a prototype grid; and finally, section 5 presents conclusions that can be extracted related to the work carried out.

2 Related work

When decomposing large-scale linear programming problems, two main methods can be used: Benders decomposition and Lagrangian relaxation. They focus on eliminating respectively the complicating variables and the complicating equations, and thus can be seen as dual methods. Benders decomposition (Benders 1962) fixes complicating variables, which in stochastic problems (Slyke and Wets 1969) correspond to the stages of the problem common to several scenarios that link these scenarios together. Eliminating these variables allows solving each scenario independently. The resolution of independent subproblems can be then performed in parallel, increasing the overall performance of the method. Additionally, a master problem needs to be formulated, in order to solve the first stages and to propose values for their variables to the subproblems. The extension for multistage problems, where one stage acts as parent for the following stage and as child for the previous one, can be found in Birge (1985).

Lagrangian relaxation (Geoffrion 1970), on the other hand, eliminates the complicating constraints that link several blocks of variables in the problem, thus enabling each block to

be solved independently. In a stochastic problem, these complicating constraints are usually the non-anticipativity constraints that link those nodes common to several scenarios. To achieve convergence, a master problem is needed to coordinate the subproblems, enforcing relaxed equations to be fulfilled in the final solution.

Other decomposition methods can be thought as a derivation of these. For instance, the Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960) introduces a variation in the main Lagrangian relaxation scheme, solving the primal formulation of the master problem instead of its dual form. Another example is the cross decomposition (VanRoy 1983), which employs both decomposition schemes to profit from the block structure of the problem, both in variables and equations.

Generally, decomposition methods increase resolution times because they require iterating between master problem and subproblems in order to reach the solution. Usually, the only situation in which this approach pays off is when the original problem is so large that it cannot be solved directly in one computer. Hence decomposition makes it possible to solve such problems, but the total computing time will be larger than the expected execution time of the original problem. If the decomposition is carefully designed, resulting subproblems can be solved independently in parallel or distributed systems, thus mitigating the effect of increased computational time.

Previous work on parallel and distributed systems was initially focused on parallelizing basic mathematical programming methods. For instance, Forrest and Tomlin (1990) study both the simplex and interior point methods on SIMD systems, while Shu and Wu (1993) parallelizes key phases of the simplex method on a MIMD system, and Bisseling et al. (1993) and Karypis et al. (1994) perform similar tasks for interior point methods on MIMD machines. Later, branch and bound methods were solved by Iamnitchi and Foster (2000) on distributed systems. Also Anstreicher et al.(2002) and Goux and Leyffer (2002) solved nonlinear integer problems with branch and bound algorithms over grid systems.

Concerning stochastic programming methods, some work was derived from Rockafellar and Wets (1991) where the augmented Lagrangian is proposed as a way of accelerating convergence of Lagrangian relaxation at the cost of increasing problem complexity. Mulvey and Ruszczynski (1995) approximate the quadratic penalty term in the objective function (augmented lagrangian) that links all subproblems, so each problem is separable and can be solved in parallel on a MIMD system.

Regarding Benders decomposition, which lies at the core of the methods presented in this

paper, two-stage stochastic problems are considered in Nielsen and Zenios (1997). There, fixing the first stage solution allows the parallel solution of the multiple second stage subproblems. Since the structure of all second stages subproblems is similar, a SIMD machine is used to solve the second stages simultaneously.

Nodal decomposition is employed in Birge et al. (1996) following nested Benders decomposition in a serial mode, until a branch in the scenario tree is found. Then, the branching scenarios are assigned to other processors to be computed in parallel, following this same strategy. Dempster and Thompson (1998) improve the performance of this method by solving together in the same subproblem the contiguous stages that are not separated by any branching point.

In distributed and especially grid environments, asynchrony is the most valuable property in terms of performance. This is analyzed for linear problems in Moritsch et al.(2001) and Linderoth and Wright (2005), where only a fraction of the subproblems needs to be fully computed in order to proceed with the master problem resolution. Finally, Linderoth and Wight (2003) and Linderoth and Wright (2005) use a bundle trust region method to limit the range of variation of primal variables in each iteration, thus achieving faster convergence.

Ferris et al [2009] decomposes a scheduling problem that minimizes the makespan. The MIP problem is decomposed using GAMS/grid over Condor middleware, by assigning subsets of the search space to different machines further partitioning subsets that are found to be harder to solve.

3 Complete-scenario and partial-scenario decomposition methods

In stochastic programming, a decomposition method proposes a partition of the scenario tree nodes into subproblems, and the resolution method is highly influenced by this decision. This partitioning strategy also limits the attainable level of parallelization and thus the performance of each method in a parallel or distributed computing environment. For a decomposition method to have good performance in a grid, this partition has to fulfill some requirements:

- Due to the latency of the communication network, coarse grained decompositions should be chosen. This means that computation time for each subproblem should be increased, grouping nodes of the scenario tree, to compensate the communication

times. Besides, a smaller number of subproblems usually decreases the number of decomposition iterations.

- Subproblems should be as independent as possible to allow their parallel resolution without dependencies among them. For instance, if a subproblem needs to wait for another subproblem from a previous stage to start its own resolution, overall performance will suffer from this lack of parallelism.

- Additionally, if the iterative process starts with solutions that are closer to the optimal one, it seems natural that convergence will be easier. This implies that the formulation of subproblems that consider all stages will result in faster convergence. If this were not the case, first stage problems might propose solutions far from the optimal to the rest of subproblems, and even solutions that are infeasible when considering the whole time scope in the iterations when the recourse approximation is not built yet. This leads to a waste of time that can be avoided by including a complete scenario, comprised of at least one node in every stage.

In this section two decomposition schemes are presented, along with their mathematical formulation. In section 3.1, partial-scenario decomposition is briefly reviewed, and will be further considered as the reference method. In section 3.2, complete-scenario decomposition is presented, a method specifically designed to reach better performance in parallel and grid computing systems.

