High Performance Computing for Asset Liability Management

Jacek Gondzio and Roy Kouwenberg

21 May 1999, revised 29 March 2000

MS-99-004

This paper has been accepted for publication in *Operations Research.*

High Performance Computing for Asset Liability Management[*][†]

Jacek Gondzio[‡]

Department of Mathematics & Statistics,
The University of Edinburgh,
Mayfield Road, Edinburgh EH9 3JZ,
United Kingdom.

and Roy Kouwenberg[§]

Econometric Institute,
Erasmus University Rotterdam,
P.O.Box 1738, 3000 DR Rotterdam,
The Netherlands.

21 May 1999, revised 29 March 2000

[*]to appear in *Operations Research.*

[†]Research partially supported through contract "HPC-Finance" (no. 951139) of the European Commission. Partial support also provided by the "HPC-Finance" partner institutions: Universities of Bergamo (IT), Cambridge (UK), Calabria (IT), Charles (CZ), Cyprus (CY), Erasmus (ND), Technion (IL) and the "Centro per il Calcolo Parallelo e i Supercalcolatori" (IT).

[‡]Email: `gondzio@maths.ed.ac.uk`, URL: `http://maths.ed.ac.uk/~gondzio/`

[§]Email: `kouwenberg@few.eur.nl`, URL: `http://www2.tinbinst.nl/~kouwenbe/`

High Performance Computing for Asset Liability Management

Abstract

Financial institutions require sophisticated tools for risk management. For company-wide risk management both sides of the balance sheet should be considered, resulting in an integrated asset liability management approach. Stochastic programming models suit these needs well and have already been applied in the field of asset liability management to improve financial operations and risk management. The dynamic aspect of the financial planning problems inevitably leads to multiple decision stages (trading dates) in the stochastic program and results in an explosion of dimensionality. In this paper we show that dedicated model generation, specialized solution techniques based on decomposition and high performance computing are the essential elements to tackle these large scale financial planning problems. It turns out that memory management is a major bottleneck when solving very large problems, given an efficient solution approach and a parallel computing facility. We report on the solution of an asset liability management model for an actual Dutch pension fund with 4,826,809 scenarios, 12,469,250 constraints and 24,938,502 variables, which is the largest stochastic linear program ever solved. A closer look at the optimal decisions reveals that the initial asset mix is more stable for larger models, demonstrating the potential benefits of the high-performance computing approach for ALM.

1 Introduction

We are concerned in this paper with the real-life financial planning problem of a Dutch pension fund. The majority of Dutch pension funds use operations research techniques to plan their investment decisions, funding decisions and risk management. Most Dutch pension funds rely on the commercial hybrid simulation and optimization system described by Boender (1997). The system of Boender only considers static investment strategies or dynamic strategies based on a given decision rule. One way of modeling the dynamic features of the underlying decision problem is to apply the stochastic programming approach. This approach seems to be widely accepted for solving dynamic asset liability management problems (Bradley and Crane 1972, Kusy and Ziemba 1986, Carino et al. 1994, Mulvey and Vladimirou 1992, Zenios 1995, Consigli and Dempster 1998).

Unfortunately, there are two factors that cause an explosion of dimensionality in the stochastic programming approach applied to asset liability management. The size of the problem grows exponentially with the number of portfolio rebalancing dates. Moreover, to be sufficiently precise, a discrete approximation of the conditional distribution of the asset returns requires a considerable number of realizations. The dimension of the resulting optimization problem grows as p^T, where p is the number of realizations for the one-period asset returns and T is the number of portfolio rebalancing dates. This creates a challenge of how to generate the models, how to solve them and which computing platform to select. In this paper we show how to integrate all the necessary elements and we reach the limit of what can be done with the technology available today. We demonstrate, in particular, that the *generation* of huge models is a key issue for success.

The generation of the deterministic equivalent form of a realistic stochastic program is a difficult task that requires a huge amount of computer resources (both memory and time). It seems to be a bottleneck that prevents the solution of very large problems in practice. To overcome this drawback, both the generation and the optimization strategy have to work with smaller manageable parts of the problem. Decomposition is a suitable optimization approach for this purpose. Decomposition methods split the problem up into a coordinator task (the *master problem*) and a set of loosely coupled *subproblems*; the latter can be solved independently. The advantage of this solution strategy is that subproblems can be generated when needed and forgotten afterwards. When problems are huge and the available memory is a bottleneck, this feature of decomposition becomes critical. However, for the whole process to be efficient, the generation of every subproblem has to be done very quickly, otherwise the overhead of repeated generation would slow down the solution method. We generate the subproblems with the modeling tool LEQGEN (Kouwenberg 1999), which is a fast and efficient C-code for general stochastic linear and quadratic programs.

When very large stochastic programs are solved, the master and the subproblems themselves are linear programs of considerable size. Their repeated solution represents the main computational effort in the decomposition algorithm. It is crucial to reduce the number of subproblem solves to a minimum. We meet this requirement by applying the *primal-dual column generation method* (Gondzio and Sarkissian 1996). In the context of stochastic programming, our decomposition approach simply reduces to the Benders decomposition (Benders 1962, Van Slyke and

Wets 1969), which is clearly an attractive candidate for parallelization (Ariyawansa and Hudson 1991). Decomposition methods are particularly well suited to a distributed computing environment (Bertsekas and Tsitsiklis 1988): independent subproblems can be solved simultaneously by different processors. Finally, to achieve efficient subproblem solves, we use the primal-dual interior point code HOPDM (Gondzio 1995).

Summing up, we gathered three crucial factors: an efficient model generation tool, a suitable optimization method and high-performance computing power. This integration allowed us to solve a practical asset liability management problem with 4,826,809 scenarios. The corresponding stochastic linear program has 12,469,250 constraints and 24,938,502 variables. To the best of our knowledge, this problem is the largest linear program ever solved. We solved it on a 16 processor PARSYTEC CC16 parallel machine at the University of Cyprus. The model generator, LEQGEN is written in ANSI C, the decomposition solver based on HOPDM is written in ANSI C and FORTRAN 77 and the parallel communication uses PVM (Geist et al., 1994). Although we have done our computations on a true parallel machine, our code is easily portable to any distributed computing environment, including a cluster of PC's running under Linux operating system.

