A parallel computation approach for solving multistage stochastic network problems

L.F. Escudero[a,b], J.L. De La Fuente[c], C. García[a] and F.J. Prieto[d]

[a]*Iberdrola Ingeniería y Consultoría, Avda. de Burgos 8, E-28036 Madrid, Spain*

E-mail: leb@iberdrolaingenieria.es

E-mail: cristina.garcia@iberdrola.es

[b]*Department of Statistics and Operations Research, Universidad Complutense, Avda. Complutense s/n, E-28040 Madrid, Spain*

E-mail: escudero@eucmvx.sim.ucm.es

[c]*Iberdrola, C/ Hermosilla 3, E-28001 Madrid, Spain*

E-mail: jl.delafuente@iberdrola.es

[d]*Department of Statistics and Econometrics, Universidad Carlos III de Madrid, C/ Madrid 126, E-28903 Getafe (Madrid), Spain*

E-mail: fjp@est-econ.uc3m.es

This paper presents a parallel computation approach for the efficient solution of very large multistage linear and nonlinear network problems with random parameters. These problems result from particular instances of models for the robust optimization of network problems with uncertainty in the values of the right-hand side and the objective function coefficients. The methodology considered here models the uncertainty using scenarios to characterize the random parameters. A scenario tree is generated and, through the use of full-recourse techniques, an implementable solution is obtained for each group of scenarios at each stage along the planning horizon.

As a consequence of the size of the resulting problems, and the special structure of their constraints, these models are particularly well-suited for the application of decomposition techniques, and the solution of the corresponding subproblems in a parallel computation environment. An augmented Lagrangian decomposition algorithm has been implemented on a distributed computation environment, and a static load balancing approach has been chosen for the parallelization scheme, given the subproblem structure of the model. Large problems – 9000 scenarios and 14 stages with a deterministic equivalent nonlinear model having 166000 constraints and 230000 variables – are solved in 45 minutes on a cluster of four small (11 Mflops) workstations. An extensive set of computational experiments is reported; the numerical results and running times obtained for our test set, composed of large-scale real-life problems, confirm the efficiency of this procedure.

1. Introduction

In this paper, we are concerned with the application of a parallel computation approach to the solution of multistage stochastic linear and nonlinear network problems obtained from the formulation of network problems with uncertainty both on the right-hand side of the constraints and in the objective function coefficients. This class of problems is among the most intractable in numerical computation. Our methodology can be extended in a straightforward manner to the case when the uncertainty also appears in the constraint matrix coefficients.

In contrast to traditional mathematical programming approaches, we have chosen to model the uncertainty using scenarios to characterize the random parameters in the objective function and in the right-hand side. A scenario tree is generated and, through the use of full-recourse techniques, an implementable solution is obtained for each group of scenarios at each stage along the planning horizon.

When this approach is used, the so-called *deterministic equivalent* (DE) model has a huge number of variables and constraints, and very often the network structure is lost as a consequence of the need to impose additional conditions on the values of the variables, to ensure the coherence of the decisions taken at different time stages. Practical problems can easily have in excess of 100,000 variables and constraints; also, very frequently the function to optimize is a nonlinear function of the variables.

Decomposition techniques are particularly suitable for this class of models, by careful choice of the approximation scheme, given the very large size of the resulting problems and their ability to recover the network structure in the subproblems.

Once the problem has been decomposed, a reasonable and efficient procedure to compute a solution is to treat the different subproblems in parallel, where each subproblem is composed of one or more nodes of the scenario tree. We have chosen a static load balancing scheme, as the subproblem structure is not modified within the decomposition procedure and the relative effort required to solve the subproblems remains fairly constant throughout the algorithm.

This approach has been tested, and shown to be very efficient, on a collection of large test problems obtained from real-life long-term hydropower generation planning applications. The tests have compared both the sequential and parallel versions of our implementation of the decomposition algorithm, and this algorithm with other alternatives for the direct solution of the DE model.

The paper is organized as follows: in section 2, we present the model of interest and we comment on its special structure. Section 3 presents the compact representation of the DE model. Section 4 gives the so-called splitting variable representation of the same model, and the related augmented Lagrangian decomposition since we consider it as the most suitable for the parallel solution of the problem. Section 5 presents the parallelization of the algorithmic framework of choice, and describes the details involved in this parallelization. Section 6 shows results obtained from the application

of these techniques to three sets of very large real-world test problems. Finally, in section 7, we draw some conclusions from our experience on the application of this approach.

2. Model description

Consider the following deterministic linear problem:

$$\begin{aligned} z = \text{ minimize } & c^T x \\ \text{subject to } & A\,x = b, \\ & x \geq 0, \end{aligned} \tag{1}$$

where c is an n-column vector, A is the $m \times n$ constraint matrix, b is an m-column vector and x is the n-column vector of the nonnegative unknowns. We are particularly interested in the solution of planning problems where time plays a significant role in the definition of the variables and the structure of the constraints. If we consider a planning problem over several time periods, we may partition the variables into sets corresponding to information concerning just one stage (a period or set of periods to be treated jointly in the model), and to the information carried over from one stage to the next. Similarly, we may partition the constraints into subsets related to the limitations imposed on the operation of the system in each stage.

In our case of interest, we shall assume that the structure of the constraints is similar for all periods, and that all constraints can be treated as pertaining to a single stage (and the variables representing the information shared between stages). Under these assumptions, the structure of the constraints is that of a collection of models for each stage, linked by terms related to the variables carrying information from each stage to the next.

In order to illustrate this structure, consider a case with three time stages, where the variables x_r represent the information related to stage r, and the variables $x_{r,r+1}$ correspond to the information carried over from stage r into stage $r + 1$. In this case, the system $Ax = b$ can be written as

$$\begin{aligned} A_1x_1 + A^1_{12}x_{12} &= b_1, \\ A^2_{12}x_{12} + A_2x_2 + A^2_{23}x_{23} &= b_2, \\ A^3_{23}x_{23} + A_3x_3 &= b_3. \end{aligned} \tag{2}$$

There exists a broad range of application fields for stage programs of this form, such as electricity/water/gas distribution, financial planning, exchange rate management with currency arbitrage, production planning, input–output tables in macroeconomic analysis, design and operation of communications networks, road networks, etc. See [1,2,6,13,17,18,21,22,24,27,33,34,36,42,47–50], among others. In many

such cases, A in (1) is an $m \times n$ node–arc incidence matrix for some directed graph $G = (V, E)$, where V denotes the set of nodes with $m \equiv |V|$, and E denotes the set of arcs with $n \equiv |E|$.

2.1. Uncertainty

The real-life problems that have motivated this work fit models (1) and (2) introduced above, but in order to obtain a sufficiently accurate representation of the operation of the real system, we must expand these models by taking into account explicitly that some elements of the matrix A or the vectors c and b may not be known with certainty.

In these cases, one needs to consider two additional features. In the first place, one must model the availability of information over time, and state what sort of decisions can be made at each of the various stages. Secondly, to compute an optimizer, any proposed solution will have to be compared with other candidate solutions but, in the stochastic setting, the criteria by which this comparison can be performed are much less clear. Thus, we need an approach to model the uncertainty in the problem data. The traditional approach is to make distributional assumptions, estimate the parameters from historical data and then develop a stochastic model to take the uncertainty into account. Such an approach may not be appropriate if only limited information is available. On the other hand, in many applications it is often necessary and possible to take into account information that is not reflected in the historical data. In many such cases, we may employ a technique called "scenario analysis", where the uncertainty is modelled via a set of scenarios, say S. (See [23] for a description of a methodology for the estimation of the number of scenarios; see also [25] for a Monte Carlo importance sampling approach for the generation of the appropriate set of scenarios.)

For example, in our model (1) the vectors for the right-hand side and the objective function coefficients may take different values for different scenarios, say b^s and c^s for $s \in S$, respectively. We also introduce weights w^s representing the likelihood that the decision maker (modeler) associates with each scenario $s \in S$.

One way to deal with the uncertainty is to obtain the solution x that best tracks each of the scenarios, while attempting to satisfy the constraints for each scenario. This can be achieved by obtaining a solution that minimizes the weighted sum or the largest of the upper differences between the proposed solution value and the optimal solution value for each scenario, see [1,2,13,26,32]. The resulting model does not increase the number of variables of the original formulation, but now there are $m|S|$ constraints. Unfortunately, this representation of the model does not preserve the structure of the deterministic model (1) (for example, in the case of network models we have two nonzero constraint entries for each variable, at most, and one of them is +1), and the objective function is no longer linear; see [1,2] for some procedures to overcome this difficulty. Models of this form are known as scenario immunization models, see [13,32], or SI models for short.