3.1 Partial-scenario decomposition

Benders decomposition, as it has already been stated, focuses on eliminating complicating variables in the problem, see Benders (1962) and Birge and Louveaux (1997). For this description, the following problem notation will be used

$$\begin{array}{ll} \min\limits_{x,y} & c^T x + d^T y \\ s.t. & Ax = b \\ & A'y = b' \\ & Tx + Wy = h \\ & x, y \geq 0 \end{array}$$

where x and y are the first and second stage variables. Fixing variables x, the resulting problem would be easily solved. In the stochastic case, where there are several second stages (one for each scenario) linked to the single first stage, fixing the latter allows independent

and parallel resolution of all second stage subproblems. This can be achieved by formulating the recourse problem (RP) which takes a fixed value of first stage variables x:

$$\begin{array}{rlll} \theta\left(x\right) = & \min\limits_{y} & d^T y & \\ & \text{s.t.} & A'y = b' & \text{(RP)} \\ & & Wy = h - Tx & : \pi \\ & & y \geq 0 & \end{array}$$

where π are the dual variables associated with the constraints linking the first and second stages.

Additionally, a first stage master problem (MP) has to be formulated, which will propose values of the first stages variables to reach the complete problem optimal solution:

$$\begin{array}{lll} \min\limits_{x,\theta} & c^T x + \theta & \\ & Ax = b & \text{(MP)} \\ & \delta^i \theta \geq \theta^i + {\pi^i}^T T\left(x^i - x\right) & \forall i \in I \\ & x \geq 0 & \end{array}$$

where θ is the recourse function, I is the set of already computed Benders cuts and $\delta^i \theta \geq \theta^i + {\pi^i}^T T\left(x^i - x\right)$ are the Benders cuts, being δ^i an indicator variable for feasibility ($\delta^i = 0$) (see Dantzig (1963)) or optimality ($\delta^i = 1$) cuts for iteration i, θ^i the value of the second stage objective function, and π^i the dual variables of the second stage constraints, all of them for the current value of the first stage variables x^i.

Alternative resolution of MP and RP builds the set of approximation cuts (indexed by i in MP) that drives the solution iteratively to the complete problem solution. This approach can be extended for multi-stage problems, where each problem acts both as a master problem for the following stage, and as a subproblem for the previous stage.

This general framework needs to be adapted for its use in distributed environments, in order to attain high grid performance and therefore to reduce overall execution time. This can be done by aggregating scenario tree nodes in the same subproblem, increasing subproblem size. A study of different node aggregation methods in the context of serial solution of a stochastic optimization problem can be found in Cerisola and Ramos (2000). In the present work aggregation is employed in a similar fashion to the method proposed in Dempster and Thompson (1998), but instead of stopping at the branching stages of the scenario tree, each subproblem will contain all the nodes of a scenario that are not assigned to the parent scenario. This increases the size of the subproblems, favoring the coarse grain of the algorithm. This method still suffers from the dependency among subtasks that lies in the

nature of the nested Benders decomposition, but it considers larger subproblems and even one full scenario in all the stages, benefiting the grain and the convergence characteristics of the method. Figure 2 shows an example of the partition of the nodes for a three-stage binary tree. It can be seen that with this decomposition there are as many subproblems as terminal nodes, and the subproblem sizes vary from as large as a node for each stage to as small as one single terminal node.

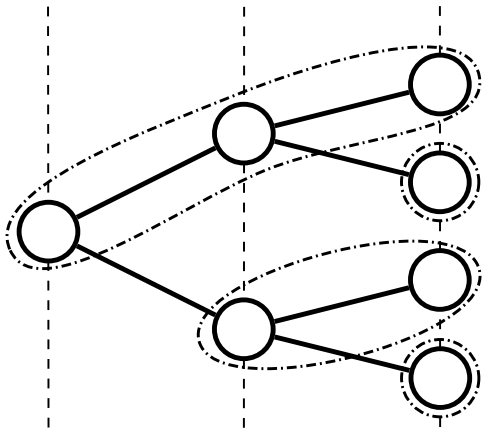

Figure 2: Partial-scenario decomposition example applied to a three stage binary tree.

Partial-scenario decomposition is a shorthand for referring to a nested Benders decomposition for multistage problems, which employs the already described scenario aggregation technique. To present the formulation of this decomposition, some notation will be introduced. The scenario tree is expressed as a set of scenarios $\left\{\omega^{\xi}\right\}$, which are divided into nodes $\left\{\omega_t^{\xi} \in \omega^{\xi}\right\}$, with t being the stage and ξ the uncertainty realization. A node ω_{t-1}^{ξ} is said to be predecessor of node ω_t^{ξ} if it belongs to the same scenario in the previous stage, and it is expressed as $pred\left(\omega_t^{\xi}\right) = \omega_{t-1}^{\xi}$. Hence, in this decomposition method a general expression of a subproblem for realization ξ at stage t of a multistage stochastic problem is the following:

$$
\begin{aligned}
\theta_{\overline{t}}^{\xi}\left(x_{\overline{t}-1}^{\xi}\right) = \min_{x_t^{\xi},\theta_{\underline{t}}^{\xi}} \quad & \sum_{(\xi,t)\in S} {c_t^{\xi}}^T x_t^{\xi} + \sum_{(\xi,\underline{t})\in C} \theta_{\underline{t}}^{\xi} & & \\
s.t. \quad & A_t^{\xi} x_t^{\xi} = b_t^{\xi} & & \forall\,(\xi,t) \in S \\
& T_t^{\xi} x_{t-1}^{\xi} + W_t^{\xi} x_t^{\xi} = h_t^{\xi} & & \forall\,(\xi,t) \in S, (\xi,t-1) \in S \quad (SP1) \\
& \delta_{\underline{t}}^{\xi i}\theta_{\underline{t}}^{\xi} \geq \theta_{\underline{t}}^{\xi i} + {\pi_{\underline{t}}^{\xi i}}^T T_{\underline{t}}^{\xi}\left(x_{\underline{t}-1}^{\xi i} - x_{\underline{t}-1}^{\xi}\right) & & \forall\,(\xi,\underline{t}) \in C, i \in I \\
& W_{\overline{t}}^{\xi} x_{\overline{t}}^{\xi} = h_{\overline{t}}^{\xi} - T_{\overline{t}}^{\xi} x_{\overline{t}-1}^{\xi} & & \forall\,(\xi,\overline{t}-1) \in P \\
& x_t^{\xi} \geq 0 & & \forall\,(\xi,t) \in S
\end{aligned}
$$