One might question whether solving very large ALM problems is actually necessary to improve the operations of financial institutions? Theoretically one would expect that large scale models are required in order to include multiple decision dates, while simultaneously approximating the underlying return distributions with sufficient realizations and precision. In the numerical section of the paper we address this issue by studying the optimal initial asset mix and objective value of the ALM model. We find that the initial optimal solution differs drastically between the smaller models, while it stabilizes for the larger models. Our results therefore demonstrate the potential benefits of solving large ALM models. Although a full investigation is beyond the scope of this paper, we believe that convergence and scenario generation are highly relevant issues for successful implementations of stochastic programming models for ALM. The high-performance computing approach presented here could be an important tool to facilitate future research in this area.

The paper is organized as follows. In Section 2 we discuss the asset liability management model of a Dutch pension fund and the application of the stochastic programming approach. In Section 3 we briefly recall the Benders decomposition (Benders 1962) and its variant for two-stage stochastic programs called the L-shaped method (Van Slyke and Wets 1969). In Section 4 we explain why in solving very large stochastic programming problems a specialized generation tool is essential, and we discuss key features of the model generator LEQGEN. In Section 5 we discuss numerical results and finally, in Section 6, we give our conclusions.

2 Asset-Liability Management

Risk management has become essential for financial service providers like banks, pension funds and insurance companies. Considering both assets and liabilities is crucial for proper risk management, leading to an integrated asset-liability management (ALM) approach (see Mulvey and Ziemba 1998). As an example of an ALM risk management problem, we will consider the financial planning problem of an actual Dutch pension fund. The goal of the pension fund is to provide its participants with a benefit payment equal to 70% of their final salary. In return, the participants and their employers pay contributions to the pension fund in each year prior to retirement. The pension fund has to decide how to invest these contribution payments, in order to fulfill its long term obligations and to meet short term solvency requirements.

An ALM model for the pension fund should recommend an investment policy and a contribution policy, given the economic expectations and the preferences of the pension fund. In the Netherlands most pension funds use a decision support system for ALM based on simulation (Boender 1997). Note that the planning horizon of most pension funds stretches out for decades, as a result of the long term commitment to pay benefits to the retired workers. However, the pension fund should also reckon with short term solvency requirements imposed by regulatory authorities. The trade-off between long term gains and short term losses should be made carefully, while anticipating future adjustments of the policy. This setting therefore seems to be suited for a multi-stage stochastic programming approach with dynamic investment strategies. Stochastic programming models have been applied previously to ALM problems by Bradley and Crane (1972), Kusy and Ziemba (1986), Carino et al. (1994), Mulvey and Vladimirou (1992), Zenios (1995) and Consigli and Dempster (1998). We will now briefly introduce stochastic programming models, with a strong focus on financial planning applications.

2.1 The Stochastic Programming Approach

Stochastic programming models allow for progressive revelation of information through time and multiple decision stages, where each decision is adapted to the available information (Birge and Louveaux 1997, Kall and Wallace 1994). The stochastic programming framework is suited for a wide range of planning problems including telecommunication, power management and financial planning. Stochastic programming models have been applied successfully, especially in the field of finance. For a financial application the decision stages of a stochastic program typically represent trading dates, at which the decision maker could rebalance his portfolio of assets. Modeling of uncertainty is required because the returns on assets such as real estate and stocks are random variables. Finally, the investor will update his optimal portfolio in the future as information about the asset returns becomes available. Later on we will introduce a stochastic programming model for the financial planning problem of a pension fund, which includes these elements of decision making under uncertainty. First we will present the stochastic programming approach in its general formulation.

We consider the problem of a decision maker who has to decide right now about an optimal policy x with costs $c'x$, while facing uncertainty about the state of the world after one period. In a financial planning model x could represent the initial number of assets bought at price c. Uncertainty is described by the random variable $\tilde{\xi}$, which is revealed at the planning horizon after one period. Given the realized scenario ξ at the planning horizon, the decision maker chooses a recourse action $y(\xi)$ which leads to the additional costs $q(\xi)'y(\xi)$. In a financial context $\tilde{\xi}$ could represent the state of the economy after one period while $y(\xi)$ could be the number of assets sold at price $-q(\xi)$. The goal of the decision maker is to minimize the sum of the first stage costs $c'x$ and the expected second stage costs $E_{\tilde{\xi}}\left[q(\tilde{\xi})'y(\tilde{\xi})\right]$. Both the first stage decisions x

and the second stage decisions $y(\tilde{\xi})$ are restricted by a set of linear equations, describing the set of feasible solutions. A financial planning problem would for example require linear budget equations. The complete two-stage stochastic linear program of the decision maker is:

$$\begin{array}{lll}(1) & \min & c'x + E_{\tilde{\xi}}\left[q(\tilde{\xi})'y(\tilde{\xi})\right] \\ & s.t. & Ax = b,\ x \geq 0, \\ & & W(\xi)y(\xi) = h(\xi) - T(\xi)x,\ y(\xi) \geq 0,\ \forall \xi \in \Xi,\end{array}$$

where $\tilde{\xi}$ is a random vector and Ξ is the set of all possible realized ξ, $x \in \mathcal{R}^{n_1}$ is the vector of first stage decision variables, $A \in \mathcal{R}^{m_1 \times n_1}$ is the first stage constraint matrix, $b \in \mathcal{R}^{m_1}$ is the first stage vector of right hand side coefficients and $c \in \mathcal{R}^{n_1}$ is the vector of first stage objective coefficients. $y(\xi) \in \mathcal{R}^{n_2}$ is a random vector of second stage decision variables, $T(\xi) \in \mathcal{R}^{m_2 \times n_1}$ is the random matrix that links the first stage and the second stage decisions, $W(\xi) \in \mathcal{R}^{m_2 \times n_2}$ is the random second stage recourse matrix, $h(\xi) \in \mathcal{R}^{m_2}$ is the random second stage right hand side and $q(\xi) \in \mathcal{R}^{n_2}$ is the random vector of second stage objective coefficients.

Alternatively, the two-stage stochastic linear program (1) could be formulated as follows:

$$\begin{array}{lll}(2) & \min & c'x + E_{\tilde{\xi}}\left[Q(x,\tilde{\xi})\right] \\ & s.t. & Ax = b,\ x \geq 0, \\ (3) & & Q(x,\xi) = \min\limits_{y(\xi)} \{\ q(\xi)'y(\xi) \mid W(\xi)y(\xi) = h(\xi) - T(\xi)x,\ y(\xi) \geq 0\ \},\ \forall \xi \in \Xi.\end{array}$$

The objective function together with the set of first stage constraints (2) is called the *first stage problem*. The decision maker has to decide now about his actions x, while facing uncertainty about the realization ξ of the random variable $\tilde{\xi}$. After one period the decision maker will observe the realized value of ξ and take a recourse action to correct the difference between $h(\xi)$ and $T(\xi)x$, while minimizing the costs $q(\xi)'y(\xi)$. This problem (3) is called the *second stage problem* or the *recourse problem*. Overall, the goal of the two-stage stochastic linear program is to find a first stage solution which guarantees the second stage feasibility and minimizes the sum of the initial costs and the expected second stage recourse costs.