As an alternative goal, we could minimize a composite function including the expected value of both the objective function of model (1) and the infeasibility penalization function, so that the new model becomes

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in S} w^s (c^s)^T x + H(x_a^s) \\ \text{subject to} \quad & A x + x_a^s = b^s, \qquad s \in S, \\ & x \geq 0, \end{aligned} \tag{3}$$

where x_a^s is the vector of slack variables for scenario $s \in S$, and $H(x_a^s)$ is the related penalization in the objective function. Note that (3) gives an implementable policy based on the so-called *simple recourse*, that is, the whole vector of decision variables is anticipated at stage 1.

2.2. Nonanticipative policies

Scenario immunization models anticipate decisions in x that for multistage applications may not be needed at stage $r = 1$. Frequently, the decisions for stage $r = 1$ are the only decisions to be made, since at stage $r = 2$ one may realize that some of the data has been changed, some scenarios vanish, etc. In this case, the models will usually be reoptimized in a rolling planning horizon mode. When only spot decisions (i.e., decisions for stage $r = 1$) are to be made, the information about future uncertainty is only taken into account for a better spot decision making. This type of approach is termed *full recourse*.

Let R denote the set of stages, and x^s denote the arc flows under scenario s, for $s \in S$. The following nonanticipativity principle has been stated in [37], see also [45].

> "If two different scenarios s and s' are identical up to stage r on the basis of the information available about them at stage r, then the values for the x-variables must be identical up to stage r for $r \in R$".

This condition guarantees that the solution obtained by the model is not dependent at stage r on information that is not yet available. To illustrate this concept, consider figure 1: each node in the figure represents a point in time where a decision can be made. Once a decision is made, contingencies can happen (in this example, the number of contingencies is three for all nodes), and information related to these contingencies is available at the beginning of the next stage. This information structure is visualized in figure 1 as a tree, where each root-to-leaf path represents one specific scenario and corresponds to one realization of the objective function coefficient vector c and right-hand side b. In our example, we have three stages; the elements of vectors c_1 and b_1 can take one value each for all scenarios, but the elements of vectors c_2 and b_2 can each take three different values, one for each realization of the uncertainty in stage 2.

Figure 1. Scenario tree.

The elements of vectors c_3 and b_3 can then each take three different values for each value of c_2 and b_2. Note that in this case, we have a total of $|S| = 9$ different scenarios.

In order to introduce this condition into our model, for each stage r we define a set of scenario groups G_r such that all scenarios having the same realizations of the uncertainty up to stage r belong to a given scenario group g, for $g \in G_r$. For example, in figure 1 we have that G_1 is composed of a single scenario group (all scenarios are identical in stage 1, as no realization of the uncertainty has yet taken place), G_2 is composed of three groups of scenarios, with three scenarios belonging to each group, and G_3 is composed of nine groups of scenarios, each one including a single scenario. For simplicity of notation, let us assume $G_r \cap G_{r'} = \emptyset, \forall r, r' \in R | r \neq r'$. The nonanticipativity principle then requires a single solution value for each scenario group in each stage, that is, a single value x_r^g for each $g \in G_r$. Let S_g denote the set of scenarios that belong to group g for $S_g \subseteq S$, $g \in G_r$ and $r \in R$, and N denote the set of solutions that satisfy the so-called *nonanticipativity* constraints. That is,

$$x \in N \equiv \{x^s | x_r^s = x_r^{s'} \quad s, s' \in S_g, \quad g \in G_r, \quad r \in R\}. \tag{4}$$

Note also that the scenario tree is defined by the set of nodes $\bigcup_{r \in R} G_r$ and the set of directed arcs E, where $(k, l) \in E$ for $k \in G_r$, $l \in G_{r+1}$ and $S_l \subseteq S_k$.

These constraints must be added to model (1) to obtain the deterministic equivalent (DE) model

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{s \in S} w^s c^{sT} x^s \\ \text{subject to} \quad & A x^s = b^s, \qquad s \in S, \\ & x \in N, \\ & x^s \geq 0, \qquad s \in S. \end{aligned} \tag{5}$$

Model (5) has a nice structure that we may exploit. Two approaches can be used to represent the nonanticipativity constraints (4). One approach is based on a *compact representation*, where (4) is used to eliminate variables and to reduce model size, so that there is a single variable x_r^g for each $g \quad G_r$, but any special structure of the constraints in (1) (e.g. any network type structure) is destroyed. The other approach is based on a *splitting variable* representation. If we denote the variables associated with each stage as x_r^g for $g \quad G_r$ and $r \quad R$, and $x_{r,r+1}^g$ denotes the connecting variables between stages, then we can use a single variable x_r^g for each $g \quad G_r$, as in the preceding case, since this reduction does not affect the network structure. On the other hand, we may introduce more than one variable $x_{r,r+1}^s$ to represent the connecting variables for $s \quad S_g$ and a given $g \quad G_r$; in this case, the nonanticipativity condition is then enforced by explicitly including the corresponding constraints from (4) as part of the model, but the network structure is preserved.

3. Compact representation

The compact representation of the *full recourse* model (5) requires modifying the model example (2) by introducing the following variables to take into account the uncertainty in the data:

x_1 and x_{12} with the same meaning as x_1 and x_{12} in (2), respectively.

x_2^k and x_{23}^k with the same meaning as x_2 and x_{23} in (2), respectively, but now related to each scenario group $k \quad G_2$.

x_3^l with the same meaning as x_3 in (2), but now related to each scenario group $l \quad G_3$.

Let the vectors $(c_1, c_{12}, c_2^k, c_{23}^k, c_3^l)$ and (b_1, b_2^k, b_3^l) denote the objective function coefficients and right-hand side parameters, partitioned according to the scenario tree and variable partitioning scheme indicated above.

For the case with three stages, our representation of model (5) is as follows:

$$
\begin{array}{lllllllll}
\min & c_1^T x_1 + & c_{12}^T x_{12} & + \sum_{k \quad G_2} (c_2^k)^T x_2^k & + \sum_{k \quad G_2} (c_{23}^k)^T x_{23}^k & + \sum_{i \quad G_3} (c_3^l)^T x_3^l & & & \\
\text{s.t.} & A_1 x_1 + & A_{12}^1 x_{12} & & & & = b_1, & & \\
& & A_{12}^2 x_{12} & + A_2 x_2^k & + A_{23}^2 x_{23}^k & & = b_2^k, & k \quad G_2, & \\
& & & & A_{23}^3 x_{23}^k & + A_3 x_3^l & = b_3^l, & (k,l) \quad E, k \quad G_2, & \\
& x_1, & x_{12}, & x_2^k, & x_{23}^k, & x_3^l & \geq 0. & & (6)
\end{array}
$$

Simplex based procedures, with crash ad hoc starting solution techniques, can be used to solve (6), as well as interior point methods. The first approach gives good results, provided the number of scenarios is small (say 25–50), and the loss of the special structure in (1) is not a great inconvenience.

The disadvantages of using interior point methods are analyzed in [10,29] and result from the fill-in in the computation of the Choleski factor required for the solution of a system of linear equations in each iteration.

The two-stage version of (6), the so-called *block angular problem*, has been extensively studied in the literature. Several special-purpose algorithms for solving linear problems with this structure have been developed, including the L-shaped method described in [44], among others. Interior point methods applied to these problems have been discussed in [8,9,46], where decomposable algorithms amenable to parallel computing approaches have been proposed. The basic idea is to decompose the system of linear equations to solve at each iteration, using versions of the Sherman–Woodbury–Morrison formula. This procedure takes advantage of the structure in the two-stage version of (6), and obtains the solution of the system from the solution of $|S|$ smaller systems.

Instead of using a direct method to solve the DE model (6), the multistage stochastic program represented in (6) can be solved using a Benders decomposition [3] approach. It has the property that if the values of the variables x_{12} and x_{23}^k are fixed, then the optimal values for the remaining variables can be obtained from the solution of separable subproblems.

The L-shaped method given in [44] was extended in [5] to the multistage setting by nesting several levels inside one another. Other nested Benders decomposition approaches are described in [14,20,35]; parallelized versions are given in [7,15]. Other parallelizations of the Benders decomposition method for the two-stage stochastic linear program are given in [11,24]. See our computational experience in section 6.

4. Splitting variable representation

4.1. The model

An alternative to the compact representation of the full-recourse model (5) can be obtained after splitting the coupling variables between stages, x_{12} and x_{23}^k in the case of our example (6). A new set of variables x_{12}^k for $k \quad G_2$ are then added and equated to x_{12}, and also a set of variables x_{23}^l for $l \quad G_3$ are added and equated to the corresponding x_{23}^k for $(k, l) \quad E$.