where I designates the set of previously computed cuts, $\overline{t}$ is the stage where scenario ω^{ξ} branches from its parent scenario, and $\underline{t}$ denotes the stages where children scenarios branch

from ω^{ξ}. For a full understanding of the previous expression, there are several sets that need to be defined, which determine the node partitioning of the stochastic problem into subproblems:

- The set of nodes that are included in each subproblem is:

$$S = \left\{ (\xi, t) \, / \omega_t^{\xi} \in \omega^{\xi}, \forall t \geq \overline{t} \right\}$$

- The set of parent nodes, or nodes not belonging to the subproblem which are predecessors of nodes in the subproblem, is:

$$P = \left\{ (\xi, t) : \omega_t^{\xi} \notin S, \omega_{t+1}^{\xi} \in S / \omega_t^{\xi} = pred\left(\omega_{t+1}^{\xi}\right) \right\}$$

- And the set of child nodes, or nodes not belonging to the subproblem whose parent do belong, is:

$$C = \left\{ (\xi, t) \, / \omega_t^{\xi} \notin S, pred\left(\omega_t^{\xi}\right) \in S \right\}$$

It should be noted that these subsets define the partitioning of this decomposition method, but also impose the order in which subproblems must be solved, that has to be followed to ensure that the information required for each subproblem is available when it is ready to be solved. The solution order for each iteration follows the forward and backward passes of the usual nested Benders decomposition method: it starts by solving the subproblem with no parent nodes, continues on with the subproblems having their parent nodes in this first subproblem, and so on, reversing the order for the backward pass when the Benders cuts are transmitted back. These cuts are computed from the current solution, which may not be the optimal one until the convergence of the complete problem is reached. The dependency between parent and child nodes restricts the performance of this method when executed in parallel, as it will be seen in the results section, because at any given moment there may be idle resources that cannot be used because there are subproblems waiting from other subproblems' solution.

3.2 Complete-scenario decomposition

With the previous decomposition scheme, the dependencies among subproblems limit the performance that can be achieved. It would be desirable to have a decomposition which formulates subproblems as independent as possible, so that all subproblems can be solved in

parallel without having idle computing servers. As it has been seen, the main disadvantage resides in breaking the links between scenario nodes, resulting in delays due to coordinating the solution of the fragments of each scenario. Thus, a better approach would be to consider complete scenarios for each subproblem.

In this section a new method is presented, the complete-scenario decomposition (Latorre et al. 2009), which enforces the independency among subproblems by including a complete scenario in each subproblem. Hence, in this method there are as many subproblems as terminal nodes, and all subproblems have the same size, each comprising as many nodes as the number of stages. An example of this decomposition can be seen in figure 3, applied to a three-stage binary tree: four subproblems are defined, each one comprising one complete scenario, from the first to the last stages. A side effect of this decomposition is that some of the nodes from the scenario tree (those that are common to several scenarios) are considered multiple times in different subproblems. As a consequence, coordination among scenarios is needed to reach the same solution in these duplicated nodes. This is usually performed by the non-anticipativity constraints, whose relaxation would lead to a Lagrangian relaxation approach.

In linear problems, first stage subproblem can consider the recourse function of the following stage, either from the explicit equations of the nodes of that following stage or from approximation cuts. In both cases, it will reach the same optimal solution as long as the approximation satisfies some "quality" requirements. This idea is the ground on which Benders decomposition is developed. This can be further extended to the case of complete-scenario decomposition, such that when formulating a subproblem the nodes common to other subproblems will receive a complete definition of the recourse function from the nodes of the following stage:

- Nodes that belong to the same subproblem will provide an exact evaluation of the recourse function on their scenario.
- Nodes coming from another subproblem will provide an approximate evaluation through their Benders cuts.

Formally, the mathematical formulation of each subproblem, corresponding to scenario