We now assume that the sample space Ξ of the random variable $\tilde{\xi}$ consists of a finite number of realizations $\xi_n = (W_n, q_n, h_n, T_n)$, for $n = 1, 2, .., N$. The sample space Ξ has an associated discrete probability measure $p_n \geq 0$ for $n = 1, 2, .., N$, where the probabilities sum up to 1. In this case the two-stage problem (2-3) becomes:

$$\begin{array}{lll}(4) & \min & c'x + \sum\limits_{n=1}^{N} p_n Q(x,\xi_n) \\ & s.t. & Ax = b,\ x \geq 0, \\ (5) & & Q(x,\xi_n) = \min\limits_{y_n} \{\ q_n'y_n \mid W_n y_n = h_n - T_n x,\ y_n \geq 0\ \},\ n = 1, 2, .., N.\end{array}$$

Given the discrete sample space Ξ the two-stage stochastic linear program can also be written as a linear program (6), which is called the deterministic equivalent. The constraint matrix of the deterministic equivalent has a clear block structure (more precisely, a dual block-angular structure) and is well suited for decomposition methods.

$$
\begin{array}{llllllll}
(6) & \min & c'x & + p_1 q_1' y_1 & + p_2 q_2' y_2 & + \cdots & + p_N q_N' y_N & \\
& s.t. & Ax & & & & & = b \\
& & T_1 x & + W_1 y_1 & + & & & = h_1 \\
& & T_2 x & + & + W_2 y_2 & + & & = h_2 \\
& & \vdots & & & \ddots & & \vdots \\
& & T_N x & + & & & + W_N y_N & = h_N \\
& & x \geq 0, & y_1 \geq 0, & y_2 \geq 0, & & y_N \geq 0. &
\end{array}
$$

In the general multi-stage case, the decision maker has to choose a sequence of decisions $x_0,\ x_1,\ \cdots,\ x_T$, up to the planning horizon T. After one period he observes the realization ξ_1 of the random variable $\tilde{\xi}_1$ and adapts the policy according to the recourse action $x_1(\xi_1)$, while minimizing the immediate costs and the future expected recourse costs. This process of observing new information and adapting the policy is continued up to planning horizon T. The multi-stage stochastic linear program is defined as:

$$
\begin{array}{lll}
(7) & \min & c'x_0 + E_{\tilde{\xi}_1}\left[Q^1(x_0, \tilde{\xi}_1)\right] \\
& s.t. & Ax_0 = b,\ x_0 \geq 0, \\
(8) & & Q^t(x_{t-1}, \xi_t) = \min\limits_{x_t} \Big\{\ q(\xi_t)'x_t\ + E_{\tilde{\xi}_{t+1}}\left[Q^{t+1}(x_t, \tilde{\xi}_{t+1}) \mid \xi_t\right] \mid \\
& & \qquad\qquad W(\xi_t)x_t = h(\xi_t) - T(\xi_t)x_{t-1},\ x_t \geq 0 \quad \Big\}, \\
& & \qquad \forall \xi_t \in \Xi_t,\ t = 1, 2, .., T, \\
& & Q^{T+1}(x_T, \xi_{T+1}) = 0.
\end{array}
$$

Equivalent to the two-stage case, the multi-stage stochastic linear problem (7-8) can be reduced to a deterministic equivalent linear program if the underlying sample space is discrete. In the discrete case the realizations $\xi_{tn} = (W_{tn}, q_{tn}, h_{tn}, T_{tn})$ for $n = 1, 2, .., N_t$ of the random variable $\tilde{\xi}$ can be described by an event tree (Figure 1). Each node n at time t in the event tree is denoted by (t, n) and represents a realization ξ_{tn} with associated probability of p_{tn}. We use $\hat{n}$ to denote the predecessor at time $t-1$ of node n at time t.

2.2 ALM Model for Pension Funds

We will now introduce a multi-stage stochastic programming model for asset liability management of pension funds. The key decision variables are X^h_{itn}, X^b_{itn} and X^s_{itn}, the number of units of assets of class i invested, bought and sold, respectively, at time t. To model the contribution payments to the pension fund by the plan sponsors, we use the contribution rate relative to the

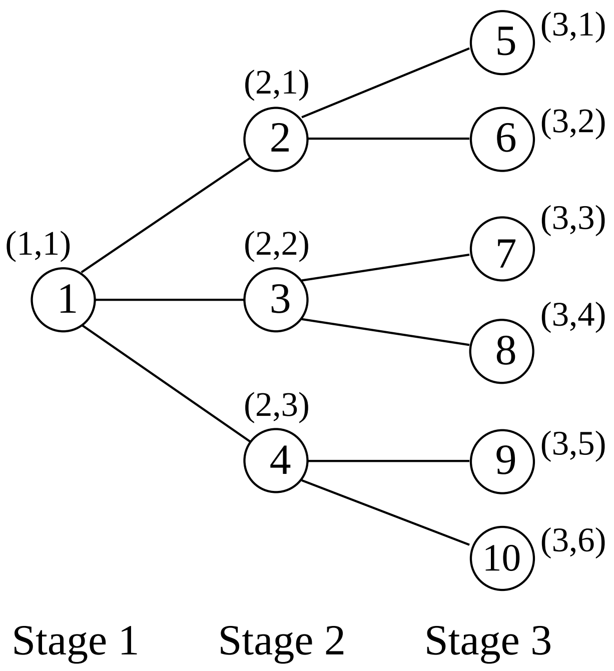

Figure 1: Event Tree

wages cr_{tn} as a decision variable. The wealth of the pension fund A_{tn} is sometimes measured relative to the value of the liabilities L_{tn}, where $F_{tn} = \frac{A_{tn}}{L_{tn}}$ is called the *funding ratio*. The pension fund avoids deficits as long as the funding ratio stays above 1, i.e. the value of the assets is higher than the value of the liabilities. A complete dictionary of variables, coefficients and parameters of the ALM model is provided in Appendix A. For a more detailed description of this model we refer to Kouwenberg (1998).