The resulting representation takes the form:

$$
\begin{array}{lllllllllll}
\min & c_1^T x_1 & +c_{12}^T x_{12} & & +\sum_k (c_2^k)^T x_2^k & +\sum_k (c_{23}^k)^T x_{23}^k & & +\sum_l (c_3^l)^T x_3^l & & & \\
\text{s.t.} & A_1 x_1 & +A_{12}^1 x_{12} & & & & & & & = b_1, & \\
 & & x_{12} & - x_{12}^k & & & & & & = 0, & k, \\
 & & & A_{12}^2 x_{12}^k & + A_2 x_2^k & + A_{23}^2 x_{23}^k & & & & = b_2^k, & k, \\
 & & & & & x_{23}^k & - x_{23}^l & & & = 0, & k, l, \\
 & & & & & & A_{23}^3 x_{23}^l & + A_3 x_3^l & & = b_3^l, & l, \\
 & x_1, & x_{12}, & x_{12}^k, & x_2^k, & x_{23}^k, & x_{23}^l, & x_3^l & & \geq 0. & (7)
\end{array}
$$

For a circular-link representation of the nonanticipativity constraints, say $x^{k-1}_{r,r+1} = x^{k}_{r,r+1}$, see [6,28,39]. The advantages of the splitting variable representation over the compact representation for solving the full recourse model (5) using interior point methods have been explored in [29], see also [10]. In [19], a computational comparison of different strategies for interior point methods and the simplex method with and without crash procedures can be found. The results of these studies do not all go in the same direction, but none of them analyze decomposition procedures. A Dantzig–Wolfe decomposition scheme, and (its dual) a Benders approach to the solution of (7) have been presented in [2], but no computational results are reported; the authors claim they are not good enough.

4.2. Augmented Lagrangian decomposition

Augmented Lagrangian methods proceed by moving the nonanticipative constraints (4) into the objective function to generate a problem with independent sets of constraints, one per node in the scenario tree, so that each subset keeps any special structure that might be present in the original problem (1).

If this procedure is applied to (7), the resulting problem becomes

$$\max_{\pi} D_\rho(\pi), \tag{8}$$

where the function $D_\rho(\pi)$ is defined as

$$\begin{aligned} D_\rho(\pi) = \text{ minimize } & \sum_{s \in S} w^s (c^s)^T x^s + \sum_k (\pi^k_{12})^T (x_{12} - x^k_{12}) + \sum_{(k,l)} (\pi^l_{23})^T (x^k_{23} - x^l_{23}) \\ & + \frac{\rho}{2} \left(\sum_k \| x_{12} - x^k_{12} \|^2 + \sum_{(k,l)} \| x^k_{23} - x^l_{23} \|^2 \right) \\ \text{subject to } & x^s \in X \end{aligned} \tag{9}$$

and we have used the notation

$$x^s \equiv (x_1, x_{12}, x^k_2, x^k_{23}, x^l_3), \tag{10}$$

the feasible set X is defined as

$$\begin{aligned} X \equiv \{ x^s \mid & A_1 x_1 + A^1_{12} x_{12} = b_1, \\ & A^2_{12} x^k_{12} + A_2 x^k_2 + A^2_{23} x^k_{23} = b^k_2, \quad k, \\ & A^3_{23} x^l_{23} + A_3 x^l_3 = b^l_3, \quad l, \\ & x_1 \in X_1, \; x_{12} \in X_{12}, \; x^k_{12} \in X_{12}, \; x^k_2 \in X_2, \; x^k_{23} \in X_{23}, \; x^l_{23} \in X_{23}, \; x^l_3 \in X_3 \}, \end{aligned} \tag{11}$$

$\pi \equiv (\pi^k_{12}, \pi^l_{23})$ is the vector of dual variables for constraints (4), and $\rho > 0$ is a penalty parameter.

A rough algorithm to solve (8) is as follows:

Algorithm A-1. **Augmented Lagrangian decomposition algorithm**

Step 1. For a given multiplier vector π_i available at iteration i, solve problem (9). Let x_i^s be the solution of this problem.

Step 2. If for some tolerance parameter $\varepsilon > 0$,

$$\|(x_{12})_i - (x_{12}^k)_i\| \le \varepsilon \quad \text{and} \quad \|(x_{23}^k)_i - (x_{23}^l)_i\| \le \varepsilon, \quad (k, l) \quad E, \; k \quad G_2,$$

holds, then stop; the optimal solution for (8), and for the original problem (6), has been found.

Step 3. Otherwise, reduce by an adequate amount the penalty parameter ρ, and update the dual multipliers π according to

$$\begin{aligned} \pi_{i+1}^k &= \pi_i^k - \beta(x_{12} - x_{12}^k), \qquad k \quad G_2, \\ \pi_{i+1}^l &= \pi_i^l - \beta(x_{23}^k - x_{23}^l), \qquad (k, l) \quad E, \end{aligned} \tag{12}$$

where $\beta > 0$ is an appropriate step length.

It is well known that if (8) has a solution, then algorithm A-1 converges in a finite number of iterations – see [4], for example.

In order to obtain decomposable subproblems, we still need to address the fact that the objective function in (9) is not linear but quasi-separable quadratic. A description of several frameworks to decompose the objective function in (9) can be found in [2].

If the quadratic terms of the form $\|x_i - x_j\|^2$ in (9) are expanded, and the cross-product terms $x_i^T x_j$ are approximated by using a suggestion from [41] – see also [30,31, 38,39] – it results that for the variables x_i, x_j and some particular values of these variables $\bar{x}_i$, $\bar{x}_j$, we can write

$$\begin{aligned} \|x_i - x_j\|^2 &= \|x_i\|^2 + \|x_j\|^2 - 2x_i^T x_j, \\ x_i^T x_j &= -\bar{x}_i^T \bar{x}_j + \bar{x}_i^T x_j + x_i^T \bar{x}_j + (x_i - \bar{x}_i)^T (x_j - \bar{x}_j), \\ \|x_i - x_j\|^2 &\approx \|x_i\|^2 + \|x_j\|^2 + 2\bar{x}_i^T \bar{x}_j - 2\bar{x}_i^T x_j - 2x_i^T \bar{x}_j, \end{aligned}$$

where we assume that the terms of the form $(x_i - \bar{x}_i)^T (x_j - \bar{x}_j)$ are negligible compared to $\|x_i\|^2$ and $\|x_j\|^2$.

Using this result, the objective function in (9) can be approximated by

$$\begin{aligned} F(x, \pi, \rho) = {} & g_1^T x_1 + g_{12}^T x_{12} + \sum_k \left((g_{12}^k)^T x_{12}^k + (g_2^k)^T x_2^k + (g_{23}^k)^T x_{23}^k \right) \\ & + \sum_l \left((g_{23}^l)^T x_{23}^l + (g_3^l)^T x_3^l \right) \\ & + \frac{\rho}{2} \left(\|x_{12}\|^2 + \sum_k \left(\|x_{12}^k\|^2 + \|x_{23}^k\|^2 \right) + \sum_l \|x_{23}^l\|^2 \right) + f, \end{aligned} \tag{13}$$

where

$$
\begin{aligned}
g_1 &= c_1, \\
g_{12} &= c_{12} - \sum_k (\pi_{12}^k + \rho \bar{x}_{12}^k), \\
g_{12}^k &= \pi_{12}^k - \rho \bar{x}_{12}, && k \quad G_2, \\
g_2^k &= c_2^k, && k \quad G_2, \\
g_{23}^k &= c_{23}^k - \sum_{\{l \mid (k,l) \quad E\}} (\pi_{23}^l + \rho \bar{x}_{23}^l), && k \quad G_2, \\
g_{23}^l &= \pi_{23}^l - \rho \bar{x}_{23}^k, && (k,l) \quad E, \\
g_3^l &= c_3^l, && l \quad G_3, \\
f &= \rho \left(\bar{x}_{12} \sum_k x_{12}^k + \sum_k x_{23}^k \sum_{\{l \mid (k,l) \quad E\}} x_{23}^l \right).
\end{aligned}
\tag{14}
$$

Now the problem

$$
\begin{aligned}
&\underset{x}{\text{minimize}} && F(x, \pi, \rho) \\
&\text{subject to} && x \quad X,
\end{aligned}
\tag{15}
$$

where X is defined as in (11), can be decomposed into quadratic subproblems with linear constraints that preserve the original structure, a network structure, for example. Our rough algorithm for solving (9) via separable quadratic approximations is as follows:

Let i be a given iteration of algorithm A-1 and π_i and x_i be the dual multipliers and current value of the variables, respectively. Replace step 1 of algorithm A-1 with the following algorithm:

Algorithm A-2. Augmented Lagrangian inner iteration

Step 0. Let $\pi = \pi_i$, $\bar{x}_{im} = x_i$ and $m = 1$.

Step 1. Solve (15) with $\bar{x} = \bar{x}_{im}$, to obtain a new point x_{im}.

Step 2. If $\| x_{im} - \bar{x}_{im} \| \leq \varepsilon$, then stop. Otherwise, set

$$\bar{x}_{i,m+1} = \bar{x}_{im} + \gamma (x_{im} - \bar{x}_{im}),$$

where γ is an appropriate step length, increase m by one and go to step 1.