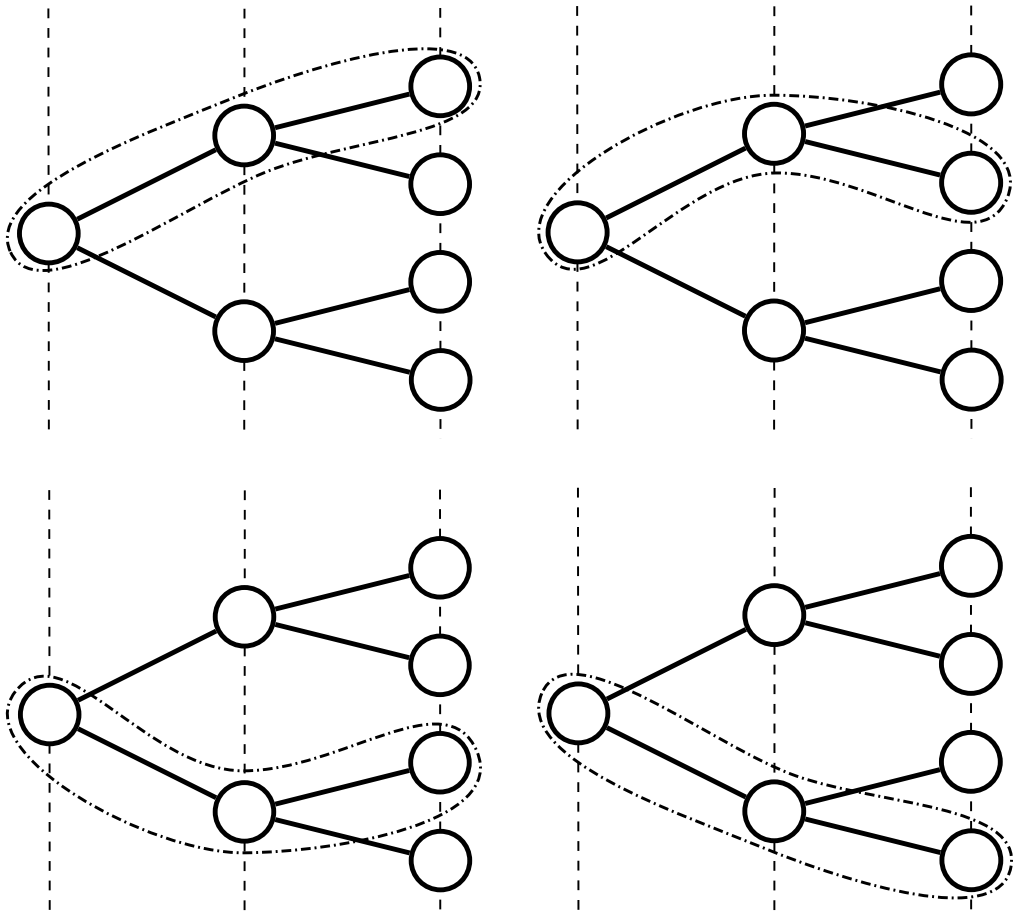

Figure 3: Partial-scenario decomposition example applied to a three stage binary tree.

ω^{ξ}, is the following:

$$
\begin{array}{lll}
\min\limits_{x_t^{\xi},\theta_{\underline{t}}^{\xi}} & \sum\limits_{(\xi,t)\in S} {c_t^{\xi}}^T x_t^{\xi} + \sum\limits_{(\xi,\underline{t})\in Sb} \theta_{\underline{t}}^{\xi} & \\
s.t. & A_t^{\xi} x_t^{\xi} = b_t^{\xi} & \forall\,(\xi,t) \in S \\
 & T_t^{\xi} x_{t-1}^{\xi} + W_t^{\xi} x_t^{\xi} = h_t^{\xi} & \forall\,(\xi,t) \in S, (\xi,t-1) \in S \quad (SP2) \\
 & \delta_{\underline{t}}^{\xi i}\theta_{\underline{t}}^{\xi} \geq \theta_{\underline{t}}^{\xi i} + {\pi_{\underline{t}}^{\xi i}}^T T_{\underline{t}}^{\xi}\left(x_{\underline{t}-1}^{\xi i} - x_{\underline{t}-1}^{\xi}\right) & \forall\,(\xi,\underline{t}) \in Sb, i \in I \\
 & x_t^{\xi} \geq 0 & \forall\,(\xi,t) \in S
\end{array}
$$

where I designates the set of cuts already generated (in the usual Benders decomposition way), $S = \left\{(\xi,t)\,/\omega_t^{\xi} \in \omega^{\xi}\right\}$ is the set of all the nodes of scenario ω^{ξ}, and Sb is the set of branching nodes from sibling scenarios, that branch from scenario ω^{ξ} at each stage $\underline{t}$:

$$Sb = \left\{(\xi,\underline{t})\,/\omega_{\underline{t}}^{\xi} \notin S, pred\left(\omega_{\underline{t}}^{\xi}\right) \in S\right\}$$

When the recourse problem of scenario ω^{ξ} for a sibling scenario $\omega^{\xi'}$ has to be solved, the variables $\left\{x_t^{\xi} : \omega_t^{\xi} \in \omega^{\xi}, \omega_t^{\xi} \in \omega^{\xi'}\right\}$ from common stages are fixed to the values proposed by

the sibling scenario $\omega^{\xi'}$. Bearing this in mind, the recourse problem for stage $\overline{t}$ (where both scenarios branch) can be formulated as follows:

$$
\begin{array}{lll}
\theta_{\overline{t}}^{\xi}\left(x_1^{\xi}=x_1^{\xi'},\ldots,x_{\overline{t}-1}^{\xi}=x_{\overline{t}-1}^{\xi'}\right)= & & \\
\min\limits_{x_t^{\xi},\theta_{\underline{t}}^{\xi}} \ \sum\limits_{t\geq\overline{t}} c_t^{\xi}x_t^{\xi}+\sum\limits_{\underline{t}>\overline{t}}\theta_{\underline{t}}^{\xi} & & \\
s.t. \quad A_t^{\xi}x_t^{\xi}=b_t^{\xi} & \forall\,(\xi,t)\in S, t\geq\overline{t} & (RP2)\\
\quad T_t^{\xi}x_{t-1}^{\xi}+W_t^{\xi}x_t^{\xi}=h_t^{\xi} & \forall\,(\xi,t)\in S,(\xi,t-1)\in S & \\
\quad \delta_{\underline{t}}^{\xi i}\theta_{\underline{t}}^{\xi}\geq\theta_{\underline{t}}^{\xi i}+\pi_{\underline{t}}^{\xi i}T_{\underline{t}}^{\xi}\left(x_{\underline{t}-1}^{\xi i}-x_{\underline{t}-1}^{\xi}\right) & \forall\,(\xi,\underline{t})\in Sb,\underline{t}\geq\overline{t},i\in I & \\
\quad W_{\overline{t}}^{\xi}x_{\overline{t}}^{\xi}=h_{\overline{t}}^{\xi}-T_{\overline{t}}^{\xi}x_{\overline{t}-1}^{\xi} & \forall\,(\xi,\overline{t})\in S & \\
\quad x_t^{\xi}\geq 0 & \forall\,(\xi,t)\in S,t\geq\overline{t} &
\end{array}
$$

The resolution method that derives from this complete-scenario partition scheme proceeds iteratively solving at each iteration each subproblem separately in two different phases:

- In the first phase, each subproblem SP2 is solved, considering its scenario in all the stages. This resolution takes into account the rest of the scenario tree by means of the approximation cuts provided in earlier iterations by the rest of subproblems. Furthermore, the solution obtained is a valid proposal for solving the other subproblems in the second phase.