Objective of the ALM model:

The objective of the model is to minimize the sum of the average contribution rates, while taking into account the risk aversion of the pension fund and the state of the fund at the planning horizon. Risk aversion is modeled with a penalty on deficits Z_{tn} in the objective function (first downside moment of the funding ratio). To measure the final wealth of the pension fund, the funding ratio is restored to the target level F^{end} at the planning horizon by setting the contribution rate at cr_s^{end}. Without this final contribution, end effects could occur and moreover it would be difficult to compare solutions.

$$(9) \quad \min \sum_{t=0}^{T-1} \left(\sum_{n=1}^{N_t} p_{tn} cr_{tn} \right) + \lambda \sum_{t=1}^{T} \left(\sum_{n=1}^{N_t} p_{tn} \frac{Z_{tn}}{L_{tn}} \right) + \sum_{n=1}^{N_T} p_{Tn} cr_n^{end}.$$

Cash balance equation:

The cash balance equation specifies that the cash inflow (including borrowing, assets sold and contributions) should be equal to the cash outflow (including lending, assets bought and benefit payments). Transaction costs on buying and selling of assets are incorporated.

$$(10)\quad m^b_{tn} - m^l_{tn} + (1 + r^l_{tn})m^l_{t-1,\hat{n}} - (1 + r^b_{tn})m^b_{t-1,\hat{n}} + cr_{tn}S_{tn} - B_{tn} + \sum_{i=1}^{I}(1-\gamma^s_i)X^s_{itn} - \sum_{i=1}^{I}(1+\gamma^b_i)X^b_{itn} = 0,\ n = 1..N_t,\ t = 1..T-1,$$

$$(11)\quad m^b_{01} - m^l_{01} + m^{ini} + cr_{01}S_{01} - B_{01} + \sum_{i=1}^{I}(1-\gamma^s_i)X^s_{i01} - \sum_{i=1}^{I}(1+\gamma^b_i)X^b_{i01} = 0.$$

Asset inventory equation:

The asset inventory equation specifies that the amount invested in an asset class at the beginning of a period is equal to the amount invested at the end of the previous period adjusted for buying and selling.

$$(12)\quad X^h_{itn} = (1 + r_{itn})X^h_{i,t-1,\hat{n}} - X^s_{itn} + X^b_{itn},\ i = 1..I,\ t = 1..T-1,\ n = 1..N_t,$$

$$(13)\quad X^h_{i01} = X^{ini}_i - X^s_{i01} + X^b_{i01},\ i = 1..I.$$

Total asset value at the end of a period:

This equation specifies the total asset value at the end of a period, which is used to measure deficits.

$$(14)\quad A_{tn} = \sum_{i=1}^{I} X^h_{i,t-1,\hat{n}}(1 + r_{itn}) + (1 + r^l_{tn})m^l_{t-1,\hat{n}} - (1 + r^b_{tn})m^b_{t-1,\hat{n}},\ t = 1..T,\ n = 1..N_t.$$

Measuring deficits and final wealth:

Whenever at the end of the year the funding ratio is less than the minimum level F^{min}, deficits are measured and penalized in the objective. At the end of the horizon the funding ratio should be greater than F^{end}. The contribution rate cr^{end}_n is set to lift the funding ratio to this target level. In this way we avoid end effects and measure the final wealth of the pension fund.

$$(15)\quad A_{tn} \geq F^{min}L_{tn} - Z_{tn},\ t = 1..T,\ n = 1..N_t,$$

$$(16)\quad A_{Tn} \geq F^{end}L_{Tn} - cr^{end}_n S_{Tn},\ n = 1..N_T,$$

$$(17)\quad Z_{tn} \geq 0,\ t = 1..T,\ n = 1..N_t,\ cr^{end}_n \geq 0,\ n = 1..N_T.$$

Policy restrictions for the contribution rates:

The level and the change of the contribution rate are bounded, as specified by the pension fund.

(18) $cr^{lo} \leq cr_{tn} \leq cr^{up},\ t = 0..T-1,\ n = 1..N_t,$

(19) $dcr^{lo} \leq cr_{tn} - cr_{t-1,\hat{n}} \leq dcr^{up},\ t = 1..T-1,\ n = 1..N_t.$

Policy restrictions for the asset mix:

The weights of the asset mix are bounded, as specified by the pension fund.

(20) $w_i^{lo} \sum_{i=1}^{I} X_{itn}^h \ \leq\ X_{itn}^h \ \leq\ w_i^{up} \sum_{i=1}^{I} X_{itn}^h,\ i = 1..I,\ t = 0..T-1,\ n = 1..N_t.$

3 Decomposition

The size of stochastic programming models grows exponentially with the number of decision stages. In order to cope with this curse of dimensionality it is crucial to take advantage of the special structure of the problem with a suitable optimization algorithm. One can exploit the structure of these problems within a direct solution approach, e.g., an interior point method (Birge and Qi 1988, Jessup et al. 1994) or by a suitable decomposition method.

In this section we briefly review Benders decomposition applied to the two-stage stochastic program (6). This method is often referred to as the L-shaped decomposition (Van Slyke and Wets 1969). The basic idea of the L-shaped method is to construct an outer linearization of the expected recourse cost function $\mathcal{Q}(x) = E_{\tilde{\xi}}\left[Q(x,\tilde{\xi})\right]$. We present this method in a form that exploits the multicut version of decomposition (Ruszczyński 1984, Birge and Louveaux 1988), which is a crucial feature for fast convergence in practice.

Observe that problem (6) can be written in the equivalent form

(21) $$\min\{c'x + \sum_{j=1}^{N} Q_j(x) \mid Ax = b,\, x \geq 0\},$$

where $Q_j(x)$, $j = 1, 2, \ldots, N$, is the optimal objective function of the so-called recourse problem

(22) $$Q_j(x) = \min\{p_j q_j' y_j \mid W_j y_j = h_j - T_j x,\, y_j \geq 0\},$$

with the understanding that $Q_j(x) = +\infty$ if problem (22) is infeasible. The function $Q_j(x)$ is piecewise linear and convex.

Problem (22) is called the jth subproblem, $j = 1, 2, \ldots, N$. Note that this linear program depends on the first stage variables x, through the right hand side $h_j - T_j x$. For a given x, there are two possibilities if we try to solve the subproblem:

Case **1.**

Problem (22) has an optimal solution $\hat{y}_j$ and an associated dual optimal solution $\hat{u}_j$. Then we can construct the subgradient inequality for $Q_j(\tilde{x})$:

$$Q_j(\tilde{x}) \geq Q_j(x) - \hat{u}_j' T_j (\tilde{x} - x), \; \forall \tilde{x}. \tag{23}$$

This inequality is called an *optimality cut.*

Case **2.**

Problem (22) is infeasible. From duality in linear programming, we conclude that the dual of (22) is unbounded: there exists a ray $\hat{u}_j$ in the unbounded direction. In this case, the jth subproblem finds that the current first stage variable x is not feasible. Moreover, it generates the following valid inequality that has to be satisfied by all feasible first stage variables:

$$\hat{u}_j' T_j \tilde{x} \geq \hat{u}_j' h_j, \; \forall \tilde{x}. \tag{24}$$

This inequality is called a *feasibility cut.*

After solving the second stage subproblems we can compute the following upper bound for the optimal objective value of problem (6):

$$\bar{\theta}(x) = c' x + \sum_{j=1}^{N} Q_j(x) \tag{25}$$

Note that this upper bound will become $+\infty$ value if at least one of the problems (22) is infeasible for the current value of the first stage variables x.

