Parallel implementation of a central decomposition method for solving large-scale planning problems*

J. Gondzio† R. Sarkissian and J.-Ph. Vial

HEC Technical Report 98.1
January 19, 1998, revised August 12, 1999.

Abstract

We use a decomposition approach to solve three types of realistic problems: block-angular linear programs arising in energy planning, Markov decision problems arising in production planning and multicommodity network problems arising in capacity planning for survivable telecommunication networks. Decomposition is an algorithmic device that breaks down computations into several independent subproblems. It is thus ideally suited to parallel implementation. To achieve robustness and greater reliability in the performance of the decomposition algorithm, we use the Analytic Center Cutting Plane Method (ACCPM) to handle the master program. We run the algorithm on two different parallel computing platforms: a network of PC's running under Linux and a genuine parallel machine, the IBM SP2. The approach is well adapted for this coarse grain parallelism and the results display good speed-up's for the classes of problems we have treated.

Keywords Decomposition, Parallel computation, Analytic center, Cutting plane method, Real-life problems.

LOGILAB/Management Studies, University of Geneva, 102, Bd Carl-Vogt, CH-1211 Genève 4, Switzerland.

E-mail: gondzio@maths.ed.ac.uk, sarkissian@hec.unige.ch, and jean-philippe.vial@hec.unige.ch.

*Supported by the Fonds National de la Recherche Scientifique Suisse, grant #12-42503.94.

†On leave from the Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland. Since October 1st, 1998 this author is at the Department of Mathematics and Statistics, University of Edinburgh.

1 Introduction

Despite all recent progresses in optimization algorithms and in hardware, model builders keep on generating larger and larger optimization models that challenge the best existing technology. This is particularly true in the area of planning problems where the quest for more realism in the description of the systems dynamics or the uncertainty, naturally yields huge models. One way to increase the computation power is to resort to parallel computations. However, there are serious obstacles to parallel implementations of optimization algorithms. First, genuine parallel machines are still very expensive; most users don't have access to such high technology equipment. Secondly, adapting optimization softwares to parallel computations is often quite complicated; again, this is a real issue for most practitioners.

In this paper we want to demonstrate that for some classes of problems, there exist solutions that do not require sophisticate hardware, nor major adaptations of existing software. Yet, they achieve interesting speed-up's that allow to solve problems that would be intractable on standard sequential machines with regular optimization software. We used three key ingredients. The first idea is to use decomposition to break the initial problem into many independent smaller size problems that can be simultaneously solved on different processors. The second idea is to use clusters of dedicated PC's to get a loose parallel environment. Finally, to overcome possible instability in the decomposition process, we base the decomposition scheme on the analytic center cutting plane method. Our second goal in this paper is to demonstrate that the parallel implementation of the central decomposition scheme scales well.

To illustrate the case, we consider three classes of realistic problems. Some problems in those classes are so large that they do not seem solvable by a direct sequential approach, even on the most powerful workstations. The first class of problems pertains to energy planning: the problems are formulated as block-angular linear programs which are solved via Lagrangian relaxation. The other class deals with the planning of excess capacity to achieve survivability in telecommunication networks. The capacity problems are formulated as structured linear programs that are solved by Benders decomposition scheme. Finally, we considered few more block-angular linear programming problems which arise in the area of Markov decision problems. Even though the problem sizes are not excessive, the problems are numerically challenging and thus make up interesting test problems.

To achieve efficiency with parallel computations, one needs to organize computations so as to have all processors simultaneously busy. In a decomposition approach, computations alternate between the master problem and the subproblems. Thus, unless one could implement parallel computations for the master problem itself, all processors but one are idle when the code deals with the master program. Consequently, a necessary condition for efficient implementation is to work on problems where the master program is comparatively small with respect to the subproblems. The applications we present in this paper are of that type.

The idea to use parallel computation via decomposition is quite natural and is certainly not new [7, 11]. However, there are not so many reports of successful implementations, probably because of the bad reputation of standard decomposition schemes. Dantzig-Wolfe decomposition [9], or its dual counterpart Benders decomposition [6] are reputed to be unstable. Both schemes can be viewed as a special implementation of the classical Kelley-Cheney-Goldstein

[26, 8] cutting plane algorithm. Though Kelley's cutting plane method often performs well, it is also known to be very slow and even to fail on some instances. Nemirovskii and Yudin [35] devised an illuminating example based on a resilient oracle that confirms the possibly slow convergence of Kelley's method. To remedy this flaw, various schemes have been proposed in the literature: some are based on a regularized scheme [38, 27, 29]; others use central query points such as Levin's center of gravity, the center of the maximum inscribed ellipsoid, the volumetric center and, lastly, the analytic center.

In this paper we use the last method. The use of the analytic center was suggested by Huard [23] in his celebrated *linearized method of centers*. Many years later the idea surfaced again: Renegar's [36] polynomial path-following algorithm for LP[1]. Sonnevend [42] suggested to use analytic centers in connection with general convex programming. Several authors realized the potential of this concept in relation with the cutting plane approach to nondifferentiable optimization [43, 17] or mixed integer programming [33]. The idea has been refined and implemented in a sophisticate code [20], named ACCPM (analytic center cutting plane method). Its current implementation is based on the projective algorithm [25, 15]. Under extensive testing, ACCPM has proved to be powerful and robust [3, 16]. We consider it a promising candidate for a parallel implementation.

In this paper, we address the parallel treatment of subproblems. In our computations we used two very different parallel machines: a cluster of 16 Pentium Pro PC's and the IBM SP2 of the University of Geneva with 8 available nodes for scientific parallel experimentations. The first system is particularly attractive, due to its very low cost and its ease of installation. The second system is a commercial machine optimized for parallel computations.

The paper is organized as follows. In Section 2 we briefly introduce three decomposable applications: energy planning problems and Markov decision models that are both well adapted to Dantzig-Wolfe decomposition; telecommunications network survivability problems that suit Benders decomposition. In Section 3, we address the issues of decomposition approaches specialized to these two classes of problems. In Section 4, we give a unified view of cutting plane methods to solve decomposed problem and two specialized algorithms: the classical Kelley-Cheney-Goldstein [26, 8] cutting plane method and the analytic center cutting plane method. In Section 5, we address the basic issues in the parallel implementation of the decomposition scheme when applied to the three applications presented earlier. In Section 6.1, we present our numerical results. Finally, in Section 7 we give our conclusions.

2 Application problems

2.1 Energy planning and CO_2 abatement

MARKAL (MARKet ALlocation) is a standard energy systems model. It has been developed under the aegis of the International Energy Agency and is currently implemented in more than 40 countries [4]. MARKAL is a multi-period linear programming model. The objective function in the LP is the discounted sum, over the horizon considered (usually between 30 and 45 years), of investment, operating and maintenance costs of all technologies, plus the cost of energy imports. The minimization of this objective function, subject to the constraints

[1] LP is the acronym for linear programming. We shall use it throughout the paper.

describing the energy system gives an optimal investment plan for the selected technologies.

An extension consisted to link existing MARKAL models of several countries (Belgium, Great Britain, Germany, Switzerland) under the joint constraint of a maximum CO_2 emission in accordance with the recommendation of the Environment and Development Conferences in Rio (1992) and in Kyoto (1997). This coupling of models provides a basis for discussing the impact of a tax on pollution emittant and also fair transfers between countries [4]. The individual MARKAL models are linear programming problems with 5,000 to 10,000 constraints and up to 20,000 variables. Linking few such models creates large block-structured linear programs. The algebraic formulation of this class of problems takes the form

$$\begin{array}{lll} \text{maximize} & \sum_{i=1}^{p} \langle c_i, x_i \rangle & \\ \text{subject to} & \sum_{i=1}^{p} A_i x_i = a & \\ & B_i x_i = b_i, & i = 1, 2, \ldots, p, \\ & x_i \geq 0, & i = 1, 2, \ldots, p, \end{array} \tag{1}$$

where $c_i, x_i \in \mathcal{R}^{n_i}, i = 1, \ldots, p,\ a \in \mathcal{R}^{m_0},\ b_i \in \mathcal{R}^{m_i}, i = 1, \ldots, p$, and all matrices A_i, $i = 1, \ldots, p$, and B_i, $i = 1, \ldots, p$, have appropriate dimensions. The matrices A_i relate to the common resource: the CO_2 emission. The matrices B_i are the country MARKAL models. The structure of this global problem is named *primal block-angular*. It is made up of several independent blocks of constraints, and of a small set of constraints that couple all variables.