- In the second phase, each subproblem solves the recourse problems RP2 for each of it sibling scenarios, i.e. it fixes the values of the variables that are common to the sibling scenario with the proposal given by this sibling subproblem in the first phase of the previous iteration. In this way, the recourse function for the sibling scenario is built, and an approximation cut (a Benders cut) can be computed for the common variables proposal provided.

These two phases are solved on each iteration of the solution algorithm, carrying out both phases for each subproblem before consulting new information from other subproblems (on the following iteration). This strategy allows solving the subproblems independently in a distributed fashion, requiring synchronization only at the end of each iteration, when new information is exchanged among subproblems. In this sense there is no forward and backward pass, since all computations are performed together when a subproblem is considered for solution.

When the approximation provided by the Benders cuts is good enough, the proposed method will converge to the complete problem optimal solution. This can be proved by

considering this method as a multiple superimposed Benders decompositions, where each scenario acts both as the master problem for himself and as a subproblem for the rest of scenarios. Hence, the convergence proof of the basic Benders decomposition can also be applied to this method, guaranteeing its convergence (Latorre 2007).

3.3 Practical implementation of the algorithms

As a final description, to clearly point out the differences between the two presented methods, their distributed algorithms are shown in figure 4 and figure 5. The first computation that is performed is the determination of the solution order of subproblems to be followed each iteration. For the partial-scenario decomposition, this order indexes subproblems first for the forward pass and later for the backward pass, including a subproblem once its parent subproblem (in the forward pass) or its children subproblem (in the backward pass) have already been included. This assures that when solving a subproblem, all the input information it needs is already been sent for solution. For the complete-scenario decomposition, since there are no dependencies between subproblems inside the iteration and there is no forward and backward pass, the execution order is a mere enumeration of subproblems.

The key point that determines the performance of each algorithm is marked with an asterisk in both figures. In partial-scenario decomposition, in order to solve a subproblem, it needs to receive information from its parent or child scenarios (depending if it is being solved in the forward or backward pass). In the complete-scenario decomposition method, it just needs to wait for an idle computer to proceed with the computations. Besides, partial-scenario is affected by additional time constraints since subproblems must wait for information from previous executions: primal solutions from parent subproblems during the forward pass, and approximation cuts from children subproblems during the backward pass.

It can be seen that, both decomposition schemes synchronize the execution at the end of each execution, but since partial-scenario decomposition generates subproblems that depend upon the solution of others to proceed with the computations in the same iteration, it limits the performance more tightly than the complete-scenario decomposition method, which is more likely to make use of all available grid resources. On the other hand, comparing the computation tasks carried out on the servers, shown in figure 6 and figure 7, partial-scenario decomposition means a lighter total computational load than complete-scenario decomposition. This can be seen in the fact that partial-scenario decomposition has to solve SP1, which is usually smaller than SP2 since it does not consider the whole scenario, while

```
begin
 Compute the subproblem solution order
 Iteration = 0
 repeat
  ProblemsSent = ProblemsReceived = 0
  while ProblemsSent < ProblemCount do
   if (There is an idle computer) and
      (Subproblem has the information) then        (*)
    Send next subproblem SP1 to the grid
    ProblemsSent = ProblemsSent + 1
   else
    Wait for subproblem completion
    Share the new information resulting from the resolution
    ProblemsReceived = ProblemsReceived + 1
   end if
  end while
  while ProbReceived < ProblemCount do
   Wait for subproblem completion
   Share the new information resulting from the resolution
   ProblemsReceived = ProblemsReceived + 1
  while end
 until (Iteration > IterationLimit) or (Convergence achieved)
end
```

Figure 4: Algorithm for the partial-scenario decomposition algorithm

```
begin
 Compute the subproblem solution order
 Iteration = 0
 repeat
  ProblemsSent = ProblemsReceived = 0
  while ProblemsSent < ProblemCount do
   if (There is an idle computer) then        (*)
    Send next subproblem (SP2 and RP2) to the grid
    ProblemsSent = ProblemsSent + 1
   else
    Wait for subproblem completion
    Share the new information resulting from the resolution
    ProblemsReceived = ProblemsReceived + 1
   end if
  end while
  while ProbReceived < ProblemCount do
   Wait for subproblem completion
   Share the new information resulting from the resolution
   ProblemsReceived = ProblemsReceived + 1
  while end
 until (Iteration > IterationLimit) or (Convergence achieved)
end
```

Figure 5: Algorithm for the complete-scenario decomposition algorithm

complete-scenario decomposition does not only need to solve SP2 (usually larger) but also needs to solve RP2 for each sibling subproblem. The balance between the two factors (grid efficiency and total computational load) can lead to different performances and provides the key for tuning the complete-scenario method.

```
begin
 Receive subproblem
 Formulate and solve subproblem SP1
 Wait to send back the results
end
```

Figure 6: Algorithm for the distributed task of the partial-scenario decomposition algorithm

```
begin
 Receive subproblem
 Formulate and solve complete scenario subproblem SP2
 for each sibling scenario do
  Formulate and solve recourse problem RP2
 end for each
 Wait to send back the results
end
```

Figure 7: Algorithm for the distributed task of the complete-scenario decomposition algorithm

4 Experimental results

This section presents the results of applying the previous decomposition methods to a real problem. The application case selected is a medium-term hydrothermal coordination problem, which is the problem that computes the amount of electricity that must be produced by all hydroelectric and thermal power plants during several time periods minimizing variable operation costs, and subject to meeting the demand and to diverse technical constraints. The example case considers an electric power system of real size, with 45 hydroelectric plants and 129 thermal plants. The problem considers a time horizon of one year with weekly decisions, and represents uncertainty in water inflows at key reservoirs by means of a binary scenario tree comprising 16 scenarios branching at weeks 5, 10, 15 and 20.