Summing up, for a given x the jth subproblem generates a valid inequality that any feasible first stage variable x must satisfy. The inequalities (*cuts*) produced by all subproblems are appended to (21) thus restricting the feasible region of x. Assume that subproblems have already been solved for several first stage variables x (*query points*) $\{x^k\}_{k=1,2,\ldots,\kappa}$. For each of these points, every subproblem returned either the optimality cut (if for a given x^k the linear program (22) had a feasible solution) or the feasibility cut (if it had not).

Problem (21) can be replaced with the so-called *restricted master program*:

$$\begin{aligned} \min \quad & c'x + \theta, \\ s.t. \quad & Ax = b, \; x \geq 0, \end{aligned} \tag{26}$$

$$\begin{aligned} &\theta \geq \sum_{j=1}^{N} z_j, \\ &z_j \geq Q_j(x^k) - (\hat{u}_j^k)' T_j (x - x^k), && \forall j \leq N, \text{ and } \forall k \text{ such that } Q_j(x^k) < +\infty, \\ &0 \geq (\hat{u}_j^k)' h_j - (\hat{u}_j^k)' T_j x, && \forall j \leq N, \text{ and } \forall k \text{ such that } Q_j(x^k) = +\infty. \end{aligned}$$

The optimal solution $\hat{x}$ of this problem is a candidate for the next query point, while the optimal objective value is a lower bound $\underline{\theta}$ for the optimal value of problem (6).

Benders decomposition for two-stage stochastic program works in the following way. All subproblems (22) are solved for a given value of the first stage variable x^k. Each subproblem returns a cut which is appended to (26). The restricted master problem is solved producing $\hat{x}$ and the optimal objective $\underline{\theta}$. Its optimal solution $\hat{x}$ becomes the new query point x^{k+1} which is then sent to subproblems (22) and the process repeats. The algorithm continues until the distance between the upper bound $\bar{\theta}(x)$ of (25) and the lower bound $\underline{\theta}$ drops below a given optimality tolerance.

There exist a variety of decomposition approaches that can exploit the special structure of multistage stochastic programs: from classical Benders decomposition presented here (Van Slyke and Wets 1969), through its nested variant (Birge 1985), regularized decomposition (Ruszczyński 1984) or the augmented Lagrangian one (Mulvey and Ruszczyński 1995). Their discussion is beyond the scope of this paper; we refer the interested reader to books devoted to stochastic programming, e.g., Birge and Louveaux (1997), Kall and Wallace (1994).

3.1 Implementation of the L-shaped Decomposition Method

In this paper we apply straightforward Benders decomposition to solve the asset liability management problems. Subproblems in this decomposition approach are themselves linear programs of considerable sizes reaching tens of thousand of rows and columns; hence we use an interior point solver, in this case HOPDM of Gondzio (1995). The restricted master problems are also large linear programs, which suggests the use of interior point technology: we apply the Gondzio and Sarkissian (1996) primal-dual column generation method. This approach offers an easy control of the degree of optimality required in master solves. In the case of decomposition of structured linear programs it is usually advantageous to solve restricted master problems to optimality. We have done so in our computations.

For the sake of storage savings we had to disable the warm start facility of HOPDM (Gondzio 1998 and Gondzio and Vial 1999) and always solved subproblems from scratch. Storing advanced solutions would have required remembering a few vectors of dimension equal to the number of variables in the deterministic equivalent problem. Note that one vector of dimension nearly 25 million requires already 200 MB of memory to be saved. Certainly, exploiting warm start facility could reduce the overall solution time but at the cost of reducing the size of problems solved.

Classical Benders decomposition can be applied to dual block-angularly structured linear programs. Multistage stochastic linear programs display nested dual block-angular structure. However, by aggregating several initial stages into one (large) first stage, they can be converted into two-stage problems. The decomposition approach can take advantage of a good guess of the first stage variables. In order to get a reliable initial solution we solve a 3-period version of the 6-period financial planning model. This requires a negligible computational effort compared to the total solution time. The start-up approach stabilizes the algorithm and reduces the number of outer iterations (master solves).

Although many of earlier mentioned decomposition algorithms have been run on parallel computing platforms (Vladimirou and Zenios 1997), to the best of our knowledge, none of them have ever been applied to solve problems of sizes comparable to those of the problems discussed in this paper. An alternative strand of literature directly exploits the structure of multi-stage stochastic programs in the implementation of an interior point algorithm (Birge and Qi 1988, Jessup, Yang and Zenios 1994 and Czyzyk, Fourer and Mehrotra 1998). With this approach Yang and Zenios (1997) solve test-problems with up to 2.6 million constraints and 18.2 million variables. Interior point algorithms seem to be very effective for solving large-scale stochastic programming problems, both when they are adjusted to exploit structure directly as in Yang and Zenios (1997) and when they are applied to solve master- and subproblems in a decomposition method as in this paper.

4 Model Generation

The largest model reported in this paper is a financial planning problem with 7 decision stages and a total of 4,826,809 scenarios at the planning horizon. The deterministic equivalent of this program consists of 24,938,502 columns, 12,469,250 rows and 63,552,952 non-zero elements. Loading this deterministic equivalent problem directly into the LP solver HOPDM would require about 14159 MB of workspace. It seems obvious that efficient memory management is crucial for solving truly large scale models. Decomposition methods are potentially suited for efficient memory management, because subproblems are solved independently.

During the iterations of the L-shaped method there is no need to hold the entire deterministic equivalent problem in the memory of the (parallel) machine. The algorithm only needs to access the master problem and one second stage subproblem (for each processor) simultaneously. While one subproblem is solved and returns cut to the master, the other subproblems could be temporarily stored on an external storage device like a harddisk. However, for the huge models discussed in this paper the available disk-space might be insufficient. As a final resort, we will therefore generate the data of a subproblem each time when it is needed and throw it away immediately afterwards. Clearly, the generation of subproblems has to be done very efficiently in order to avoid dominating the total solution time.