Similar problems have also been studied in [34] where MESSAGE models were used. These linear programs were even larger as a single problem has a size of over 40,000 rows and 73,000 variables. The coupling of 8 such models yields a linear program (of about 330,000 rows and 600,000 variables) that goes beyond the limits of the standard, state-of-the-art software for workstations.

Multiregional planning problems suit quite well the type of parallelism we want to use. They are loosely coupled (7 to 20 linking constraints) and have small number (3 to 8) of subproblems, each of considerable size (up to 40,000 rows). In a sequential implementation time spent in the master problem remains only a small fraction—about 2%—of the overall solution time. Unfortunately, subproblems may differ considerably in their sizes and in the predicted computational effort to solve them.

2.2 A Markov decision model

The second test problem in the class of block-angular linear programs was generated from a Markov decision problem for systems with slow and fast transition rates. Abbad and Filar [1] showed how one can reformulate such problems as block-angular problems. Filar and Haurie [12] applied this approach to manufacturing problems. The particular problems we solved were communicated to us by Day, Haurie and Moresino. The problems have 16 subproblems and 17 coupling constraints. The problems we solved are not immense. However, they are numerically difficult and unstable. The variables are probabilities that sometimes take very small values, but, in the mean time, those small values have a very large impact on the global solution. State-of-the-art codes applied in a frontal manner (i.e., without decomposition) encounter serious difficulties even for smaller instances of these problems. The problems we solved are not real application problems. They were included in our experiments as being particularly challenging.

2.3 Survivablity in telecommunications network

Survivability is a critical issue in telecommunications. Individual components of a telecommunications network are prone to failure. Those events interrupt traffic and cause severe damage to the customers. A network is survivable if it can cope with failures. This raises a number of issues regarding the topology of the network, its capacity and the message rerouting policy in case of a failure. In the present case, we are interested in the planning of capacity on a network with a fixed topology. This facet of the telecommunications survivability problem was first presented by Minoux [31] who formulated it as a large linear programming model under the following assumption: when a failure occurs, the entire traffic may be changed to restore feasibility through planned excess capacity. However, telecommunications operators have a strong preference for minimal changes in the traffic assignment. In [30], Lisser et al. considered the same model with the proviso that reroutings are restricted to that segment of the traffic that was interrupted by the failure. They solved the optimization problem via a Lagrangian relaxation. Their approach does not lead to easy parallel implementation, because the master program is very large and computationally much more demanding than the subproblems. We shall present in the subsequent sections an alternative decomposition approach. For a survey on related survivability problems, see [2].

We give now a formal description of the problem. A telecommunications network is an undirected graph $G = (V, E)$, without loops and parallel edges. In the normal operational state, the traffic assignment meets regular demands between origin-destination pairs and is compatible with the capacity of the network. We consider that the initial traffic assignment in normal condition is given and fixed.

When some component of the network—edge or node—fails, the traffic through this component is interrupted: it must be rerouted through alternative routes in the network. We consider that at most one component can fail at a time. We denote by S the set of failure states. Clearly, the cardinality of S cannot exceed[2] $|V| + |E|$. For a state $s \in S$, the operational part of the network consists of a subnetwork $G(s) = (V(s), E(s))$ of G of working components. When the state s originates from the failure of the node $v \in V$, then $G(s)$ is the subgraph $G \setminus v$ of G; when it originates from the failure of the arc $a \in E$, then $G(s)$ is the subgraph $G \setminus a$ of G. For each state, one defines, from the initial traffic assignment, a set of new demands $k \in \mathcal{R}_s$ between pairs of nodes. Those conditional demands are created by the traffic implicated in the failures; they are described, for each $k \in \mathcal{R}_s$, by a vector $b_k \in \mathbb{R}^{|E(s)|}$ and a single origin/destination pair.

Let $K^{(s)} = (K^{(s)}_{(i,j)})_{[i,j]\in E(s)}$ be the vector of unused capacity in the normal operational state. This unused capacity can be utilized to route the conditional demands. If necessary, more capacity $y = (y_{ij})_{[i,j]\in E}$ can be installed at a given positive unitary cost $c = (c_{ij})_{[i,j]\in E}$. The objective of the capacity planning problem consists in finding the least capacity investment cost that allows the rerouting of all conditional demands $k \in R_s$ for each state $s \in S$.

The standard flow model applies to directed graphs. In that case, the mass-balance equations establish an algebraic relation between in-flow and out-flow. However, telecommunications networks are essentially undirected graphs: messages can travel on a link $[i, j]$ in either directions. To retrieve the usual flow formulation on directed graphs, we replace each arc $[i, j]$ by a pair of opposite arcs (i, j) and (j, i), but we must remember that the capacity

[2] This is to account for the possible case of some edge with no assigned traffic.

usage of an arc is the sum of the flows on the two opposite arcs. This is the basis for our problem formulation.

Given an arbitrary orientation on the graph G, we define for each $s \in S$ the node-arc adjacency matrix N_s of $G(s)$. Then, the matrix $(N_s, -N_s)$ is the adjacency matrix of the graph formed of the two graphs $G(s)$ and the one opposite to it. Let x_{ij}^k be the flow component of commodity k on the arc (i,j). The mass balance equations involve the vectors $(x_{ij}^k)_{(i,j)\in E(s)}$ and $(x_{ji}^k)_{(i,j)\in E(s)}$ which represent opposite flow components. With these definitions, one can state our problem as follows:

$$\min \sum_{[i,j]\in E} c_{ij} y_{ij}$$

$$\text{s.t.} \sum_{k\in\mathcal{R}_s} x_{ij}^k + x_{ji}^k - y_{ij} \leq K_{ij}^s, \quad \forall (i,j) \in E(s), \forall s \in S, \tag{2a}$$

$$(N_s, -N_s) \begin{pmatrix} (x_{ij}^k)_{(i,j)\in E(s)} \\ (x_{ji}^k)_{(i,j)\in E(s)} \end{pmatrix} = b_k, \quad \forall k \in \mathcal{R}_s, \forall s \in S, \tag{2b}$$

$$x_{ij}^k \geq 0, \quad \forall k \in \mathcal{R}_s,\ \forall (i,j) \in E(s),\ \forall s \in S, \tag{2c}$$

$$y_{ij} \geq 0 \quad \forall [i,j] \in E(s). \tag{2d}$$

The inequalities (2a) are capacity constraints which impose that in any operational state s the flow through an arc $[i,j]$ does not exceed the capacity. Equations (2b) are the node balance equations for each demand k resulting of an operational state $s \in S$.

3 Problem reformulation via decomposition

Decomposition often means two different things. Firstly, decomposition is a mathematical technique to transform an initial large-scale problem into one of smaller dimension. Secondly, decomposition refers to the algorithm that is used to solve the transformed problem. In this section we consider the mathematical transformation only.

The usual transformation techniques in decomposing large-scale optimization problems are the Lagrangian relaxation and Benders decomposition. To paraphrase Lagrangian relaxation, let us assume the problem of interest is a maximization one and that the constraints can be split into two sets: simple constraints and complicating ones. The method goes as follows: the complicating constraints are relaxed, but the objective is modified to incorporate a penalty on the relaxed constraint. The penalty parameters are nothing else than dual variables associated with the relaxed constraints. Solving this problem yields an optimal value that is a function of those dual variables. Under appropriate convexity and regularity assumptions, this function is convex and its minimum value coincides with the optimal one of the original problem. This defines a new problem, equivalent to the original one, but having a variable space of smaller dimension. In the linear programming case, the convexity and regularity assumption hold; besides, the method is just the same Dantzig-Wolfe decomposition.