In some of the computational experiments to be described in section 6, the value of the penalty parameter ρ in (13) has been kept constant for all iterations, but in other cases, the value of ρ has been adjusted in the algorithm according to the following heuristic rules:

Algorithm A-3. Penalty parameter update

Step 0. At the start of the algorithm define two problem-dependent constants, C_p and C_F, where C_F should be a lower bound on the optimal value of the objective function.
For the test problems in section 6, we have used $C_p = 10^5 n$, where n denotes the number of variables in (15), and $C_F = 0$.

Step 1. In each iteration i of algorithm A-1, compute the values

$$M_i = \max(\|(x_{12})_i - (x_{12}^k)_i\|_\infty, \|(x_{23}^k)_i - (x_{23}^l)_i\|_\infty),$$

$$N_i = \sum_k \|(x_{12})_i - (x_{12}^k)_i\|_2 + \sum_{(k,l)} \|(x_{23}^k)_i - (x_{23}^l)_i\|_2.$$

Step 2. The new value ρ_{i+1} is obtained from the value ρ_i as

$$\rho_{i+1} = \begin{cases} 1.5\rho_i & \text{if } M_i > 1, \\ C_p/N_i & \text{if } M_i < 1 \text{ and } C_p/N_i < 10^2(F(x_i) - C_F), \\ 10^2(F(x_i) - C_F) & \text{otherwise.} \end{cases}$$

Note. Alternatively, a different penalty value may be used for each constraint, to be updated according to rules similar to the ones presented above.

5. Parallel implementation

From the preceding description of the Augmented Lagrangian algorithm, it follows that step 1 in algorithm A-2 can be carried out by solving a collection of quadratic subproblems having very similar structure and complexity. The additional computational work required in step 3 of algorithm A-1 and step 2 of algorithm A-2 is a very small fraction of the total computational effort required by the decomposition algorithm.

Taking advantage of this favorable structure, we have developed a parallel implementation of the algorithms presented in section 4. In this implementation, the computations required in step 1 of algorithm A-2 are conducted over a distributed network of computers (nodes).

For a different approach to the parallelization of the Augmented Lagrangian decomposition method, see [31,39]. The similarly motivated row-action parallelization scheme is presented in [34] for the two-stage problem.

The Augmented Lagrangian parallel implementation that we have developed follows a master/slave strategy with static load balancing. Given this structure, the execution flow of the program can be summarized as follows:

Algorithm A-4. Distributed code execution flow

Step 1. The master program is loaded onto a node of the parallel computer, or a distributed network of computers.

Step 2. The master program loads the data for the problem, determines the subtasks to be performed by the slaves, and allocates these subtasks to the nodes in the system, according to the procedure described in section 5.3.

Step 3. The master program loads the slave programs (where the computationally intensive part of the algorithm is implemented) onto the remaining nodes of the computer (or network).

Step 4. The information required to complete the allocated tasks (step 1 in algorithm A-2) in a given iteration is sent to the slaves.

Step 5. The slaves perform the allocated tasks. Once these tasks are complete, the results are returned to the master program.

Step 6. The parameters in step 3 of algorithm A-1 and step 2 of algorithm A-2 are updated by the master program.

Step 7. With these new values of the parameters, new tasks are generated and allocated to the slaves by the master program.

Step 8. Steps 4–7 in this procedure are repeated until the master program determines that the termination conditions in step 2 of algorithm A-1 have been satisfied.

5.1. Load balancing

In our implementation of the decomposition algorithm, we have chosen to define each subproblem as corresponding to one stage and one scenario group. This choice implies that all subproblems are similar in size and structure, and simplifies the allocation of tasks to processors. Other allocation schemes are possible; for example, if the efficiency in the solution of the subproblems is an important consideration, it may be worth assigning to each subproblem the largest number of stages for which the network structure is preserved.

For this choice, and as the number of subproblems to solve in each iteration is known and fixed in advance, we have chosen to follow a static load balancing scheme. As a consequence, the allocation of subproblems to tasks and the assignment of the individual tasks to the nodes is done once at the beginning of the execution.

If the network connecting the computer nodes is not heavily loaded, then a static load balancing scheme is the most efficient choice, as for our decomposition approach the load remains fairly constant throughout the execution and the overheads associated with dynamic schemes are avoided. Note that the solution times for the different subproblems in a given iteration, measured as a fraction of the total elapsed time for the iteration, do not change significantly from iteration to iteration.

For the tests we have conducted in our computational experiments (see section 6), we have made use of a homogeneous allocation of subproblems to tasks. In this sense, all tasks that we generate are in principle equivalent regarding their associated computational load, a reasonable choice for dedicated machines of roughly similar computing power. Nevertheless, our implementation is also able to generate non-homogeneous allocations, based on a priori knowledge of the processing capabilities of the different nodes in the (nonhomogeneous) network, or on some estimate of the expected load of the nodes. In these cases, it is assumed that this load will remain sufficiently stable that our static load allocation does not become too inefficient.

5.2. Communications between processors

Given that the program has two or more separate parts (master and slaves), one of our main concerns has been to ensure the efficiency of the procedures to communicate the information for the variables shared by each part. The specific design of these communication tasks between programs has a significant impact on the overall efficiency of the parallel algorithm, and the speedup that can be achieved with regard to the sequential version.

In our implementation, this communication is performed at different points in the execution of the program:

- Initially, immediately after the master program loads the slave to each of the nodes (step 3 in algorithm A-4), all the information that will not change throughout the execution of the algorithm is sent to each slave.
- Immediately before a slave must begin processing a particular task, the master sends all information specific to this task (step 4 in algorithm A-4).
- If the “child” of a given subproblem has been allocated to a different node, the values corresponding to the common variables for the subproblem and its child are sent to the corresponding node immediately after solving the subproblem.
- The values of the multiplier vectors for the shared variables between subproblems are updated from the new values for the connecting variables (step 3 in algorithm A-1) immediately after solving both subproblems. If the “parent” for a given subproblem has been allocated to a different node, the new values for the multipliers are not sent to the corresponding parent node until the end of the current iteration, that is, until all the tasks currently allocated to the node have been completed.
- Once a slave finishes a given task, the results generated are sent to the master program, in order to check the optimality conditions and to decide on the need to conduct a new iteration.

Note that the amount of information to be sent to a slave depends on the structure of the subproblems assigned to it. For example, in order to solve a subproblem whose “parent” has been allocated to the same node, no additional information is required.

The most likely cause for a node having to wait to start solving a given subproblem is the need to receive information from another node (the results from solving the parent of the current subproblem), and the waiting for the master to send the information to process a task to a given slave. As a consequence, in order to attain a high speedup, it is necessary that the task allocation algorithm both balances the load between processors, and minimizes the total time a slave must wait for information from another slave.

5.3. Task allocation

As we mentioned above, the allocation of subproblems to tasks has been performed so that all tasks require a similar computational effort, and the information exchange required between slaves is limited. This allocation is performed by the master program immediately after reading all input data and starting the slaves in all available nodes.

Task allocation is conducted according to the following algorithm:

Algorithm A-5. Task allocation to processors

Step 1. Once the input data has been loaded, the scenario tree is analyzed.
Let $\bar{n}_r$ denote the number of different scenario groups associated with stage r, that is, $\bar{n}_r = |G_r|$, and let N be the number of nodes available in the parallel computer, or the network. Find an $\bar{r}$ such that $N \approx 4\bar{n}_{\bar{r}}$.
This last value has been chosen to ensure that a sufficiently well-balanced first assignment can be made for the subproblems corresponding to that stage, without having to consider any later stages.

Step 2. All subproblems in stage $\bar{r}$ are assigned to one of the nodes.
Let s_g, $g = 1,\ldots, \bar{n}_{\bar{r}}$, denote each of the subproblems in stage $\bar{r}$. We define one subproblem for each scenario group at each stage. Also, let n_{s_g} denote the total number of subproblems in later stages associated with subproblem s_g, then n_{s_g} denotes the total number of nodes in the scenario tree from node g to the leafs for $g \quad G_r$.
A given set of ratios for each computer node, u_k, $k = 1,\ldots,N$, satisfying $\sum_k u_k = 1$, and representing a measure of the availability of each processor, or the speed of the processor, is specified in advance.
Each subproblem s_g together with its associated subproblems is assigned to one of the computer nodes. If V_k denotes the set of subproblems in stage $\bar{r}$ assigned to node k, the assignment is conducted so that $\|a - t\|$ is minimized, where

$$a_k = \sum_{i:s_i \; V_k} n_{s_i}, \quad t_k = u_k \sum_{s_g | g \; G_r} n_{s_g},$$

that is, a denotes the vector of assigned loads to each computer node, and t denotes the vector of desired loads for each computer node, obtained from

the prespecified ratios u_k and the total number of subproblems to be allocated. We attempt to minimize the distance between the actual and the desired assignments. In our case, we have used $u_k = 1/N$.