The solution of this problem has been carried out in two prototype grid systems, each made up of 10 individual PCs (comprising single-core or dual-core CPUs) and a manager computer. This number of PCs is big enough to compare the performance of both methods and matches the "small-department" license limit of CPLEX (ILOG 2009), the solver used to solve individual LP problems in each grid computer. The methods will perform in larger systems similarly to what is presented here, as it will be shown. The grid is managed with NetSolve (Seymour et al. 2005), a middleware that offers an easy access to the grid resources with a simple programming interface. The basic structure of a NetSolve system is organized around three main components:

- Server component, which performs the computing tasks that are distributed to the grid, and is executed in all PCs that are grouped in the grid.
- Client component, which provides the user an entry point to the grid resources, and thus it is linked into the user application.
- Agent component, that constitutes the core of the matching system, tracking each available resource and assigning those resources to the requests from the users.

In the prototype grids used in this paper, one computer runs the agent and the client application, while the remaining CPUs are devoted to computing tasks as servers in the grid. The precise configuration of the prototype grids are as follows:

- Grid1, the first grid prototype, is composed of 10 different computers as computing servers. Among these computers, 8 of them have slightly faster CPUs (2.4 GHz Intel Pentium 4 processors, instead of 1.8 GHz), which contributes to produce a mixture of capabilities that might be found in general grids. All of them have 1 GB of RAM memory. And finally, the manager computer has the same capabilities as the fastest computing servers.
- In Grid2, the second grid prototype, there are 5 computer servers each with a dual-core CPU (3.40 GHz Intel Pentium D processors) and 1 GB of RAM memory, making a 10 cores grid. In this case, each individual core is faster than in Grid1, but since the access to RAM memory is shared by both cores, it will affect the performance. The manager computer is another dual-core CPU like the computing servers.

Although grid systems are usually geographically disperse and not totally controlled by the user, this work shows results for these two prototype grids where the focus is brought into the decomposition methods performance. Hence, no analysis about system availability is performed. The results shown in this section are divided into two subsections, according to the grid prototype in which they were obtained: section 4.1 presents results for Grid1 and section 4.2 presents the respective results for Grid2. Results are shown for the partial-scenario and complete-scenario decomposition algorithms, while other non node-aggregating methods (like the Benders nested decomposition) are not considered because of the communication penalty imposed due to their high number of subproblems.

4.1 Results in the grid prototype with single-core CPUs

As a first step, to test the performance of the methods in the presence of a non-limiting grid, the 16 scenarios are grouped in pairs to make 8 subproblems. This way, the amount of computers in the grid is larger than the number of subproblems, and thus the grid is more realistic and will not hinder the top performance of the methods. In Table 1 the main metrics for each method are shown. These metrics include:

- The number of iterations needed by each method to achieve convergence.
- The wall clock time seen by the user to obtain the solution.
- The CPU or computing time, addition of all CPU times of the computers in the grid, which is mainly spent on solving the LP problem, but also includes ancillary tasks like processing input information in each server.
- And the performance of the grid system with each method, computed as the ratio:

$$\text{Performance} = \frac{\text{CPU time}}{\text{Wall clock} \cdot \text{Number of computers}}$$

Decomposition method	Iterations	Wall clock [sec]	CPU time [sec]	Performance
Partial-scenario	7	736	1064	18.1 %
Complete-scenario	8	716	4100	71.5 %

Table 1: Experimental results in Grid1 with 8 subproblems

In this experiment, it can be seen that the complete-scenario decomposition needs more iterations to converge, and more CPU time because of the extra resolutions it performs for

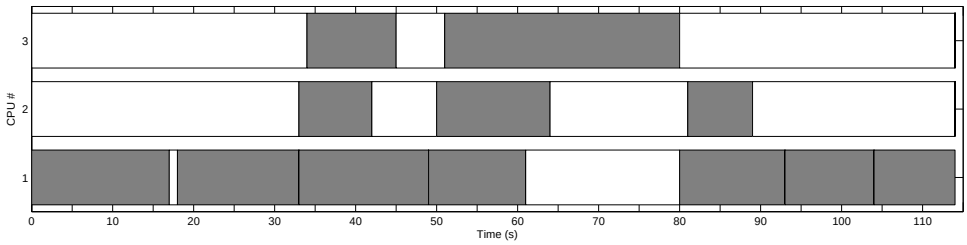

Figure 8: Chronogram of one iteration of the partial-scenario decomposition with 8 subproblems

the siblings of each subproblem. However, it achieves better performance in the grid, and as a consequence, the total user time (wall clock) is shorter than in the partial-scenario decomposition. The difference in performance can be explained because the complete-scenario method does not create any high task dependency that might force some machines to wait for others to conclude their computations. In fact, as it has been previously noted, complete-scenario decomposition can send all the subproblems simultaneously to the grid. To further analyze this fact, figure 8 and figure 9 show, for one iteration, the chronograms of both decomposition methods executed in Grid1. It can be seen that, although there are 10 computers in the grid, partial-scenario decomposition uses just 3 of them, because of the dependencies among subproblems, thus reducing the performance achievable by this method. On the other hand, the complete-scenario decomposition method reaches a better performance because it does not suffer from this execution constraint and makes full use of the 8 computers from the beginning.

Trying to find a situation in which complete-scenario decomposition does not beat partial-scenario decomposition, it was decided to decompose the problem in a number of subproblems that exceeds the size of the grid. Decomposing the problem into 16 subproblems leads to longer solution times in both approaches (see table 2). As the grid consists of 10 computers, in both methods there are more subproblems than computers in the grid and the subproblems cannot be solved simultaneously: at most 10 subproblems can begin their execution while the remaining 6 must wait until more resources become available. This simulates a congested or limited grid, as opposed to the previous case when there were no restrictions.