To paraphrase Benders decomposition, let us assume the problem of interest is a minimization one and that the variables can be split into two sets. By fixing the first set of variables, one obtains a problem whose optimal solution depends on the variables in the first set only. The transformed problem consists in minimizing this function. Under appropriate convexity and

regularity assumptions, this function is convex and its minimum value coincides with the optimal value of the original problem. Again, the minimization of this function defines a new problem that is equivalent to the original one, but has dimension of the first set of variables.

In linear programming, the two schemes are dual of one another. If the first set of constraints (or dually of variables) has small dimension, the transformation yields a new equivalent problem, much smaller in size, but whose components – objective function and/or constraints – are implicitly defined. That is, these components are given by the optimal values (objective and variables) of appropriate optimization problems. The next subsections provide detailed examples of such transformations, and emphasize the similarity of the transformed problem in the two cases.

Even though the new problem has smaller dimension, it is not likely to be easy. Difficulties arise first from the nondifferentiability of the objective: derivatives are replaced by subgradients, which do not provide the same quality of approximation. The second difficulty comes from the computation of the subgradients themselves: they are obtained from the solution of an auxiliary optimization problem. To achieve efficiency and accuracy, one must resort to advanced algorithmic tools. Section 4 describes one such tool.

3.1 Lagrangian relaxation of block-angular programs

A natural class of structured problems eligible to decomposition are large-scale block-angular linear programs of the following form

$$\begin{array}{lll} \text{maximize} & \sum_{i=1}^{p} \langle c_i, x_i \rangle & \\ \text{subject to} & \sum_{i=1}^{p} A_i x_i = a & \\ & B_i x_i = b_i, & i = 1, 2, \ldots, p, \\ & x_i \geq 0, & i = 1, 2, \ldots, p, \end{array} \tag{3}$$

where $c_i, x_i \in \mathcal{R}^{n_i}, i = 1, \ldots, p,\ a \in \mathcal{R}^{m_0},\ b_i \in \mathcal{R}^{m_i}, i = 1, \ldots, p$, and all matrices $A_i, i = 1, \ldots, p$, and $B_i, i = 1, \ldots, p$, have appropriate dimensions.

We associate with (3) the partial Lagrangian function

$$\begin{array}{rcl} L(x, u) & = & \sum_{i=1}^{p} \langle c_i, x_i \rangle + \langle \sum_{i=1}^{p} A_i x_i - a, u \rangle \\ & = & -\langle a, u \rangle + \sum_{i=1}^{p} \langle c_i + A_i^T u, x_i \rangle. \end{array}$$

Assuming problem (3) has an optimal solution, we may, by duality, replace it with

$$\min_{u} L(u), \tag{4}$$

where $L(u)$ is the optimal value of

$$\begin{array}{lll} \text{maximize} & -\langle a, u \rangle + \sum_{i=1}^{p} \langle c_i + A_i^T u, x_i \rangle & \\ \text{subject to} & B_i x_i = b_i, & i = 1, 2, \ldots, p, \\ & x_i \geq 0, & i = 1, 2, \ldots, p. \end{array} \tag{5}$$

In the usual parlance, Problem (4) is the master program while Problem (5) is the subproblem. Note that the feasible set of the subproblem is independent of u. Since the objective is the point-wise maximum of linear functions in u, it is convex in u.

Let us observe that subproblem (5) is *separable*; in consequence, the partial Lagrangian is *additive*

$$L(u) = -\langle a, u\rangle + \sum_{i=1}^{p} L_i(u), \tag{6}$$

where, for $i = 1, \ldots, p$

$$L_i(u) = \max_{x_i} \left\{ \langle c_i + A_i^T u, x_i\rangle \mid B_i x_i = b,\ x_i \geq 0 \right\}. \tag{7}$$

Problem (7) is always feasible, and either has an optimal solution, or is unbounded. Moreover, $L_i(u)$ is convex. Its domain is

$$\mathrm{dom}L_i = \{u \mid L_i(u) < \infty\},\ i = 1, \ldots p.$$

We can now rewrite problem (4)

$$\min\{-\langle a, u\rangle + \sum_{i=1}^{p} L_i(u) \mid u \in \cap_{i=1}^{p} \mathrm{dom}L_i\}.$$

Let us say a word about the oracle. Assume first that $u \in \mathrm{dom}L_i$; let $x_i(u)$ be an optimal solution of (7). Then, the optimality cut follows directly from the definition of L_i in (7)

$$L_i(v) \geq L_i(u) + \langle A_i x_i(u), v - u\rangle,\ \forall v.$$

Assume now that $u \notin \mathrm{dom}L_i$; then, problem (7) is unbounded. Let $d_i(u)$ be a ray along which $\langle c_i + A_i^T u, d_i\rangle$ is unbounded. We then have the feasibility cut

$$\langle A_i d_i(u), v\rangle \leq 0,\ \forall v,$$

that excludes u as not being part of the domain of L_i.

In practical implementations, it might be useful to work with a relaxation of the subproblems. More precisely, assume that the subproblems are solved with an ϵ-precision, $\epsilon > 0$. That is, we generate a point $\bar{x}_i$ that is feasible for the subproblems $i = 1, \ldots, p$ with

$$\langle c_i + A_i^T u, \bar{x}_i\rangle \geq L_i(u) - \epsilon. \tag{8}$$

Thus for an arbitrary v, we have the ϵ-subgradient inequality

$$L_i(v) \geq \langle A_i \bar{x}_i, v - u\rangle + L_i(u) - \epsilon. \tag{9}$$

The ϵ-subgradients can be used to define a weaker LP relaxation. This relaxation has been successfully used [21] in a sequential implementation of the decomposition. We also use it in the parallel implementation.

The application of Lagrangian relaxation to the two block-angular classes of problems that we described in Sections 2.1 and 2.2 is obvious. For the chosen problems, the number of coupling constraints is very small: hence, the variable space of the transformed problem is also very small.

3.2 Benders decomposition

Linear programs addressed in the previous section display a *primal* block-angular structure in which a set of complicating constraints links the independent blocks. A similar formulation can be given in which a subset of variables links the independent blocks. Consider the problem

$$\begin{array}{lll} \text{minimize} & \langle c_0, x_0 \rangle & + \sum_{i=1}^{p} \langle c_i, x_i \rangle \\ \text{subject to} & T_i x_0 & + W_i x_i = b_i, \quad i = 1, 2, \ldots, p, \\ & & x_i \geq 0, \qquad i = 1, 2, \ldots, p, \end{array} \tag{10}$$

where $c_i, x_i \in \mathcal{R}^{n_i}, i = 0, 1, \ldots, p,\ b_i \in \mathcal{R}^{m_i}, i = 1, \ldots, p$, and all matrices $T_i, i = 1, \ldots, p$, and $W_i, i = 1, \ldots, p$, have appropriate dimensions. The constraint matrix of this linear program displays a *dual* block-angular structure.

For a given x_0, let $Q(x_0)$ be the optimal value of the subproblem

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{p} \langle c_i, x_i \rangle & \\ \text{subject to} & W_i x_i = b_i - T_i x_0, & i = 1, 2, \ldots, p, \\ & x_i \geq 0, & i = 1, 2, \ldots, p. \end{array} \tag{11}$$

Then we can replace (10) with

$$\begin{array}{ll} \text{minimize} & \langle c_0, x_0 \rangle + Q(x_0) \\ \text{subject to} & x_0 \geq 0. \end{array} \tag{12}$$

The function $Q(x_0)$ is *additive*

$$Q(x_0) = \sum_{i=1}^{p} Q_i(x_0), \tag{13}$$

where $Q_i(x_0)$, for $i = 1, \ldots, p$, is the value of

$$\begin{array}{ll} \text{minimize} & \langle c_i, x_i \rangle \\ \text{subject to} & W_i x_i = b_i - T_i x_0, \\ & x_i \geq 0. \end{array} \tag{14}$$

In other words, the subproblem (11) is *separable*. It is interesting to consider the dual of (11):

$$\begin{array}{ll} \text{maximize} & \langle b_i - T_i x_0, \pi \rangle \\ \text{subject to} & W_i^T \pi_i \leq c_i. \end{array} \tag{15}$$

The objective value is defined as the point-wise maximum of linear forms in x_0: it is thus convex. Note that the constraints in (15) are independent of x_0. We can safely assume that (15) is feasible, otherwise the original problem is either unbounded or infeasible.