Step 3. The subproblems corresponding to stages $1, 2, \ldots, \bar{r} - 1$ are sequentially assigned to one of the nodes, according to the following rules:

3.1. Rank the unassigned subproblems starting with stage $\bar{r} - 1$ and proceeding to stage 1. Within each stage, use any ordering for the subproblems. Make the first subproblem in this list the current one.

3.2. For the current subproblem, the number of children already assigned to each node, h_k for node k, are determined and ranked. Let k_1 denote the node having the largest number of children, k_2 the second largest, etc. Let l denote the number of nodes having a number of children assigned equal to h_{k_1}, that is, let

$$l = \max\{i \geq 1 : h_{k_1} = h_{k_i}\}.$$

3.3. If $l = 1$, then the current subproblem is assigned to node k_1.

3.4. If $l > 1$, then the current subproblem is assigned to node k_j, where j is taken from

$$j \quad \arg\max_{i \leq l} (t_{k_i} - a_{k_i}),$$

that is, the node having the greatest difference between the required and actual load is selected from all nodes having the largest number of children.

3.5. Make the next subproblem the current one, and go to step 3.2.

This scheme has the advantages of being fairly simple to implement, and producing a reasonably well-balanced assignment, while reducing the amount of information to be exchanged between slaves (see section 5.2). Note that the exact amount of effort to solve a given subproblem will vary during the solution process, and cannot be known in advance; so it does not seem efficient to try to compute an "optimal" allocation scheme.

Figure 2 shows an illustration of the behavior of algorithm A-5. In this figure, we indicate the partitioning of the subproblems generated by this algorithm for the case of having 4 stages and 27 scenarios. We also show the communications that need to be established between slaves to interchange information in each iteration, and the connections between subproblems that require these interchanges. Note that there are two different types of communication involved in the solution process. The continuous arrows denote the communication between slaves, taking place in one direction as the values for the common variables for the subproblems are sent from parent to child during the iteration. The broken arrows denote the (bidirectional) communications between master and slaves, involving the slaves sending to the master information on

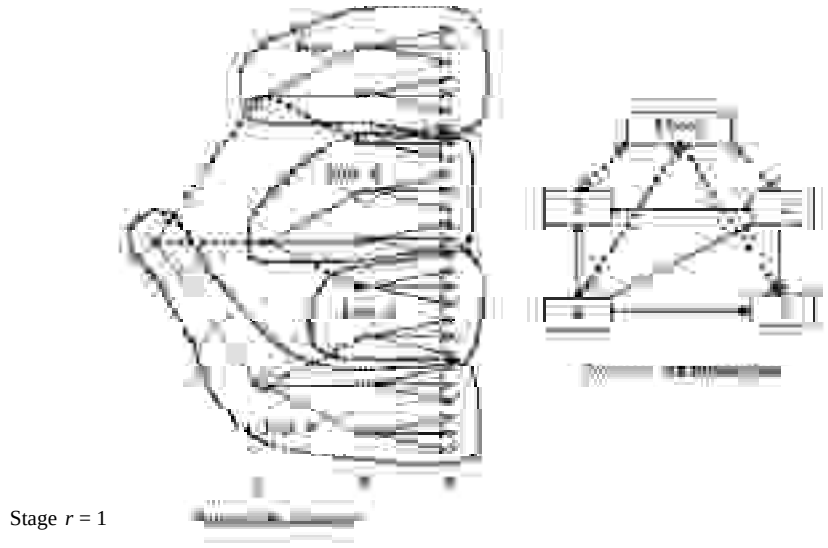

Figure 2. Task allocation and processor communications.

the values of the variables and multipliers at the end of the iteration (in order to evaluate the satisfaction of the termination conditions), and the master sending to the slaves the new values for the multipliers of the complicating constraints at the beginning of the new iteration.

In tables 1 and 2, we present results of the application of these procedures to some of the test problems described in section 6. In table 1, we indicate the number of subproblems assigned to each slave, and also the total number of data interchanges that must be conducted between slaves in each major iteration.

These values should give an indication of the performance of algorithm A-5, that is, if the algorithm behaves as expected, then the number of subproblems allocated to each processor, a measure of the load balancing between processors, should be roughly equal for all processors. Also, the number of data interchanges should be small (it has to be at least $N - 1$) in order to minimize delays for processors waiting to receive information before being able to start solving their assigned subproblems.

Note that a smaller number of data interchanges is closely related to a less homogeneous assignment. It should also be noted that the structure of the scenario tree for all these problems is highly nonhomogeneous, with parts of the tree having a significantly larger number of scenarios than others, and very different numbers of outcomes (children) for different stages and scenarios.

In order to evaluate the practical behavior of algorithm A-5, it is important to also consider the practical results of the allocation since, in addition to the factors included in table 1, the final speedup achieved by the algorithm will also be affected by the total waiting time in each of the slaves. Table 2 shows the time the slaves wait

Table 1

Full recourse. Augmented Lagrangian decomposition.
Subproblem allocation between processors.

Problem	Total	2 WS	3 WS	4 WS
net10-9-30	3848	1892	1298	940
	1956	1312	872	
			1238	1008
				1028
Data int.		3	4	7
net10-15-14	18448	9147	5618	4222
		9301	6402	4850
			6428	4681
				4695
Data int.		3	5	6
net35-60-7	385	188	128	96
		197	129	98
			128	95
				96
Data int.		2	9	8
net35-60-10	4586	2269	1403	1051
		2317	1594	1165
			1589	1194
				1176
Data int.		4	6	10

Legend: Total : total number of subproblems (cardinality of the scenario tree).
x WS : number of subproblems allocated to each node for *x* nodes.
Data int. : information interchanges between nodes at each Lagrangian iteration.

to receive information from the master at the beginning of each iteration, plus the time spent waiting for information from other slaves, accumulated for all iterations in the algorithm.

It is remarkable to note how small the total waiting time is compared to the total (sequential) execution time. It is clear from these results that the communication delays are not very significant, and the load assigned to each slave is fairly well balanced. Also, the assumption that the solution times for the different subproblems are very similar is clearly supported by these results.

Table 2

Total processor waiting time (seconds).

Problem	m	n	$\|S\|$	# Subp	Total	2 WS	3 WS	4 WS
net10-9-30	38480	71572	154	3848	691	13	13	50
						1	10	16
							4	12
								2
net10-15-14	184480	368850	9235	18448	2477	24	52	47
						2	36	43
							4	12
								2
net35-60-7	13475	28420	233	385	267	6	6	5
						1	3	5
							2	7
								3
net35-60-10	39585	87705	564	1131	1130	39	69	76
						5	45	54
							4	53
								7

Legend: Total : total sequential solution time for the problem.
x WS : waiting time for each node, x nodes.
m : number of constraints.
n : number of variables.
$|S|$: number of scenarios.
Subp : total number of subproblems.

6. Computational results

In this section, we present the results obtained from the implementation of the algorithm described in the preceding sections. Our test set (see below) is composed of real-life problems with network structure in the constraints. We have written a C code, SPNET, implementing the Augmented Lagrangian algorithm A-1/A-2, to run on a distributed computation environment. In order to solve the quadratic network subproblems generated in algorithm A-2, we have also developed a C code, QPNET, based on the preconditioned variable reduction truncated Newton approach described in [12,16], see also [40,42], where the search direction is obtained from an approximate solution of the Newton equations by using a preconditioned conjugate gradient algorithm whose preconditioner has been obtained from the diagonal of the reduced Hessian matrix. The C codes use the HP-UX "cc" compiler, and the parallel code was prepared using PVM version 3.3.7.

The instances in the problem set are taken from the hydropower generation management field (see for example [2]). In this case the matrices A_k correspond to the

Table 3

Problem specifications. Deterministic version.

	BN			RN	
Problem	*m*	*n*	*T*	*m*	*n*
Set 1. Linear programs					
red4-3-7	4	3	7	28	45
red4-3-10	4	3	10	40	66
red4-3-30	4	3	30	120	206
red10-9-6	10	9	6	60	104
red10-9-7	10	9	7	70	123
red10-9-14	10	9	14	140	256
red10-9-30	10	9	30	300	560
red10-9-50	10	9	50	500	940
red10-15-14	10	15	14	140	340
red25-35-12	25	35	12	300	695
red25-41-5	25	41	5	125	305
red25-41-6	25	41	6	150	371
red25-41-7	25	41	7	175	437
red25-41-10	25	41	10	250	635
red25-41-12	25	41	12	300	767
red35-52-12	35	52	12	420	1009
red35-60-5	35	60	5	175	440
red35-60-6	35	60	6	210	535
red35-60-7	35	60	7	245	630
red35-60-10	35	60	10	350	915
red35-60-12	35	60	12	420	1105
Set 2. Linear programs					
net10-9-6	10	9	6	60	104
net10-9-7	10	9	7	70	123
net10-9-30	10	9	30	300	560
net10-9-50	10	9	50	500	940
net10-15-14	10	15	14	140	340
net10-9-14	10	9	14	140	256
net35-60-5	35	60	5	175	440
net35-60-6	35	60	6	210	535
net35-60-7	35	60	7	245	630
net35-60-10	35	60	10	350	915
net35-60-12	35	60	12	420	1105
net35-52-12	35	52	12	420	1009

continues . . .