In this situation, it can be seen that the number of iterations increases due to the larger number of subproblems to be coordinated in the resolution, although the ratio in the iteration count between the two methods is similar to the previous case. For complete-scenario decomposition, the main drawback of having a number of subproblems larger than the com-

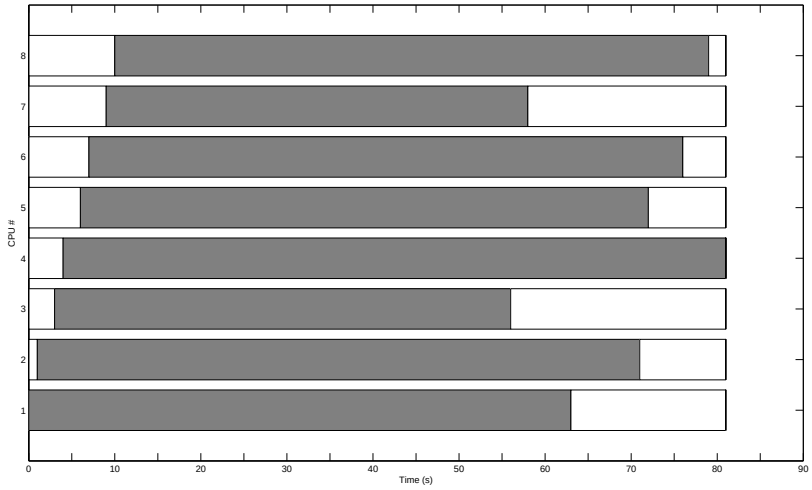

Figure 9: Chronogram of one iteration of the complete-scenario decomposition with 8 subproblems

Decomposition method	Iterations	Wall clock [sec]	CPU time [sec]	Performance
Partial-scenario	11	919	1732	18.9 %
Complete-scenario	13	1461	8474	58.0 %

Table 2: Experimental results in Grid1 with 16 subproblems

puting servers in the grid is that these subproblems cannot be solved simultaneously. This increases the wall clock time when compared with partial-scenario decomposition, whose subproblems are easier to solve. Complete-scenario decomposition solves the 16 subproblems in two batches: the first set of 10 subproblems and the remaining 6 subproblems. Having to perform several consecutive batches of resolutions of subproblems for each iteration increases the total user time. Furthermore, the second set of 6 subproblems does not use all the computers in the grid, and this explains the performance decrease from the previous case to this one, from 71.5% to 58.0%.

4.2 Results in the prototype grid with dual-core CPUs

In this section, the same problem is solved using Grid2, made up of 5 computers with dual-core CPUs. This test shows the effect of RAM bottleneck and cache efficiency over the performance. For this purpose, two tests are executed:

- In case C1 the 16 scenarios are grouped into 4 subproblems, and the solution procedure is allowed to solve one subproblem at a time in each CPU. Hence, only one of the

cores of each CPU is used, and there is no interference in the access to the memory. Consequently, four computers are used, each running one task.

- In case C2 the 16 scenarios are also grouped into 4 subproblems, but this time each CPU is allowed to use both of its cores. This employs the default task scheduling strategy of NetSolve, which schedules all the tasks that are permitted at one CPU (in this case two tasks) before sending any task to the next idle CPU. In this case, two computers are used, each running two tasks.

Table 3 shows the results from the execution of these cases in the second grid prototype, Grid2. These results include the amount of iterations needed to achieve convergence, the total wall clock time measured from the user point of view, the CPU time spent on calculations with CPLEX, and the performance obtained in the grid as described in the previous section (but taking into account the number of cores actually used).

Case	Decomposition method	Iterations	Wall clock [sec]	CPU time [sec]	Performance per core
C1	Partial-scenario	6	514	555	27.0 %
	Complete-scenario	6	428	1149	67.2 %
C2	Partial-scenario	6	556	639	28.7 %
	Complete-scenario	6	581	1608	69.2 %

Table 3: Experimental results in Grid1 with 16 subproblems

Comparing the results from cases C1 and C2, it can be seen the effect of the collision in the access to the RAM memory that increases CPU time. In C1 there are no memory collisions as there is only one core at work at each CPU. On the other hand, case C2 gives the grid access to two cores per CPU, which will share the bandwidth of the memory bus and may block each other, or may cause the cache memory to be less effective due to the increased RAM range (two subproblems instead of one) that is being accessed. This affects in a different degree the partial-scenario decomposition and the complete-scenario decomposition, as can be seen in Table 3. Although the performance of the methods is kept similar in both cases, because it mainly depends on the decomposition strategy, the increase in CPU time from C1 to C2 is quite different for each method: a 15% for partial-scenario decomposition against a 40% for complete-scenario decomposition. As the subproblems considered in the complete-scenario decomposition are larger, this effect in RAM access is more noticeable in this method. The CPU time increase forces the wall clock time to increase, thus reaching the limit in which a better grid performance does not compensate the longer CPU time.

5 Conclusions

This paper analyzes critical features of stochastic programming decomposition when distributed computing environments are used, especially when the amount of resources available is uncertain like in grid computing systems. Decomposition methods that require larger CPU times would be ignored from a classical (single-computer) point of view, but it may pay off the increased total computing time by obtaining better grid performance if the subproblems can be solved more independently in parallel.

This paper describes two methods (partial-scenario decomposition and complete-scenario decomposition) that stem from Benders decomposition, describing how they partition the scenario tree into subproblems as well as the algorithms for solving those subproblems in a grid environment. In a nutshell, partial-scenario decomposition relies upon conventional nested Benders decomposition with node level aggregation, whereas complete-scenario decomposition employs scenario level aggregation to obtain more independent subproblems.

To evaluate the capabilities of these methods, this paper presents results for solving real-sized stochastic linear programming problems and analyzes their performance when executed on two different grid systems. Two different aspects have been tested: first the effects of resource availability and task dependency, and second the effectiveness of new multiple-core CPU architectures.