Problem (14) may be infeasible for some values of x_0. Assume first that it is feasible. Then it has an optimal solution $x_i(x_0)$ with value $Q_i(x_0)$. Let $\pi_i(x_0)$ be the dual optimal solution. In view of (15), the optimality cut writes

$$Q_i(y) \geq Q_i(x_0) + \langle -T_i^T \pi_i(x_0), y - x_0 \rangle, \ \forall y.$$

Assume now that (14) is not feasible. Having assumed that the dual (15) is feasible, then it is unbounded. Let $d_i(x_0)$ be a ray along which the dual objective $\langle b_i - T_i x_0, \pi \rangle$ is unbounded, equivalently $\langle b_i - T_i x_0, d_i \rangle > 0$. Then we have the feasibility cut

$$\langle T_i^T d_i(x_0), y \rangle \geq 0, \; \forall y,$$

that excludes $v = x_0$ as not being part of the feasible domain of (10).

Similarly to the Lagrangian relaxation, the exact optimality requirement may be replaced with the ϵ-optimality; we then obtain the weaker inequality

$$Q_i(y) \geq -\langle T_i^T \bar{\pi}_i, y - x_0 \rangle + Q_i(x_0) - \epsilon, \; \forall y.$$

4 Cutting plane methods to solve decomposed problems

In the formulation of the transformed problem, the Lagrangian relaxation and Benders decomposition generate problems that are usually difficult despite the relatively small dimension of the variable space. The difficulty stems from the fact that the problems are essentially nondifferentiable. Their solving requires advanced algorithmic tools. In this section we concentrate on the solution method. More specifically, we present the generic cutting plane method, and its analytic center variant, to solve the nondifferentiable problem of interest.

Cutting plane methods apply to the canonical convex problem

$$\min \{ f(x) \mid x \in X \cap X_0 \}, \tag{16}$$

where $X \subset R^n$ is a closed convex set, $X_0 \subset R^n$ is a compact convex set, $f : R^n \mapsto R$ is a convex function.

We assume that the set X_0 is defined explicitly, while the function f and the set X are defined by the following *oracle*. Given $\bar{x} \in \operatorname{int} X_0$, the oracle answers either one statement:

1. $\bar{x}$ is in X and there is a support vector
   $\xi \in \partial f(\bar{x})$, i.e., $f(x) \geq f(\bar{x}) + \langle \xi, x - \bar{x} \rangle$, $\forall x \in R^n$.
2. $\bar{x}$ is not in X and there is a separation vector a such that
   $\langle a, x - \bar{x} \rangle \leq \gamma$, $\gamma \leq 0$, $\forall x \in X \cap X_0$.

Answers of the first type are named *optimality* cuts, whereas answers of the second type are *feasibility* cuts.

The successive answers of the oracle to a sequence of query points $x^k \in X_0$ can be used to define an outer polyhedral approximation of the problem in the epigraph space. Let $\{x^j\}$, $j \in K = \{1, \ldots k\}$, be the sequence of query points. The set K is partitioned into $K = K_1 \cup K_2$, where K_1 and K_2 correspond to optimality cuts and feasibility cuts respectively. Let the oracle answer at x^j be (d^j, c_j), with $(d^j, c_j) = (\xi^j, f(x^j))$, for $j \in K_1$, and $(d^j, c_j) = (a^j, \gamma^j)$ for $k \in K_2$. Let $\theta^k = \min_{j \in K_1} \{ f(x^j) \}$ be the best recorded function value. The polyhedral approximation defines, in the epigraph space (x, z), the *set of localization*:

$$\begin{aligned} \mathcal{F}_k = \{ (x, z) \mid x \in X_0, \; & z \leq \theta^k, \\ & \langle \xi^j, x - x^j \rangle - z \leq -f(x^j), \; j \in K_1, \\ & \langle a^j, x - x^j \rangle \leq \gamma^j, \; j \in K_2 \big\} . \end{aligned}$$

Clearly, $\mathcal{F}_k$ contains the optimal solutions of the initial problem.

The k-th step of the generic cutting plane algorithm is as follows.

1. Pick $(x^k, z^k) \in \mathcal{F}_k$.
2. The oracle returns the generic cut $\langle d^k, x\rangle - \delta^k z \leq c_k$, where $\delta^k = 1$ if the cut is optimality one, and 0 otherwise.
3. Update $\mathcal{F}_{k+1} := \mathcal{F}_k \cap \left\{(x,z) \mid \langle d^k, x\rangle - \delta^k z \leq c_k\right\}$.

Specific cutting plane algorithms differ in the choice of the query point x^k. Let us focus on two strategies. Kelley's strategy consists in selecting $(x,z) = \arg\min\{z \mid (x,z) \in \mathcal{F}_k\}$. This strategy is possibly the simplest, but by no means the only alternative. For a more comprehensive discussion on nondifferentiable optimization methods, we refer to [28, 18].

To define the analytic center strategy let us introduce the slacks

$$s_j = \begin{cases} z - f(x^j) - \langle \xi^j, x - x^j\rangle, & \forall\, j \in K_1, \\ \gamma^j - \langle a^j, x - x^j\rangle, & \forall\, j \in K_2, \end{cases}$$

and the associated potential

$$\phi_k(x,z) = F_0(x) - \log(\theta^k - z) - \sum_{j\in K} \log s_j,$$

where $F_0(x)$ is the barrier function associated with the set X_0. The analytic center strategy selects

$$(x,z) = \arg\min \phi_k(x,z).$$

For a concise summary of the method and its convergence properties, see [18].

In practice, it is often the case that the function f is additive and the set X is the intersection of many sets. As shown in Section 3.1, this is the case in Lagrangian relaxation, when the Lagrangian dual turns out to be the sum of several functions L_i (each one of them corresponding to a subproblem) and the feasible set X is the intersection of the domains of these functions. In such a case, the oracle returns separate information for each function L_i. We also deal with the similar situation in Benders decomposition with separable subproblem, see Section 3.2.

Let us reformulate Problem (16) as

$$\min\{\sum_{i=1}^{m} f_i(x) \mid x \in X = X_0 \cap \bigcap_{i=1}^{p} X_i\}. \tag{17}$$

For any $x \in X_0$ the oracle associated with this formulation provides the answers

1. $\bar{x}$ is in X and there are support vectors
   $\xi_i \in \partial f_i(\bar{x})$, i.e., $f_i(x) \geq f_i(\bar{x}) + \langle \xi_i, x - \bar{x}\rangle\ \forall x \in R^n$, $i = 1, \ldots, m$.
2. $\bar{x}$ is not in X and there are separation vectors a_i such that
   $\langle a_i, x - \bar{x}\rangle \leq \gamma_i$, $\gamma_i \leq 0$, $\forall i$ such that $\bar{x} \notin X_i$, $i = 1, \ldots, p$, and $\forall x \in X \cap X_0$.

The above oracle can be viewed as a collection of *independent* oracles, one for each function f_i and one for each X_i. This feature has two beneficial consequences. Firstly, the disaggregate formulation much improves the performance of Kelley's cutting plane method [24] and ACCPM [10]. Secondly, and most importantly in our case, disaggregation naturally allows parallel computations.

5 Parallel implementation of cutting plane methods

Decomposition is an obvious candidate for parallel computation [7, 11]. All *independent oracles* can be examined in parallel to reduce the time needed to gather information from the oracle. To make sure that this scheme is computationally advantageous, the ratio of the times spent to solve the consecutive master problems and to solve (sequentially) the subproblems associated with the oracle should be small. The smaller this ratio, the larger the part of work can be done in parallel. Such a situation often appears in decomposition of real-life problems. There are two different cases which may render the oracle expensive:

1. there are relatively few computationally involved independent oracles; or
2. there are many possibly very cheap independent oracles.