Table 3 (continued)

	BN			RN	
Problem	*m*	*n*	*T*	*m*	*n*
Set 3. Nonlinear programs					
nl9-8-10	9	8	10	80	162
nl9-8-11	9	8	11	88	179
nl9-8-1	9	8	12	96	196
nl9-8-13	9	8	13	104	213
nl9-8-14	9	8	14	112	230
nl9-13-11	9	13	11	143	229
nl9-13-12	9	13	12	156	251
nl9-13-13	9	13	13	169	273
nl9-13-14	9	13	14	182	295
nl9-13-15	9	13	15	195	317
nl9-13-16	9	13	16	208	339

Legend: BN : basic network.
RN : replicated network.
m : number of constraints (nodes).
n : number of variables (arcs)
T : number of periods (stages).

node–arc incidence matrix of a given basic network (representing the reservoir network of a river basin), that is replicated for a certain number of time periods, and the coefficients c and b are random parameters. The sizes of the basic and replicated networks for the test problems are shown in table 3. There are three sets of problems, two sets corresponding to linear problems, and one set of nonlinear problems whose objective function is a polynomial function of degree 12 or higher. The parameter values that we have used throughout our computational experiments (except where we indicate otherwise) are as follows: $\varepsilon = 10^{-5}$, $\beta = 1$, $\gamma = 1$ and $\rho = 1$.

The number of scenarios, the number of subproblems (i.e. the cardinality of the scenario tree) and the size of the compact representation for model (5) (see an instance in model (6)) corresponding to each test problem are shown in table 4. All subproblems in the decomposition scheme have been defined so that each stage corresponds to a single time period. Given the interdependencies between the values of the uncertain parameters related to consecutive periods for the problem that motivates this work, the resulting scenario trees are highly asymmetric. Nevertheless, this property of the problem we have used to obtain our computational results does not impact our algorithm approaches.

Table 4

DE model specifications. Full recourse version (compact representation).

Problem	$\|S\|$	# Subp	m	n	nel
Set 1. Linear programs					
red4-3-7	233	385	1540	1765	4454
red4-3-10	3625	6027	24108	27689	69874
red4-3-30	154	3848	15392	26320	53252
red10-9-6	96	152	1520	1928	4806
red10-9-7	233	385	3850	4985	12290
red10-9-14	9235	18448	184480	258162	608664
red10-9-30	154	3848	38480	71572	144674
red10-9-50	154	6928	69280	130092	261714
red10-15-14	9235	18448	184480	368850	830040
red25-35-12	2301	4586	114650	217635	492770
red25-41-5	34	56	1400	2846	6517
red25-41-6	96	152	3800	7632	17639
red25-41-7	233	385	9625	19585	44970
red25-41-10	564	1131	28275	60546	135167
red25-41-12	2301	4586	114650	217635	457802
red35-52-12	2301	4586	160510	318447	717394
red35-60-5	15	32	1120	2515	5520
red35-60-6	96	152	5320	11080	25485
red35-60-7	233	385	13475	28420	64960
red35-60-10	564	1131	39585	87705	195115
red35-60-12	2301	4586	160510	355135	790770
Set 2. Linear programs					
net10-9-6	96	152	1520	1928	4806
net10-9-7	233	385	3850	4985	12290
net10-9-30	154	3848	38480	71572	144674
net10-9-50	154	6928	69280	130092	261714
net10-15-14	9235	18448	184480	368850	830040
net10-9-14	9235	18448	184480	258162	608664
net35-60-5	15	32	1120	2515	5520
net35-60-6	96	152	5320	11080	25485
net35-60-7	233	385	13475	28420	64960
net35-60-10	564	1131	39585	87705	195115
net35-60-12	2301	4586	160510	355135	790770
net35-52-12	2301	4586	160510	318447	717394

continues . . .

Table 4 (continued)

Problem	$\|S\|$	# Subp	m	n	nel
Set 3. Nonlinear programs					
nl9-8-10	564	1131	10179	14151	33369
nl9-8-11	1154	2285	20565	28459	67295
nl9-8-12	2301	4586	41274	57253	135206
nl9-8-13	4627	9213	82917	114978	271590
nl9-8-14	9235	18448	166032	230501	544108
nl9-13-11	311	751	6759	13723	30236
nl9-13-12	569	1320	11880	23919	52950
nl9-13-13	997	2317	20853	42001	92966
nl9-13-14	1729	4046	36414	73451	162454
nl9-13-15	3036	7082	63738	128480	284275
nl9-13-16	5296	12378	111402	224652	496959

Legend: $|S|$: number of scenarios.
Subp : total number of subproblems.
m : number of constraints.
n : number of variables.
nel : number of nonzero entries in the constraint matrix.

Table 5 shows the comparison for problem set 1 (linear problems) between an interior point code (OB1 [28]) for the solution of the full recourse version (splitting variable representation), our Benders decomposition algorithm and our Augmented Lagrangian algorithm. The comparison has been conducted running OB1 on an HP 730 with 32 Mb of internal memory, and running the two decomposition implementations on an HP 720 (16 Mb of internal memory). These workstations have theoretical ratings of 23 and 17 Mflops respectively; for the C complier used, their effective LINPACK 100 × 100 ratings were 11 and 8.5 Mflops, respectively. The results show the superiority of the Augmented Lagrangian approach, and the good properties of this approach as the dimension of the problem increases, both in terms of the storage space required, and the running time to compute a solution. We should also note the remarkable performance of OB1 while the dimensions of the problems are sufficiently small. For this table, the solution was computed to an accuracy of 5 correct decimal places. Results from a parallel implementation of the code on a network of 3 workstations (two HP 720's and one HP 730) are also included.

Table 6 shows additional experiments along the lines of comparing different procedures for solving in a sequential environment the full recourse model (5) with a splitting variable representation for problem set 2 (linear programs). This comparison has been conducted to provide a reference for both the sequential and parallel versions of our Augmented Lagrangian decomposition algorithm, regarding the behavior of

Table 5

Set 1. Full recourse, splitting variable representation. Computational results.

	OB1		SBD		SLD		PLD
Problem	nit	time	nit	time	nit	time	time
red4-3-7	21	0:07	11	1:30	1010	2:14	1:48
red4-3-10					906	30:49	13:00
red4-3-30	29	1:51			122	3:12	1:31
red10-9-6	22	0:09	22	2:45	910	2:42	1:54
red10-9-7	25	0:19			726	5:42	2:58
red10-9-14					1430	8:49:23	4:54:38
red10-9-30					322	24:08	12:17
red10-9-50					337	45:08	22:34
red10-15-14					254	1:33:16	45:48
red25-35-12					871	4:14:59	2:13:05
red25-41-5	27	0:18	69	13:42	214	0:46	0:29
red25-41-6	28	0:57			468	4:02	1:57
red25-41-7	33	3:22			266	6:21	3:08
red25-41-10					321	22:15	12:44
red25-41-12					443	2:03:23	1:14:23
red35-52-12					610	4:31:57	2:32:32
red35-60-5	28	0:25	> 80	> 1:00:00	167	0:52	0:39
red35-60-6	31	1:41			407	5:47	2:34
red35-60-7	36	6:31			319	11:21	5:18
red35-60-10					463	47:39	26:15
red35-60-12					408	2:51:24	1:31:31

Legend: SBD : Benders decomposition.
SLD : Sequential Augmented Lagrangian decomposition.
PLD : Parallel Augmented Lagrangian decomposition.
nit : number of iterations.
time : running time.

efficient codes for the sequential solution of our problems of interest. The sequential codes used were LoQo (see [43]) and OB1, two efficient interior point codes. Simplex type algorithms seem to behave poorly on problems with the structure described in this work. The termination conditions for all three codes were to stop when five significant digits in the objective function had been correctly determined. The comparison was conducted on an HP 735/125 workstation (rated at 65 Mflops) with 90 Mb of internal memory. From the results in the table, it should be noted that, while the running times (as given in seconds) for the decomposition algorithm are in general larger than those for the interior point methods, they are within the same order of magnitude, and the differences tend to decrease with problem size. The missing entries

Table 6

Set 2. Full recourse, splitting variable representation.
Sequential comparison, running times.