In the approach used by Fragnière et al. (1998), the complete deterministic equivalent of the stochastic linear program was generated by the GAMS algebraic modeling language (Brooke et al. 1992). This problem was then passed to the parallel decomposition solver that split data across different processors. The generation of the deterministic equivalent problem and later splitting it into subproblems were clearly opposite operations that not only contributed

Table 1: Parameters of the Stochastic Programming Model

cr_{ini}	cr^{lo}	cr^{up}	dcr^{lo}	dcr^{up}	F^{ini}	F^{end}	w^{lo}	w^{up}	λ	γ^b	γ^s
0.12	-0.10	0.25	-0.08	0.04	1.00	1.15	0.00	1.00	4.00	0.01	0.01

significantly to the overall solution time but also created a huge peak memory requirement. In the approach presented in this paper a dedicated model generation tool, LEQGEN of Kouwenberg (1999), is used to avoid these drawbacks.

Although it is far less general than any commercial algebraic modeling language, our generator is optimized for speed and memory requirements when building stochastic linear programs. Moreover, it easily generates any user-specified subset of constraints or variables of the deterministic equivalent problem. This is a key feature used extensively in our integrated optimization environment when the decomposition code calls the generator to build only one specified subproblem. A detailed discussion of the LEQGEN design is beyond the scope of this paper. We refer the interested reader to Kouwenberg (1999).

To illustrate the efficiency of our model generation tool, we generated an instance of the asset liability model presented in Section 2, both with LEQGEN and with GAMS. The problem consists of 6 trading dates for rebalancing the investment portfolio (7 decision stages). Four branches emanate from each node in the event tree to describe the conditional return distributions of 3 asset classes and the wage inflation. More details about the implementation of the ALM model are provided in Section 5. The deterministic equivalent of this stochastic program consists of 17,741 rows, 35,484 columns and 92,800 non-zero elements in the constraint matrix. It is the smallest problem reported in this paper.

LEQGEN takes 2 seconds to generate the problem on a Pentium 200 Mhz PC with 64 MB of RAM, while GAMS needs 1630 seconds: this gives a speedup of 815 over GAMS, a commercial algebraic modeling language. More important, the speed of LEQGEN allowed us to regenerate subproblems during the iterations of a decomposition method at an acceptable overhead cost. Indeed, the time of generating subproblem never exceeds 28% of the solution time with the interior point code HOPDM.

5 Numerical Results

In this section we report the computational results of experiments with the multi-stage asset liability model for pension funds presented in Section 2. We solve ALM models with 6 trading dates (7 decision stages), separated by a one-year period. At each trading date the pension can invest in the following asset classes: bonds, stocks and real estate. Each transaction, either buying or selling, incurs a proportional cost of 1%. Each year the plan sponsor pays contributions to the pension fund, in order to fund current and future pensions. Both the level and the yearly change of the contribution payments are restricted, in order to prevent large fluctuations. The objective of the model is to minimize the payments by the plan sponsor, while penalizing deficits. The parameters of the model are displayed in Table 1.

The multi-stage ALM model needs as input an event tree with asset returns and actuarial information about the liabilities of the pension fund. In order to generate the actuarial scenarios we use the simulation system of Boender (1997) for projecting the careers of the participants. This model is widely used by Dutch pension funds. In order to generate economic scenarios, the returns of the assets and the wage inflation are modeled with a Vector Auto-Regressive model, which was calibrated using 40 years of historical data (see Boender 1997 and Kouwenberg 1998).

We construct the event for the stochastic programming model with the tree fitting method of Høyland and Wallace (1999) and Kouwenberg (1998). Let p denote the number of successors in each node of the event tree, that can be used for approximating the conditional return distribution. We solve a small non-linear optimization model to fit the mean, the covariance, the skewness and the kurtosis of the return distribution with the available p nodes. The decision variables in the non-linear model are the asset returns for the p nodes, while the probabilities are fixed at $1/p$. For some of the smaller models the skewness and the kurtosis could not be fitted, due to an insufficient number of degrees of freedom $(4 \times p)$. Consequently, we expect that the quality of the solutions will increase with the number of succeeding nodes p.

Kouwenberg (1998) analyses the performance of a 5-period version of the ALM-model with rolling horizon simulations, while applying three different scenario generation methods. The results show that the scenario generation procedure has a major impact on the solutions of the model. Moreover, fitting the first few moments of the return distribution greatly improves the performance of the model. If we want to fit higher order moments, include more asset classes in the model or increase the number of trading dates, then we require larger event trees and consequently the size of the stochastic programming models increases. We will now demonstrate that an integrated high-performance computing approach can be applied to solve truly large scale ALM problems.

The models are solved on a PARSYTEC CC16 parallel machine, consisting of 16 Motorola PC-604 RISC processors.[1] Each processor has 64 MB of memory, except for the two input/output entry nodes which have 128 MB. We use the PVM3 protocol to run parallel applications and to pass information between the master and the subproblems. Although we have done our computations on a true parallel machine, our code is easily portable to any distributed computing environment, including a cluster of PC's running under the Linux operating system.

5.1 Model Sizes

Different sizes of the ALM problem are created by varying the number of branches p emanating from each node in the event tree for the 6-period model. Table 2 reports the size of the deterministic equivalent linear program. This size refers to a standard linear program with equality constraints: slacks are added to the inequalities and free variables are split. The large scale models reported in Table 2 not only require considerable computational effort to be solved, they also need a huge amount of workspace to be stored. The second column of Table 3 reports the

[1]We thank the University of Cyprus for providing us access to their parallel machine, funded by the EU-project *High Performance Computing for Financial Planning under Uncertainty.* Specifically, we would like to thank Hercules Vladimirou and Andrea Consiglio.

Table 2: Problem sizes of the financial planning model

Nodes p	Scenarios p^6	Rows	Columns	Non-zeros
4	4,096	17,741	35,484	92,800
5	15,625	58,586	117,174	304,648
6	46,656	158,623	317,248	821,108
7	117,649	372,548	745,098	1,921,564
8	262,144	786,425	1,572,852	4,044,472
9	531,441	1,527,886	3,055,774	7,838,720
10	1,000,000	2,777,771	5,555,544	14,222,188
11	1,771,561	4,783,208	9,566,418	24,447,508
12	2,985,984	7,872,133	15,744,268	40,175,024
13	4,826,809	12,469,250	24,938,502	63,552,952

estimate of the memory that the interior point code HOPDM would require if it was applied to solve the deterministic equivalent linear programs. The memory requirement includes all workspace needed for storing the problem data and the Cholesky factors. Direct solution of the largest problem with the interior point solver would have required, for example, more than 14 GB of memory. This greatly exceeds the capacities of the majority of today's computers.