Both cases can benefit from parallel computation on a machine with relatively few, but powerful, parallel processors. Moreover, we expect that decomposition working in any of the two above-mentioned conditions will be made *scalable* when implemented on parallel machine. Roughly speaking, scalability means here that the ratio of computing times between the sequential algorithm and a parallel algorithm increases linearly with the number of subproblems (as long as that does not exceed the number of parallel processors available), see [7, 11].

A related notion used extensively in the performance evaluation of parallel algorithms is the *speed-up*. It is the ratio of the solution time when the problem is solved on one processor machine to that of its solution on k-processor parallel machine. Ideally, for *scalable* algorithms, speed-up's keep close to k for k-processor machine.

5.1 The block-angular application problems

The applications presented in Sections 2.1 and 2.2 have relatively small number of subproblems varying from 3 to 16. These subproblems may have large size but, unfortunately, may significantly differ in size among themselves. Consequently, it is not always possible to achieve a good load balancing in the parallel code.

5.2 Survivablity in telecommunications network

The survivability problem (2) is a block-angular linear program with two types of coupling elements: the constraints (2a) couple the independent blocks in (2b), while the variable y couples the $|S|$ blocks of otherwise independent constraints (2a). In [30], the choice was to apply a Lagrangian relaxation to (2a). Note that those constraints are rather numerous. In this paper we apply Benders decomposition by fixing y as primary variable. We obtain a

problem of type (12). In this application, the variable space y has small dimension, but, as we shall see, the subproblems are rather involved.

Since there is no cost associated with the rerouting flows x, the subproblems of type (14) take one of the two values 0 or $+\infty$. Formally, one can write:

$$
\begin{aligned}
L_s(y) = \inf \quad & 0 \\
& \text{s.t.} \sum_{k \in \mathcal{R}_s} x_{ij}^k + x_{ji}^k - y_{ij} \leq K_{ij}^s, \quad \forall (i,j) \in E(s), && \text{(18a)} \\
& (N_s, -N_s) \begin{pmatrix} (x_{ij}^k)_{(i,j)\in E(s)} \\ (x_{ji}^k)_{(i,j)\in E(s)} \end{pmatrix} = b_k, \quad \forall k \in \mathcal{R}_s, && \text{(18b)} \\
& x_{ij}^k \geq 0, \quad \forall k \in \mathcal{R}_s,\ \forall (i,j) \in E(s). && \text{(18c)}
\end{aligned}
$$

This problem is essentially a feasibility problem. It is named the compatible multicommodity flow problem. Although the problem involves relatively few commodities[3], its dimension is not small.

Owing to the block angular structure of (18), one can apply a decomposition scheme to solve it. In our case we introduce an auxiliary problem to handle the first phase

$$
\begin{aligned}
\min & \sum_{(i,j)\in E} z_{ij} \\
\text{s.t.} & \sum_{k \in \mathcal{R}_s} x_{ij}^k + x_{ji}^k - z_{ij} - y_{ij} \leq K_{ij}^s, \quad \forall (i,j) \in E(s), && \text{(19a)} \\
& (N_s, -N_s) \begin{pmatrix} (x_{ij}^k)_{(i,j)\in E(s)} \\ (x_{ji}^k)_{(i,j)\in E(s)} \end{pmatrix} = b_k, \quad \forall k \in \mathcal{R}_s, && \text{(19b)} \\
& x_{ij}^k \geq 0, \quad \forall k \in \mathcal{R}_s,\ \forall (i,j) \in E(s). && \text{(19c)}
\end{aligned}
$$

This problem has an optimal solution that is zero, if $L_s(y) = 0$, or takes a positive finite value if $L_s(y) = +\infty$. To solve this first phase problem, we use a Lagrangian decomposition scheme by dualizing the constraints (19a). If the optimal solution of (19) is positive, then the optimal dual variables associated with the dualized constraints can be used to build a feasibility cut. (For more details on compatible multicommodity flows, see Minoux [32].) The Lagrangian relaxation of the compatible multicommodity flow problem (19) is relatively easy to solve, and Kelley's cutting plane scheme performs well.

In contrast, Problem (12) in the y-space is difficult, with a costly oracle. Indeed, we deal here with a nested (two-level) decomposition scheme. The subproblems of the higher level decomposition, i.e., problems (19), are themselves decomposed. For this reason the analytic center cutting plane method is a method of choice to solve it, as it achieves greater stability and efficiency.

Let us now turn our attention to load balancing among independent processors. The oracle is made of relatively many independent oracles, each of them being an independent compatible multicommodity flow problem. The number of independent oracles is $|S|$, a number that may be rather large (over a hundred elements). The computations can be distributed on the

[3]The number of commodities is equal to the number of different origin/destination communications which are interrupted by the network component failure.

various computing units. Conceivably, the computation load varies from one oracle to the other, i.e., from one element of S to the other. If we use as many computing units as $|S|$, the computing load will be unbalanced. Parallel computation will not be scalable. However, we are not interested in very large parallel machines, but rather in clusters of workstations involving at most a few dozens of them. Then, we face the issue of distributing evenly work among the computing units.

We proceed as follows. At the beginning, the subproblems are cyclically distributed among the nodes. When all subproblems have been solved, the subproblems are sorted according to the time needed to solve them. The time spent on each node is computed, as well as the average time $\bar{t}$ for a node. Tasks are then transferred from the nodes where the time spent is above the average to those where it is below the average as illustrated in Figure 1. For each node, the tasks are processed in increasing order of completion time. This is done as long as the total time of the tasks processed so far is below $\bar{t}$. When the time spent on one node is above the average, the procedure reaches a point where a few tasks are left. These jobs are transferred to the nodes for which the time spent is below average.

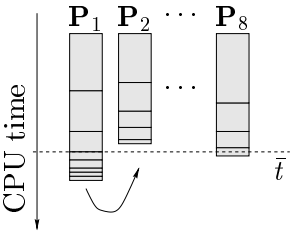


Figure 1: Load balancing

Note that as the decomposition scheme progresses, the relative difficulty of the subproblems may vary. For this reason, we choose to rebalance the load dynamically at each iteration.

6 Numerical results

6.1 Computational resources

To perform the numerical experiments we used two different parallel machines: a cluster of 16 Pentium Pro PC's linked with fast ethernet and a parallel supercomputer IBM SP2. The latter is an expensive machine intended to cover the need of a large institution, whereas the former is a cheap form of parallelism that benefits from recent advances in fast ethernet technology. We briefly expose some salient characteristics of both machines.

Following the idea of the Beowulf project [5, 37], the cluster of 16 Pentium Pro PC's runs under LINUX operating system (RedHat 4.2 distribution). The cluster is linked with fast ethernet that allows a 100MB/s transfer rate. Six machines have 384 MB RAM (one of them is the master) and the remaining 10 machines have 64MB RAM each. We installed MPICH, a public domain version of MPI, to handle communications between processors.

The IBM RS/6000 Scalable POWER parallel System (RS/6000 SP2) at the University of Geneva has 15 nodes. Each of them is a complete RS/6000 workstation with CPU, disk and memory that runs its own operating system AIX 4.1. The processors are standard POWER2 architecture RS/6000 running at 77MHz on one "wide" node and at 66MHz on fourteen "thin" nodes. The nodes are divided into three pools: one dedicated to interactive jobs; another for intensive input/output jobs and a last one for parallel-production work. The last pool was used for the numerical tests; it contains eight processors (all of them are thin nodes). Four of these nodes have 192 MB of RAM memory, the remaining four nodes have 128 MB RAM. The SP2 machine belongs to a message-passing family. Access to the

switch is made possible by call to a library provided by the constructor. Standard libraries such as PVM [14] or MPI [13, 41] are also available. The PVM and MPI libraries are portable to other platforms, a definite advantage in our view.