Problem	SLD	LoQo	OB1
net10-9-6	22.5	2.8	1.4
net10-9-7	49.8	9.1	4.7
net10-9-30	264.2	129.9	82.8
net10-9-50	490.0	–	–
net10-15-14	1319.2	–	–
net10-9-14	5009.5	–	–
net35-60-5	8.3	4.3	2.4
net35-60-6	48.7	48.8	32.3
net35-60-7	102.8	274.0	125.8
net35-60-10	436.6	–	–
net35-60-12	1590.2	–	–
net35-52-12	2608.8	–	–

correspond to those cases that exceeded the resources (internal memory) available in the workstation. In this sense, the decomposition code is far more parsimonious in the use of computational resources than both interior point codes. Note that problem net10-15-14 (compact dimensions $m = 184480$ and $n = 368850$) was solved in less than 22 minutes by using our SLD approach.

Tables 7 and 8 show the computational running times for both the sequential and parallel versions of the Augmented Lagrangian code, on problem set 2 (linear programs). The sequential results were obtained on an HP 720 workstation, while the parallel version uses three HP 720's and one HP730 (this last workstation was used only for the results in the Par. 4 WS column). The two tables correspond to different strategies for the updating of the parameters, and the termination tolerance. In table 7, the value of ρ has been kept constant throughout the algorithm, and the solution was computed to a relative precision of 5 decimal places in the objective function. In table 8, the strategy to modify the penalty parameter described in section 4.2 has been used, and the solution process was terminated after two correct digits had been identified for the objective function.

We note the satisfactory results that have been obtained by using the sequential versions of the algorithm. Also, the closeness of the speedup ratios for the large problems to the theoretical maximum, proves that a parallel implementation can be very efficient for using Augmented Lagrangian decomposition approaches to solve large stochastic network problems. Note that a comparison of the times in tables 7 and 8 implies a clear advantage for the parallel code over any sequential alternative, particularly for large problems, even when just two processors are used. Finally, it is

Table 7

Set 2. Augmented Lagrangian code. Fixed ρ = 1. Sequential vs. parallel computation.

Problem	nit	Seq	Par.2WS	SpUp	Par.3WS	SpUp	Par.4WS	SpUp
net10-9-6	910	0:01:04	0:00:45	1.42	0:00:37	1.73	0:00:32	2.00
net10-9-7	726	0:02:17	0:01:20	1.71	0:01:01	2.25	0:00:48	2.85
net10-9-30	322	0:11:32	0:06:20	1.82	0:04:20	2.66	0:03:24	3.39
net10-9-50	337	0:21:35	0:11:52	1.82	0:08:03	2.68	0:06:19	3.42
net10-15-14	255	0:41:42	0:22:39	1.84	0:15:43	2.65	0:11:33	3.61
net10-9-14	1430	3:41:53	1:58:21	1.87	1:22:11	2.70	1:00:10	3.69
net35-60-5	265	0:00:20	0:00:16	1.25	0:00:13	1.54	0:00:10	2.00
net35-60-6	407	0:02:08	0:01:14	1.73	0:00:53	2.42	0:00:41	3.12
net35-60-7	319	0:04:28	0:02:26	1.84	0:01:41	2.65	0:01:17	3.48
net35-60-10	463	0:18:53	0:10:05	1.87	0:07:01	2.69	0:05:13	3.62
net35-60-12	408	1:09:00	0:35:37	1.94	0:24:53	2.77	0:18:45	3.68
net35-52-12	610	1:51:09	0:56:23	1.97	0:39:01	2.85	0:29:26	3.78

Legend: nit : number of major (Lagrangian) iterations.
Seq : running time for the sequential code.
Par.*x* WS : running time for the parallel code on *x* workstations.
SpUp : speedup factor for the corresponding number of workstations.

Table 8

Set 2. Augmented Lagrangian code. Variable ρ. Sequential vs. parallel computation.

Problem	nit	Seq	Par.2WS	SpUp	Par.3WS	SpUp	Par.4WS	SpUp
net10-9-6	44	0:00:06	0:00:05	1.20	0:00:04	1.50	0:00:04	1.50
net10-9-7	62	0:00:21	0:00:12	1.75	0:00:09	2.33	0:00:07	3.00
net10-9-30	35	0:03:26	0:01:55	1.79	0:01:23	2.48	0:01:02	3.32
net10-9-50	33	0:06:01	0:03:20	1.81	0:02:22	2.54	0:01:50	3.28
net10-15-14	25	0:09:36	0:05:52	1.64	0:04:02	2.38	0:03:02	3.16
net10-9-14	96	0:25:34	0:13:48	1.85	0:09:42	2.64	0:07:15	3.53
net35-60-5	24	0:00:08	0:00:07	1.14	0:00:05	1.60	0:00:04	2.00
net35-60-6	42	0:00:38	0:00:22	1.73	0:00:17	2.24	0:00:13	2.92
net35-60-7	53	0:01:49	0:01:00	1.82	0:00:43	2.53	0:00:33	3.30
net35-60-10	41	0:05:24	0:02:57	1.83	0:02:05	2.59	0:01:36	3.38
net35-60-12	50	0:23:37	0:12:31	1.89	0:08:54	2.65	0:06:43	3.52
net35-52-12	79	0:35:57	0:18:07	1.98	0:12:28	2.88	0:11:38	3.09

Legend: nit : number of major (Lagrangian) iterations.
Seq : running time for the sequential code.
Par.*x* WS : running time for the parallel code on *x* workstations.
SpUp : speedup factor for the corresponding number of workstations.

important to consider that the sequential nature of the interior point codes offers far less scope for their efficient parallel implementation than the decomposition codes that we have described in the paper. Nevertheless, a computational comparison between our Augmented Lagrangian decomposition (LD) approach and recent approaches for the two-stage stochastic problem [8,9,46] based on decomposing the system of linear equations to be solved at each interior point iteration would be useful for a comprehensive assessment of the LD performance.

Finally, table 9 shows the computational running times for both the sequential and parallel versions of the Augmented Lagrangian code for problem set 3 (nonlinear problems) described in table 4. The computer environment for tables 7 and 8 is used again for this table. Here, the value of ρ has been kept constant, and the convergence

Table 9

Set 3. Augmented Lagrangian code. Fixed $\rho = 1$. Sequential vs. parallel computation.

Problem	nit	Seq	Par.2WS	SpUp	Par.3WS	SpUp	Par.4WS	SpUp
nl9-8-10	88	0:05:56	0:03:01	1.97	0:02:07	2.80	0:01:34	3.79
nl9-8-11	86	0:10:50	0:05:41	1.91	0:04:01	2.70	0:02:59	3.63
nl9-8-12	113	0:28:49	0:14:13	2.03	0:09:58	2.89	0:07:23	3.90
nl9-8-13	121	0:58:18	0:35:54	1.62	0:20:50	2.80	0:15:48	3.69
nl9-8-14	179	2:49:18	1:25:57	1.97	1:00:02	2.82	0:45:09	3.75
nl9-13-11	47	0:10:20	0:05:42	1.81	0:04:11	2.47	0:03:12	3.23
nl9-13-12	54	0:20:42	0:11:30	1.80	0:08:02	2.58	0:06:22	3.25
nl9-13-13	59	0:37:24	0:19:37	1.91	0:14:24	2.60	0:10:36	3.53
nl9-13-14	68	1:06:27	0:35:53	1.85	0:25:14	2.63	0:19:27	3.42
nl9-13-15	80	2:04:48	1:06:38	1.87	0:47:08	2.65	0:35:52	3.48
nl9-13-16	71	3:46:36	1:49:29	2.07	1:17:17	2.93	0:57:28	3.94

Legend: nit : number of major (Lagrangian) iterations.
Seq : running time for the sequential code.
Par.*x* WS : running time for the parallel code on *x* workstations.
SpUp : speedup factor for the corresponding number of workstations.

criterion has been to stop when two significant digits in the objective function have been computed. Again, the sequential results are quite satisfactory, and the speedup ratio is very close to the theoretical maximum for the whole set of problems. (Note: the reported speedups 2.03 and 2.07 exceed the theoretical maximum due to system overheads and roundings.

7. Conclusions

In this paper, we have presented a procedure for the solution of linear and nonlinear problems under uncertainty, where this uncertainty is treated via scenario

analysis and a splitting variable representation of the model. This methodology results in a huge deterministic equivalent (DE) model (with hundreds of thousands of variables and constraints) where the constraint structure has been lost. Nevertheless, some blocks of constraints still retain this structure.

A decomposition framework is considered for the solution of these models, based on an Augmented Lagrangian decomposition approach, allowing the solution of the model via separable quadratic (or general nonlinear) approximations of the subproblems (of small to moderate size) that retain the original structure in the constraints.

The separability of the subproblems, and the reduced overhead required for the parameter updating, have motivated the development of a parallel version of the decomposition code. This code (and its sequential version) has been tested on a collection of large problems obtained from the hydropower management field, and compared to efficient alternatives for the solution of the DE model.