In order to apply two-stage Benders decomposition, we divide the 6 period problem into a first stage problem from time 0 up to time $T_{sub} - 1$ and a set of second stage subproblems from time T_{sub} up to the planning horizon T. In the third column of Table 3 we report the HOPDM memory requirements for one second stage subproblem starting at time $T_{sub} = 2$. A subproblem of the 13-node problem starting at time 2 consists of $13^4 = 28,561$ scenarios and interior point solver requires about 83.79 MB of workspace to solve it. There are $13^2 = 169$ subproblems to deal with.

Alternatively, we could choose to split up the problem at time $T_{sub} = 3$. In that case the HOPDM memory requirement for a subproblem of the 13-node model drops to only 6.46 MB (a linear program corresponds to $13^3 = 2,197$ scenarios). Note however that there are $13^3 = 2,197$ of these second stage subproblems at time 3. Storing these subproblems simultaneously might still be infeasible, because it requires about 15 GB of memory. As an alternative, we could use LEQGEN to generate a subproblem when it is needed by the L-shaped algorithm and throw it away immediately afterwards. This is the approach used to solve the largest problems.

5.2 Model Generation and Solution Times

We solved the 6-period ALM problems reported in Table 2 with the L-shaped method on a Parsytec CC16 parallel machine with 16 processors.[2] In panel A of Table 4 we report the solution times of models that are split into a first stage master problem and a set of second stage subproblems starting at time 2. The third column of Table 4 shows the number of parallel processes used for solving the subproblems, while the fifth column reports the solution time in seconds needed by the decomposition solver (excluding model generation overhead).

[2]All problems are solved to the relative accuracy $\epsilon = 10^{-6}$.

Table 3: HOPDM workspace

Nodes p	Scenarios p^6	Deterministic LP	Subproblem at time 2	Subproblem at time 3
4	4,096	20.30 Mb	1.28 Mb	0.33 Mb
5	15,625	66.93 Mb	2.69 Mb	0.55 Mb
6	46,656	180.96 Mb	5.04 Mb	0.85 Mb
7	117,649	424.55 Mb	8.67 Mb	1.25 Mb
8	262,144	895.42 Mb	14.00 Mb	1.77 Mb
9	531,441	1738.39 Mb	21.47 Mb	2.40 Mb
10	1,000,000	3158.56 Mb	31.59 Mb	3.18 Mb
11	1,771,561	5436.10 Mb	44.94 Mb	4.10 Mb
12	2,985,984	8942.67 Mb	62.11 Mb	5.19 Mb
13	4,826,809	14159.45 Mb	83.79 Mb	6.46 Mb

As a first approach, the subproblems are generated before the start of the decomposition algorithm and stored in memory during the iterations. The model generation time in panel A refers to the time needed by LEQGEN to generate the subproblems on each processor before the start of the algorithm. For the problems with $p \geq 8$ the available memory was insufficient to store all subproblems and therefore the problems could not be solved. In the second experiment we exploited LEQGEN's ability to generate subproblems quickly: we report these results in panel B of Table 4. In these runs, every subproblem was regenerated at each outer iteration of the decomposition algorithm. Once the subproblem was solved and produced a cut, the whole data of this subproblem was discarded.

The sixth column in panel B of Table 4 reports the total time in seconds spent by LEQGEN to generate subproblems during the iterations of the decomposition algorithm, on one processor of the parallel machine. In all cases the overhead for regenerating the subproblems during the iterations is less than 28% of the solution time needed by the decomposition code (excluding model generation). Note however that the approach results in a huge increase of the size of models which can be solved: the largest problem corresponds to $p = 11$. For $p \geq 12$ the memory requirement for a second stage subproblem at time 2 is too big (see Table 3), so we could not report the results.

In order to limit the memory requirements even further, we split the 6-period financial planning problem into a first stage master problem and a set of second stage subproblems at time 3. The computational results of this experiment are reported in panel C of Table 4. As before, the subproblems are regenerated during each iteration of the decomposition algorithm. Due to the relatively small sizes of the second stage subproblems at time 3, we can now solve models with up to $p = 13$ realizations at every node of the event tree on the parallel machine.

In the case of the largest problem, the algorithm solved $13^3 = 2,197$ subproblems 8 times, i.e. a total of 17,576 linear programs were solved in the overall time of 14,138 seconds. The generation of these problems took only 2,294 seconds (both the generation and solution were executed in parallel on 13 processors). Note that the overhead of regenerating the subproblems with LEQGEN is limited: the generation time never exceeds 28% of the solution time with the

Table 4: Model Generation and Solution Times

Nodes p	Subproblems	Parallel processes	Outer iterations	Decomposition solution time	LEQGEN model generation time
A. All subproblems stored in memory, second stage starts at time 2.					
4	16	4	11	62	3
5	25	5	9	203	4
6	36	6	9	393	7
7	49	7	10	1194	13
≥ 8	64	8	NA	NA	NA
B. Subproblems regenerated during the iterations, second stage starts at time 2.					
4	16	4	11	68	32
5	25	5	9	219	37
6	36	6	9	422	64
7	49	7	10	1166	128
8	64	8	10	2329	221
9	81	9	9	4261	339
10	100	10	10	9644	572
11	121	11	10	16102	1213
≥ 12	144	12	NA	NA	NA
C. Subproblems regenerated during the iterations, second stage starts at time 3.					
4	64	4	11	78	56
5	125	5	11	235	114
6	216	6	10	380	182
7	343	7	10	767	277
8	512	8	10	1438	408
9	729	9	10	2612	614
10	1000	10	11	4876	988
11	1331	11	9	6296	1311
12	1728	12	9	10256	2071
13	2197	13	8	14138	2294

decomposition code for problems with $p \geq 8$. In other words, an increase of about 30% of the overall CPU time needed by the integrated generation-optimization approach is a small price to pay for the ability of solving truly large problems, compared with an approach in which the whole problem is generated once and stored throughout the whole solution process.

5.3 Efficiency of the Parallel Implementation

In this section we report the speed-ups achieved by the parallel code. Due to the large number of subproblems, the effort needed to solve the restricted master problems is negligible. Hence only the subproblems are solved in parallel. The restricted master problem is solved by the first processor, while the other processors remain idle temporarily. Observe that the communication between different processors is restricted to the minimum: the processor that handles the master problem sends the query point to the remaining processors and receives cuts back from them.