The implementation of ACCPM corresponds to the one described in [20]. The code combines three programming languages: `C, C++` and `FORTRAN 77`. For the cluster of PC's we compiled them with GNU compilers: `gcc, g++` and `g77`, respectively. On IBM SP2 machine, we compiled them with the AIX compilers: `mmxlc, mmxlC` and `mmxlf`, respectively. In both cases all compilations were done with a default optimization level (`-O` option).

6.2 Programming language for parallel computations

As claimed earlier, our goal was to keep our code general and portable to many parallel computing platforms. Thus, we implemented parallel communication with the Message Passing Interface (MPI) library [22, 13, 41]. An important advantage of this parallel communication model is that it can be implemented on any multiprocessor machine as well as on a cluster of independent machines loosely coupled by a network of the Ethernet family. Public domain implementation of MPI is available for the most popular workstations [22]. Moreover, the MPI standard has been accepted by commercial vendors and is implemented on a majority of parallel computers (in particular, on SP2 of IBM). Consequently, one can develop an implementation using MPI, test it on a network of workstations or a network of PC's, and then port it to a sophisticate parallel machine.

6.3 Solving the block-angular linear programs

6.3.1 Problem characteristics

We solved a set of realistic block-angular linear programs with the number of subproblems varying from 3 to 16.

MARKAL is the multi-country, multi-period energy planning model with the global constraint limiting the emission of CO_2 [4]. ENERGY are the energy planning problems developed at IIASA [34]. There are two variants of these problems: a small and a large one, named ENERGY-S and ENERGY-L, respectively. Few variants of the large model are considered. They differ in the number of subproblems that vary from 2 to 8. MACHINES problems are the decentralized versions of the Markov decision models [12]. We solve one small, two medium, and one large model of that type.

The problem statistics are collected in Table 1. Columns 2, 3 and 4 specify for every problem: the size of the master, i.e., the number of coupling constraints m_0, the number of rows that are active at the optimum, and the number of GUB rows which is the same as the number of subproblems p, respectively. The following columns characterize the subproblems. They report the total number of rows, columns and nonzero elements in all p subproblems. Several problems have well balanced subproblems, the other have not. The last two columns give an insight into this balance: they specify the minimum and the maximum column size of the subproblems.

Problem	Master			Subproblems			Balance	
	rows	active	GUB	rows	cols	nonz	min cols	max cols
MARKAL	7	7	3	17393	36885	204108	8289	15323
ENERGY-S	5	5	4	14364	24080	95202	6020	6020
ENERGY-L2	9	9	2	83160	147790	376072	73895	73895
ENERGY-L4	9	9	4	166320	295580	752144	73895	73895
ENERGY-L8	9	9	8	332640	591160	1504288	73895	73895
MACHINES-S	17	17	16	1600	24624	292948	324	1944
MACHINES-M2	17	17	16	2224	36784	426020	484	2904
MACHINES-M1	17	17	16	2992	51376	615148	676	4056
MACHINES-L	17	17	16	3616	68400	798884	900	5400

Table 1: Statistics of the block-angular decomposable problems.

Problem	Calls	Cuts	Time
MARKAL	26	78	2530
ENERGY-S	12	48	489
ENERGY-L2	30	60	60132
ENERGY-L4	28	112	112391
ENERGY-L8	30	240	231849
MACHINES-S	11	176	172
MACHINES-M2	11	176	282
MACHINES-M1	16	152	565
MACHINES-L	9	144	532

Table 2: Performance of sequential ACCPM on block-angular LPs.

6.3.2 Numerical results

We solved the problems to the relative precision $\epsilon = 10^{-6}$. For all problems considered here, the master program is very small compared to each subproblem. As a result, the time to compute analytic centers is negligible. A key element in the performance is the number of times the master program calls the oracle. We concentrate on that element and neglect internal iterations in the master program.

Our first results pertain to a sequential implementation of the decomposition code on one node of the cluster of PC's. We use the code [21] with the analytic center cutting plane method [20]. Table 2 exhibits the results for problems listed in Table 1. We report: the number of outer iterations, the total number of cuts handled by ACCPM and the overall CPU time in seconds required to reach the optimal solution. The subproblems are solved with HOPDM (Higher Order Primal-Dual Method) code [19] and benefit from an interior point warm start technique.

The problems in the family ENERGY-L are particularly demanding with respect to memory. Each subproblem needs more than 80 MB of storage. Their solution time with the sequential code is also considerable.

Problem	SubPb	1 Proc.	2 Processors		4 Processors		8 Processors	
		time	time	sp-up	time	sp-up	time	sp-up
MARKAL	3	2530	1302	1.94	1187	2.13	nr	nr
ENERGY-S	4	489	256	1.91	133	3.68	nr	nr
ENERGY-L2	2	60132	32370	1.86	nr	nr	nr	nr
ENERGY-L4	4	112391	58215	1.93	30269	3.71	nr	nr
ENERGY-L8	8	231849	121516	1.91	65741	3.52	40098	5.78
MACHINES-S	16	172	113	1.52	60	2.88	34	4.97
MACHINES-M2	16	282	178	1.58	94	3.00	58	4.81
MACHINES-M1	16	565	372	1.52	201	2.81	121	4.65
MACHINES-L	16	532	339	1.57	172	3.09	102	5.21

Table 3: Performance of the parallel decomposition of block-angular LPs.

The parallel decomposition was implemented on the cluster of 16 PC's running under Linux. All problems could all be solved when the appropriate number of processors, 1, 2, 4 or 8, was used. In Table 3 we report the CPU times in seconds required to solve each problem and the speed-up with respect to the solution time required by the sequential code. We did not include the results of running the problems with p subproblems on a parallel machine with more than p processors (we put "nr" to mark that the problems were not run). Clearly, with our parallel decomposition approach one cannot expect speed-up's to exceed p.

The results collected in Table 3 show that our parallel decomposition reaches reasonable speed-up's: from 1.52 to 1.94 when 2 processors are used, about 3 when 4 processors are used, and about 5 on 8-processor parallel machine. These results seem more than satisfactory for the problems we solved. The best speed-up's have been obtained for ENERGY problems which have very well balanced subproblems (cf. Table 1).

The Markov decision problems do not scale so well, due to the discrepancy in the size and the difficulty of the subproblems. Yet the speed-up's of about 5 on an 8-processor machine seem fairly satisfactory. Although these problems are not very large for today's LP codes, they are very difficult due to the presence of low probability transitions rates (very small coefficients in the matrix) with important global consequences. An attempt of solving a stright undecomposed formulation breaks down commercial LP solvers. The use of decomposition spreads numerical difficulties across subproblems and the overall problem can easily be solved.

6.4 Survivability of telecommunications networks

6.4.1 Problem characteristics

The tests have been performed with problems of different nature: eight of them were generated with a random-problem generator (see [40]); other are real-life disguised problems (T1, T2, T3). In Table 4 we report their characteristics. For each network, we first give the number of nodes (NODES), then the number of arcs (ARCS) and the number of rout-

Problem	Basic data			Failure data	
	Nodes	Arcs	Routings	Failures	Cond. Demand
P22	10	22	50	30	124
P29	15	29	400	42	1488
P53	15	53	400	67	1374
P56	35	56	1000	85	5832
P109	35	109	1000	109	4674
P145	25	145	1000	170	3784
P227	40	227	1000	267	3834
P239	45	239	1200	283	5164
PB1	30	57	150	81	1110
PB2	37	71	300	102	3266
PB3	40	77	200	109	1942
PB4	45	87	300	123	3658
PB5	65	127	400	179	7234
T1	16	23	84	38	300
T2	31	68	961	99	2877
T3	68	78	961	99	2887

Table 4: Characteristics of the survivability problems

ings (Routings)[4]. The latter characteristics concern the network in the normal operational state. We next indicate features that are related to the failure states: the number of failures (Failures)[5] and the overall number of conditional demands resulting from the failures (Cond. Demand). Let us observe that each problem involves a large number of routings.