The results show the sequential version of the decomposition approach to be comparable to the LP codes that solve the DE model, and the parallel version to be significantly superior. Also, we show that the efficiency of the parallelization is very high, as measured by its speedup factor, particularly for very large problems.

In summary, we have shown that for a family of problems (linear and nonlinear networks under uncertainty) a solution method of choice seems to be the application of an Augmented Lagrangian decomposition approach on a distributed environment.

Some issues still remain to be studied in greater detail, such as the improvement of certain aspects of our Augmented Lagrangian algorithm (updates of parameters), and the development of alternatives for the static load balancing scheme for heavily loaded networks or shared (nondedicated) computers.

References

[1] S. Ahn, L.F. Escudero and M. Guignard-Spielberg, On modeling financial trading under interest rate uncertainty, in: *Optimization in Industry 3*, ed. A. Sciomachen, Wiley, New York, 1995, pp.127–144.

[2] M. Alvarez, C.M. Cuevas, L.F. Escudero, J.L. de la Fuente, C. Garcia and F.J. Prieto, Network planning under uncertainty with an application to hydropower generation TOP 2(1994)25–58.

[3] J.F. Benders, Partitioning procedures for solving mixed variables programming problems, Numerische Mathematik 4(1962)238–252.

[4] D.P. Bertsekas, *Constrained Optimization and Lagrange Multipliers*, Academic Press, 1982.

[5] J.R. Birge, Decomposition and partitioning methods for multistage stochastic linear programs, Operations Research 33(1985)1089–1107.

[6] J.R. Birge, Models and model value in stochastic programming, Annals of Operations Research 59(1995)1–18.

[7] J.R. Birge, C.J. Donohue, D.F. Holmes and O.G. Svintsistski, A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs, Report 94-1, Department of Industrial and Operations Engineering, University of Michigan, 1994.

[8] J.R. Birge and D.F. Holmes, Efficient solution of two-stage stochastic linear programs using interior point methods, Computational Optimization and Applications 1(1992)245–276.

[9] J.R. Birge and L. Qi, Computing block-angular Karmarkar projections with applications to stochastic programming, Management Science 34(1988)1472–1479.
[10] J. Czyzyk, R. Fourer and S. Mehrotra, A study of the augmented system and column splitting approaches for solving two-stage stochastic linear programs by interior point methods, ORSA Journal of Computing 7(1995)474–480.
[11] G.B. Dantzig, J. Ho and G. Infanger, Solving stochastic linear programs on a hypercube multicomputer, Report SOL 91-10, Department of Operations Research, Stanford University, California, 1991.
[12] R. S. Dembo, A primal truncated Newton algorithm with application to large-scale nonlinear network optimization, Mathematical Programming Study 31 (1987)43–72.
[13] R.S. Dembo, Scenario optimization, Annals of Operations Research 30(1991)63–80.
[14] M.A.H. Dempster, On stochastic programming II: Dynamic problems under risk, Stochastics 25 (1985)15–42.
[15] M.A.H. Dempster and R.T. Thompson, Parallelization of EVPI based importance sampling procedures for solving multistage stochastic linear programmes on MIMD architectures, *Parallel Optimization Colloquium*, Laboratoire PRiSM, Université de Versailles, Versailles, France, 1996.
[16] L.F. Escudero, Performance evaluation of independent superbasic sets, European Journal of Operational Research 23(1986)343–355.
[17] L.F. Escudero, On solving a nondifferentiable transshipment problem, Investigación Operativa 1(1990)199–211.
[18] L.F. Escudero, Robust portfolio for mortgage based securities, in: *Quantitative Methods, Supercomputers and AI in Finance*, ed. S.A. Zenios, Stanley Thornes, London, 1995, pp. 202–228.
[19] L.F. Escudero, P.V. Kamesam, A.J. King and R.J-B Wets, Production planning via scenario modelling, Annals of Operations Research 43(1993)311–335.
[20] M.I. Gassman, MSLiP: A computer code for the multistage linear programming problem, Mathematical Programming 47(1990)407–423.
[21] B. Golub, M. Holmer, R. McKendall, L. Pohlman and S.A. Zenios, Stochastic programming models for money management, European Journal of Operational Research 85(1995)282–296.
[22] F. Glover, D. Klingman and N.V. Phillips, *Network Models in Optimization and Their Applications*, Wiley, New York, 1992.
[23] J.L. Higle, W.W. Lowe and R. Odio, Conditional stochastic decomposition: An algorithmic interface for optimization and simulation, Operations Research 42(1994)311–322.
[24] R.S. Hiller and J. Eckstein, Stochastic dedication: Designing fixed income portfolios using massively parallel Benders decomposition, Management Science 39(1993)1422–1438.
[25] G. Infanger, *Planning under Uncertainty – Solving Stochastic Linear Programs*, Boyd and Fraser, 1994.
[26] D. Jensen and A.J. King, Linear-quadratic efficient frontiers for portfolio optimization, Report RC-17156. IBM T.J. Watson Research Center, Yorktown Heights, New York, 1991.
[27] F.V. Louveaux, A solution method for multistage stochastic programs with recourse with application to an energy investment problem, Operations Research 28(1980)889–902.
[28] I.J. Lustig, R.E. Marsten and D.F. Shanno, On implementing Mehrotra predictor–corrector interior point method for linear programming, SIAM Journal on Optimization 2(1992)435–449.
[29] I.J. Lustig, J.M. Mulvey and T.J. Carpenter, Formulating stochastic programs for interior point methods, Operations Research 39(1991)757–769.
[30] J.M. Mulvey and A. Ruszczyński, A diagonal quadratic approximation method for large-scale linear programs, Operations Research Letters 12(1992)205–221.
[31] J.M. Mulvey and A. Ruszczyński, A new scenario decomposition method for large-scale stochastic optimization, Operations Research 43(1995)477–490.
[32] J.M. Mulvey, R.J. Vanderbei and S.A. Zenios, Robust optimization of large-scale systems, Operations Research 43(1995)264–281.

[33] J.M. Mulvey and H. Vladimirou, Stochastic networks, optimization models for investment planning, Annals of Operations Research 20 (1989) 187—217.
[34] S.S. Nielsen and S.A. Zenios, A massively parallel algorithm for nonlinear stochastic network problems, Operations Research 41(1993)319–337.
[35] M.C. Noel and Y. Smeers, Nested decomposition of multistage nonlinear programs with recourse, Mathematical Programming 37(1987)131–152.
[36] M.V.F. Pereira and L.M.V.G. Pinto, Multistage stochastic optimization applied to energy planning, Mathematical Programming 52(1991)359–375.
[37] R.T. Rockafellar and R.J-B Wets, Scenario and policy aggregation in optimization under uncertainty, Mathematics of Operations Research 16(1991)119–147.
[38] A. Ruszczyński, Interior point methods in stochastic programming, Report WP-93-8, IIASA, Vienna, 1993.
[39] A. Ruszczyński, Parallel decomposition of multistage stochastic programs, Mathematical Programming 58(1993)201–208.
[40] A. Sartenaer, On some strategies for handling constraints in nonlinear optimization, Ph.D. Thesis, Department of Mathematics, FUNDP, Namur, Belgium, 1991.
[41] G. Stephanopoulos and W. Westerberg, The use of Hestenes' method of multipliers to resolve dual gaps in engineering system optimization, Journal of Optimization Theory and Applications 15(1975) 285–309.
[42] Ph.L. Toint and D. Tuyttens, On large-scale nonlinear network optimization, Mathematical Programming 48(1990)125–159.
[43] R.J. Vanderbei and T.J. Carpenter, Symmetric indefinite systems for interior point methods, Mathematical Programming 58(1993)1–32.
[44] R. Van Slyke and R.J-B Wets, L-shaped linear programs with applications to optimal control and stochastic programming, SIAM Journal on Applied Mathematics 17(1969)638–663.
[45] R.J-B Wets, Large-scale linear programming techniques in stochastic programming, in: *Numerical Techniques for Stochastic Optimization*, eds. Y. Ermoliev and R.J-B Wets, Springer, Berlin, 1988, pp. 65–94.
[46] D. Yang and S.A. Zenios, On a scalable parallel implementation of interior point algorithms for stochastic linear programming and robust optimization, *Parallel Optimization Colloquium*, Laboratoire PRiSM, Université de Versailles, Versailles, France, 1996.
[47] S.A. Zenios (ed.), *Financial Optimization*, Cambridge University Press, Cambridge, 1993.
[48] S.A. Zenios, Massively parallel computation for financial planning under uncertainty, in: *Very Large-scale Computation in the 21st Century*, ed. J.P. Mesirov, SIAM, Philadelphia, 1991, pp. 273–294.
[49] S.A. Zenios, A model for portfolio management under uncertainty for fixed-income securities, Annals of Operations Research 43(1993)337–356.
[50] S.A. Zenios, Asset/liability management under uncertainty for fixed-income securities, Annals of Operations Research 59(1995)77–79.