The complete-scenario decomposition generates independent subproblems that can be solved in parallel from the beginning (as long as resources are available). This method has attained its best results in the presence of a large-enough grid (with a number of computers that exceeds the amount of subproblems that can be launched at the same time) and high throughput CPUs (which provide the computing power and memory needed by this method). Otherwise, it can be outperformed by the partial-scenario decomposition, which resembles the classical Benders decomposition with stage aggregation and inherits parallelization constraints, but requires less computing time due to the lower size of the subproblems.

References

Anstreicher, K., N. Brixius , J.-P. Goux, J.T. Linderoth. 2002. Solving Large Quadratic Assignment Problems on Computational Grids. *Mathematical Programming* Series B **91** 563–588.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4** 238–252.

Birge, J. R. 1985. Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs. *Operations Research* **33** no 5 989-1007.

Birge, J.R., C.J. Donohue, D.F. Holmes, O.G. Svintsitski. 1996. A Parallel Implementation of the Nested Decomposition Algorithm for Multistage Stochastic Linear Programs. *Mathematical Programming* **75** 327–352.

Birge, J.R., F. Louveaux. 1997. *Introduction to stochastic programming*. Springer Verlag.

Bisseling, R.H., T.M. Doup, L.D.J.C. Loyens. 1993. A Parallel Interior Point Algorithm for Linear Programming on a Network of Transputers. *Annals of Operations Research* **43** 51–86.

Cerisola, S., A. Ramos. 2000. Node Aggregation in Stochastic Nested Benders Decomposition Applied to Hydrothermal Coordination. *6th International Conference on Probabilistic Methods Applied to Power Systems (PMAPS), Madeira, Portugal, September 2000*.

Dantzig, G.B., P. Wolfe. 1960. Decomposition principle for linear programs. *Operations Research* **8** no 1 101–111.

Dantzig, G.B. 1963. *Linear Programming and Extensions*. Princeton University Press. Princeton, USA.

Dempster, M.A.H., R.T. Thompson. 1998. Parallelization and Aggregation of Nested Benders Decomposition. *Annals of Operations Research* **81** 163–188.

Ferris, M.C., C.T. Maravelias, A. Sundaramoorthy. 2009. Simultaneous Batching and Scheduling Using Dynamic Decomposition on a Grid. *INFORMS Journal on Computing* **21** no 3 398410

Forrest, J.J.H., J.A. Tomlin. 1990. Vector Processing in Simplex and Interior Methods for Linear Programming. *Annals of Operations Research* **22** 71–100.

Foster, I., C. Kesselman. 1998. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Geoffrion, A.M. 1970. Elements of Large Scale Mathematical Programming: Part I: Concepts. *Management Science* **16** no 11 652–691.

Goux, J.-P., S. Leyffer. 2002. Solving Large MINLPs on Computational Grids. *Optimization and Engineering* **3** no 3 327–346.

Iamnitchi, A., I.T. Foster. 2000. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. *International Conference on Parallel Processing*, 4–14.

ILOG. 2009. ILOG CPLEX: High-performance software for mathematical programming and optimization, http://www.ilog.com/products/cplex/, accessed on Nov 2009.

Karypis, G., A. Gupta, V. Kumar. 1994. A Parallel Formulation of Interior Point Algorithms. *Proceedings Supercomputing* (IEEE Computer Society, ed.). New York, NY, USA. 204–213.

Latorre, J.M. 2007. *Resolución Distribuida de Problemas de Optimización Estocástica. Aplicación al problema de coordinación hidrotérmica* (*Distributed Resolution of Stochastic Programming Problems. Application to the Hydrothermal Coordination Problem*), Ph.D. Thesis, Universidad Pontificia Comillas. Madrid, Spain.

Latorre, J.M., S. Cerisola, A. Ramos, R. Palacios. 2009. Analysis of stochastic problem decomposition algorithms in computational grids. *Annals of Operations Research* **166** no 1, 355–373.

Linderoth, J.T., S.J. Wright. 2003. Decomposition Algorithms for Stochastic Programming on a Computational Grid. *Computational Optimization and Applications* **24** no 2-3 207–250.

Linderoth, J.T., S.J. Wright. 2005. Computational Grids for Stochastic Programming. *Applications of Stochastic Programming* (S. Wallace, y W. Ziemba, eds.), SIAM, SIAM Mathematical Series on Optimization. 61–77.

Moritsch, H., G.Ch. Pflug, M. Siomak. 2001. Asynchronous Nested Optimization Algorithms and Their Parallel Implementation. *Proceedings of the International Software Engineering Symposium, Wuhan, China, 2001.*

Mulvey, J.M., A. Ruszczynski. 1995. A New Scenario Decomposition Method for Large-Scale Stochastic Optimization. *Operations Research* **43** no 3 477–490.

Nielsen, S.S., S.A. Zenios. 1997. Scalable Parallel Benders Decomposition for Stochastic Linear Programming. *Parallel Computing* **23** no 8 1069–1088.

Rockafellar, R.T., R.J.-B. Wets. 1991. Scenarios and Policy Aggregation in Optimization under Uncertainty. *Mathematics of Operations Research* **16** 119–147.

Ruszczynski, A. 1986. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming* no 35 309–333.

Seymour, K., A. Yarkhan, S. Agrawal, J. Dongarra. 2005. NetSolve: Grid Enabling Scientific Computing Environments. *Grid Computing and New Frontiers of High Performance Processing* (L. Grandinetti, ed.), Elsevier, Volume 14 of Advances in Parallel Computing.

Shu, W., M.-Y. Wu. 1993. Sparse Implementation of Revised Simplex Algorithms on Parallel Computers. *The Sixth SIAM Conference on Parallel Processing for Scientific Computing, 1993*, 501–509.

Slyke, R. Van, R. J-B. Wets. 1969. L-shaped linear programs with applications to control and stochastic programming. *SIAM Journal on Applied Mathematics* **17** 638663.

Van Roy, T.J. 1983. Cross decomposition for mixed integer programming. *Mathematical programming* **25** 46–63.

List of Figures