To give an idea of the problem dimension, it is possible to reformulate the survivability problem as a large LP [31]. The size of each problem for a compact LP formulation involving mass-balance equations can be found in Table 5. The size is computed according to the formulas: the number of rows is

$$\sum_{s\in S}|E(s)|+\sum_{s\in S}\sum_{k\in\mathcal{R}_s}|V(s)| \tag{20}$$

while the number of columns is

$$\sum_{s\in S}\sum_{k\in\mathcal{R}_s}|E(s)|+|E|. \tag{21}$$

Let us observe that P239 has the largest dimension: it requires more than 1.5 million variables and about 2 hundred thousand constraints. Such a large dimension results from an important number of failures and conditional demands.

[4]Recall that the routings are fixed data in the problem.

[5]The theoretical number of failures is equal to $|S|+|N|$. However, some arcs may have no regular flow on them and their failure has no consequence. We consider in Failures only failures that force a rerouting of messages.

Problem	Variables	Constraints
P22	3676	1332
P29	52893	14972
P53	92301	16061
P56	375816	122640
P145	689041	80680
P227	1092711	147982
P239	1514871	200072
PB1	70617	22092
PB2	249691	70941
PB3	162869	48692
PB4	340475	96537
PB5	962011	264107
T1	8471	3578
T2	257214	64045
T3	64494	22158

Table 5: Problem characteristics

6.4.2 Numerical results

Just as in the case of the block-angular linear programs, the time spent in computing the analytic center in the master program is relatively small, if not negligible, with respect to the time spent in the subproblems. The main difference with the previous set of experiments concerns the number of subproblems. They are in far greater number; they are also smaller, but more involved as they are themselves solved via a decomposition scheme. Again, the key efficiency factor in the overall performance is the number of calls to the oracle.

The experiments were conducted on 8 processors[6] of the IBM SP2. As we mentioned previously, the subproblem needs solving multicommodity network flow problems for which Kelley's cutting plane method performs well. We therefore needed an efficient simplex to solve the subproblems. Unfortunatly, when the numerical experiments were made, no commercial simplex was running under the operating system Linux (this is no longer true). Consequently, we did not solve these problems on our cluster of PC's.

For each problem, in Table 6 we provide the number of calls to the oracle (each call corresponds to the solving of several compatible multiflow problems), the CPU time in seconds for the oracle (t_o), the master (t_m) and the whole algorithm (t_t), as well as the time speed-up (one processor compared with 8 processors). Note that the total time t_t includes input/output operations that add up to the oracle and the master computing times.

Let us first observe that the number of iterations grows regularly with the number of network arcs. Note that the time spent in the master of ACCPM does not exceed 2%; most of the time is spent in the oracle, as we previously remarked. All problems, even the largest, could be solved in a reasonable time.

Problems *P227* and *P239* were not solved sequentially, due to time constraints. Conse-

[6]This is the maximum number of processors available for parallel experiments on this machine.

quently, speed-up's were not computed for them; in these cases we wrote "nc" in the corresponding column. For all other problems, we computed the speed-up's. The figures run from 3.8 for the smaller problem to 7 for the larger ones. Note that problems *P22* and *T1* are much smaller than the other instances. Yet the speed-up's are 3.8 and 5.3 respectively. We feel that similar ratios could be achieved on larger problems with more processors, giving hope that much larger problems could be solved in practice.

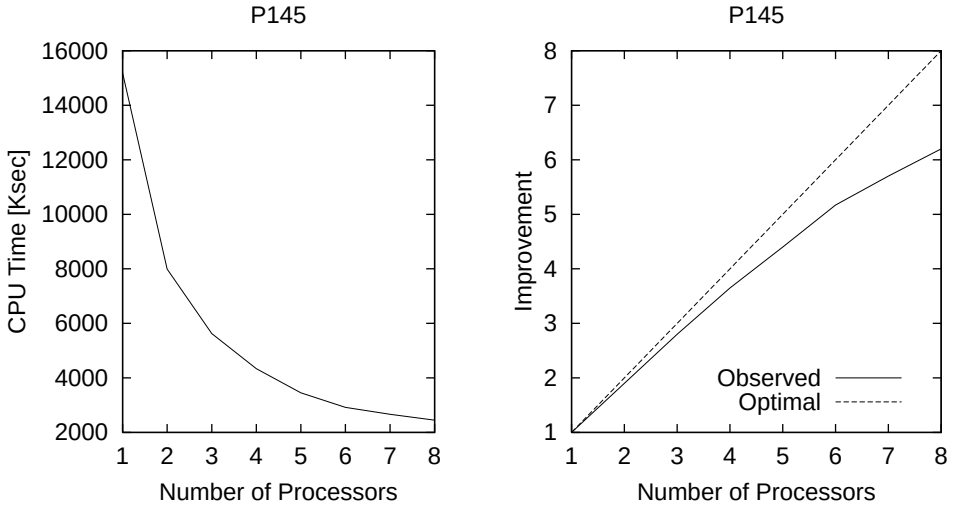
Ideally the running time of parallel application should decrease linearly when the number of processors increases. In Figure 2, we attempt to reveal the influence of the number of processors on the overall running time for two problems.

Problem	Calls	Cuts	CPU (secs)			Speed-up
			t_o	t_m	t_t	
P22	17	45	4.86	0.11	5.0	4.01
P29	32	56	49.8	0.15	49.9	6.35
P53	26	41	137.7	0.35	138	4.42
P56	27	118	137.3	0.6	138	5.36
P145	45	161	2432	17	2450	6.2
P227	51	203	5128	39	5275	nc
P239	58	272	9121	85	9208	nc
PB1	25	93	25.6	0.49	26.2	6.5
PB2	23	135	123.6	0.9	124.6	6.24
PB3	24	128	49.4	1.1	50.6	6.01
PB4	25	180	137.5	1.7	139.3	6.01
PB5	26	282	1215.9	8.78	1225	5.85
T1	25	100	6.35	0.16	6.58	5.31
T2	34	91	590.3	1.39	591.8	6.46
T3	29	96	28.2	0.59	28.92	4.47

Table 6: Results

Figure 2 concerns the problem P145. In Figure 2 (a) we display the overall running time according to the number of processors that were used in parallel. We can observe a large decrease in the running time: from more than 15190 to less than 2450 seconds. Figure 2 (b) translates this into speed-up: the dashed line represents perfect scalability, that is the linear decrease of time compared to the number of processors, whereas the pure line represents the observed speed-up. The pure line stays close to the bold one. It stems from the figure, that the scalability is rather good on this example. In fact, for 8 processors the speed-up is 6.2.

We have not been able to measure the speed-up for larger problems than P145; this would have required freezing the parallel machine for a considerable time. We were not allowed to do so. However, we can conjecture that we should expect similar—if not better—speed-up's for the larger problems. A large number of failures allows more freedom in the load balancing: it is intuitively quite clear that it should be easier to obtain equal times when dividing a larger number of tasks.



(a) **(b)**

Figure 2: CPU time and speed-up as a function of the number of nodes

7 Conclusions

We have shown in this paper computational advantages resulting from the parallelisation of the oracle in a decomposition using central prices. We have applied the code to solve two different types of problems: with small number of computationally expensive independent subproblems and with a large number of relatively cheap subproblems. In both cases remarkable speed-up's have been obtained. In the latter, our parallel code reaches scalability.

Two realistic applications, in energy planning and in telecommunications, as well as a challenging Markov decision problem have been shown to take advantage of our parallel implementation of the analytic center cutting plane method.

An interesting alternative is to modify the original ACCPM method to have the master program activated any time it is idle and a subproblem has been solved on another processor. Reciprocally, solution of a subproblem starts as soon as a processor becomes available, similarly to the idea of "asynchronous" parallel decomposition of [39]. The dual prices to be used are the most recent output of the master program. In that way one may expect to increase the efficiency of decomposition and possibly improve the speed-ups of the algorithm.

Acknowledgment

We are grateful to Jemery Day, Alain Haurie and Francesco Moresino for providing us with challenging Markov decision problems arising in manufacturing.

References

[1] M. Abbad and J. Filar, *Algorithms for singularly perturbed limiting average Markov control problems*, IEEE Transactions on Automatic Control, 9 (1992), pp. 153–168.

[2] D. Alevras, M. Grötschel, and R. Wessäli, *Capacity and survivability models for telecommunication networks*, tech. report, Konrad-Zuse-Zentrum für Informationstechnik, Takustrasse 7, D-14195 Berlin, Germany, June 1997.

[3] O. Bahn, O. du Merle, J.-L. Goffin, and J.-P. Vial, *A cutting plane method from analytic centers for stochastic programming*, Mathematical Programming, 69 (1995), pp. 45–73.

[4] O. Bahn, A. Haurie, S. Kypreos, and J.-P. Vial, *A multinational MARKAL model to study joint implementation of carbon dioxide emission reduction measures*, in Joint Implementation of Climate Change Commitments: Opportunities and Apprehensions, P. Ghosh and J. Puri, eds., Tata Energy Research Institute, New Delhi, 1994, pp. 43–50.

[5] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer, *Beowulf: A parallel workstation for scientific computation.* Proceedings of the International Conference on Parallel Processing (ICPP), pp. 11–14, 1995.

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

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel an Distributed Computations*, Prentice-Hall, Englewood Cliffs, 1989.

[8] E. Cheney and A. Goldstein, *Newton's method for convex programming and Tchebycheff approximation*, Numerische Mathematik, 1 (1959), pp. 253–268.

[9] G. B. Dantzig and P. Wolfe, *The decomposition algorithm for linear programming*, Econometrica, 29 (1961), pp. 767–778.

[10] O. du Merle, J.-L. Goffin, and J.-P. Vial, *On the comparative behavior of Kelley's cutting plane method and the analytic center cutting plane method*, tech. report, Logilab, University of Geneva, 102 Bd Carl-Vogt, CH-1211, March 1996. To appear in *Computational Optimization and Applications.*

[11] J. Eckstein, *Large-scale parallel computing, optimization, and operations research: A survey*, ORSA CSTS Newsletter, 14 (Fall 1993), pp. 1–28.

[12] J. Filar and A. Haurie, *Optimal ergodic control of singularly perturbed hybrid stochastic systems*, Lectures in Applied Mathematics, 33 (1997), pp. 101–126.

[13] M. P. I. Forum, *MPI: A message-passing interface standard*, International Journal of Supercomputer Applications, 8 (1994).

[14] A. Geist, A. Begelin, J. Dongarra, W. Jiang, and R. Mancheck, *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.

[15] G. Ghellinck and J.-P. Vial, *A polynomial Newton method for linear programming*, Algorithmica, 1 (1986), pp. 425–453.

[16] J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-P. Vial, *Solving nonlinear multicommodity flow problems by the analytic center cutting plane method*, Mathematical Programming, 76 (1997), pp. 131–154.

[17] J.-L. Goffin, A. Haurie, and J.-P. Vial, *Decomposition and nondifferentiable optimization with the projective algorithm*, Management Science, 38 (1992), pp. 284–302.

[18] J.-L. Goffin and J.-P. Vial, *Interior point methods for nondifferentiable optimization*, in Operations Research Proceedings 1997, P. Kischka, H.-W. Lorenz, U. Derigs, W. Domaschke, P. Kleinschmidt, and R. Möhring, eds., Springer Verlag, Berlin, Germany, 1998, pp. 35–49.

[19] J. Gondzio, *HOPDM (version 2.12) – a fast LP solver based on a primal-dual interior point method*, European Journal of Operational Research, 85 (1995), pp. 221–225.

[20] J. Gondzio, O. du Merle, R. Sarkissian, and J.-P. Vial, *ACCPM - a library for convex optimization based on an analytic center cutting plane method*, European Journal of Operational Research, 94 (1996), pp. 206–211.

[21] J. Gondzio and J.-P. Vial, *Warm start and ε-subgradients in cutting plane scheme for block-angular linear programs*, Computational Optimization and Applications, 14 (1999), pp. 17–36.

[22] W. Gropp and E. Lusk, *User's guide to MPICH, a portable implementation of MPI*, Tech. Report ANL/MCS-TM-ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, USA, 1996.

[23] P. Huard, *Resolution of mathematical programming with nonlinear constraints by the methods of centers*, in Nonlinear Programming, J. Abadie, ed., North-Holland, Amsterdam, 1967, pp. 209–222.

[24] K. L. Jones, I. J. Lustig, J. M. Farvolden, and W. B. Powell, *Multicommodity network flows: the impact of formulation on decomposition*, Mathematical Programming, 62 (1993), pp. 95–117.

[25] N. K. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[26] J. E. Kelley, *The cutting plane method for solving convex programs*, Journal of the SIAM, 8 (1960), pp. 703–712.

[27] K. C. Kiwiel, *A survey of bundle methods for nondifferentiable optimization*, in Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers, 1989, pp. 262–282.

[28] C. Lemaréchal, *Nondifferentiable optimization*, in Handbooks in Operations Research and Management Science, G. Nemhauser, A. R. Kan, and M. Todd, eds., vol. 1 of Optimization, North-Holland, 1989, pp. 529–572.

[29] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, *New variants of bundle methods*, Mathematical Programming, 69 (1995), pp. 111–147.

[30] A. Lisser, R. Sarkissian, and J.-P. Vial, *Survivability in telecommunication networks*, Tech. Report 1995.3, Logilab, University of Geneva, 102 Bd Carl-Vogt, CH-1211, March 1995.

[31] M. Minoux, *Optimum synthesis of a network with non-simultaneous multicommodity flow requirements*, in Studies on Graphs and Discrete Programming, P. Hansen, ed., Noth-Holland, 1981, pp. 269–277.

[32] ——, *Mathematical Programming: Theory and Algorithms*, Wiley, New York, 1986.

[33] J. E. Mitchell and M. J. Todd, *Solving combinatorial optimization problems using Karmarkar's algorithm*, Mathematical Programming, 56 (1992), pp. 245–284.

[34] N. Nakicenovic, A. Gruebler, A. Inaba, S. Messner, S. Nilsson, Y. Nishimura, H.-H. Rogner, A. Schaefer, L. Schrattenholzer, M. Strubegger, J. Swisher, D. Victor, and D. Wilson, *Long-term strategies for mitigating global warming*, Energy—The International Journal, 18 (1993), pp. 409–601.

[35] A. Nemirovsky and D. Yudin, *Informational Complexity and Efficient Methods for Solution of Convex Extremal Problems*, J. Wiley & Sons, New York, 1983.

[36] J. Renegar, *A polynomial-time algorithm, based on Newton's method, for linear programming*, Mathematical Programming, 40 (1988), pp. 59–93.

[37] D. Ridge, D. Becker, P. Merkey, and T. Sterling, *Beowulf: Harnessing the power of parallelism in a pile-of-PCs.* Proceedings, IEEE Aerospace, 1997.

[38] A. Ruszczyński, *A regularized decomposition method for minimizing a sum of polyhedral functions*, Mathematical Programming, 33 (1985), pp. 309–333.

[39] ——, *Parallel decomposition of multistage stochastic programs*, Mathematical Programming, 58 (1993), pp. 201–228.

[40] R. Sarkissian, *Telecommunications Networks: Routing and Survivability Optimization Using a Central Cutting Plane Method*, PhD thesis, École Polytechnique Fédérale de Lausanne, CH-1205 Ecublens, November 1997.

[41] M. Snir, S. W. Otto, S. Huss-Ledernan, D. W. Walker, and J. Dongarra, *MPI: the Complete Reference*, MIT Press, Cambridge, 1996.

[42] G. Sonnevend, *New algorithms in convex programming based on a notion of "centre" (for systems of analytic inequalities) and on rational extrapolation*, in Trends in Mathematical Optimization: Proceedings of the 4th French–German Conference on Optimization in Irsee, Germany, April 1986, K. H. Hoffmann, J. B. Hiriart-Urruty, C. Lemaréchal, and J. Zowe, eds., vol. 84 of International Series of Numerical Mathematics, Birkhäuser Verlag, Basel, Switzerland, 1988, pp. 311–327.

[43] Y. Ye, *A potential reduction algorithm allowing column generation*, SIAM Journal on Optimization, 2 (1992), pp. 7–20.