We solve the financial planning model with six branches in each node of the event tree ($p = 6$) on 1 up to 12 processors of the parallel machine. This stochastic programming problem has 46,656 scenarios, which results in a deterministic equivalent linear program with 158,623 rows and 317,248 columns. HOPDM requires approximately 181 MB of memory to solve the deterministic equivalent LP. We split the model into a first stage problem and a set of second stage problems starting at time 2, so there are 36 subproblems which return cuts to the master.

Table 5 shows the solution times and speed-ups if we divide the workload of solving the 36 subproblems over 1 up to 12 processors. With 3 processors or more we generate the 36 subproblems before the start of the algorithm and store each subproblem in the distributed memory of the parallel machine during the iterations. With 2 processors or 1 processor this approach fails due to lack of memory, so we regenerate the subproblems during the iterations of the decomposition algorithm.

The third column of Table 5 shows the speed-ups of our decomposition solver run on multiple processors, relative to the solution time on one processor. With more than 4 processors these speed-ups are super-linear. However, these results are due to the model-generation procedure. With 1 or 2 processors the subproblems are regenerated during the iterations of the algorithm and this causes some additional overhead which is included in the solution time. With 3 processors the available memory is insufficient and disk swapping slows the solution down.

We get a better view of the speed-ups by using the solution time with 4 processors as a benchmark (reported in the fourth column of Table 5). In this case the speed-ups are slightly worse than linear for more than 5 processors, which might be expected as the communication between the master and the subproblems increases. Moreover, if the number of subproblems is not a multiple of the number of processors then the workload is not divided equally. We expect imperfect speed-ups in these cases (i.e. for 5,7,8,10 or 11 processors), as the master cannot solve the first stage problem until the cuts of each subproblem have been returned. However, better parallel performance could be obtained by splitting up the multi-stage problem into a larger number of second stage subproblems.

5.4 Benefits of Solving Larger Models

In the previous sections we have concentrated on the computational aspects of solving large scale models. A crucial issue that remains to be addressed is whether solving huge models is truly beneficial for practical asset liability management problems. Theoretically, one would

Table 5: Speed-up of decomposition on the parallel machine

Processors	Solution time	Speed-up w.r.t 1 processor	Speed-up w.r.t 4 processors
1	2810	1	-
2	1384	2.03	-
3	880	3.19	-
4	570	4.93	4
5	497	5.65	4.59
6	396	7.10	5.76
7	370	7.59	6.16
8	320	8.78	7.13
9	264	10.64	8.64
10	260	10.81	8.77
11	257	10.93	8.87
12	205	13.71	11.12

expect that solving large scale models is important as it allows for more fine-grained event trees and more future decision dates. Multiple decision dates are required to incorporate the dynamic aspect of the decision problem, while fine-grained event trees are necessary to reduce the approximation error relative to the true underlying return distributions.

Concentrating on the scenario generation issue, Kouwenberg (1998) shows that the approximation error in the event tree can seriously influence the optimal decisions of the ALM model for pension funds. Given a 5-period version with 5760 scenarios of the ALM model in Section 2, Kouwenberg (1998) compares scenario generation methods based on random sampling, adjusted random sampling and tree-fitting with rolling horizon simulations. The results indicate that the tree-fitting method and the adjusted random sampling method lead to better ALM performance than straightforward random sampling. With the tree-fitting method for scenario generation the stochastic programming model could outperform a fixed mix benchmark model in terms of risk and return.

Given these results, in this paper we have employed tree-fitting to construct the event trees for the large scale ALM models. For all event trees we could fit the mean and the standard deviation of the underlying normal return distributions (i.e. wage growth and the return on bonds, stocks and real estate). For the event trees with $p \geq 5$ nodes we could also fit the correlations between the series, for $p \geq 6$ we could include skewness (3rd moment) and finally for $p \geq 11$ we could also match the kurtosis perfectly (4th moment). As a result, we expect that the quality of the optimal solutions will improve with the number of nodes in the event tree.

In Figure 2 we display the optimal initial asset mix and the objective value of the ALM models with $4 \leq p \leq 13$ nodes. If we concentrate on the smaller models with $p \leq 8$, then it is clear that the optimal solution changes drastically from one model to the next. However, if we look at the larger models with $p \geq 9$ then the solution seems to stabilize: the weight of bonds varies between 62% and 78%, the stock weight is between 19% and 38%, while real estate is mostly absent in the asset mix. A possible explanation for these results might be the increasing fit to

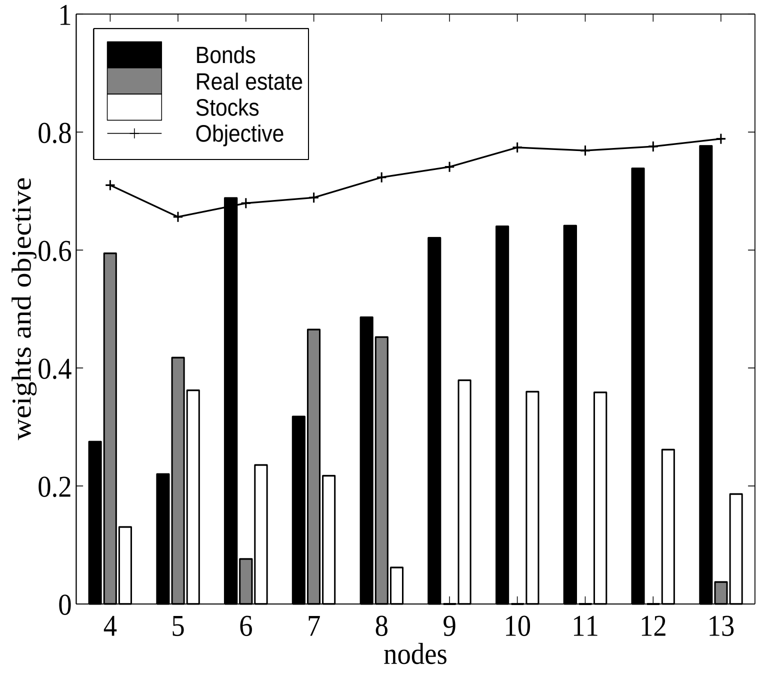

Figure 2: Optimal Initial Solution and Objective of the ALM Model

the kurtosis (fat-tailedness) of the underlying distributions for larger event trees ($p \geq 9$).

With a small number of nodes the kurtosis of the underlying normal return distribution is underestimated in the event tree. In order to match the kurtosis it is necessary to include at least one large outlier in the set of nodes (e.g. a return of -21% for stocks). In order to preserve the symmetry of normal distribution at the same time (i.e. match the skewness), a negative outlier must be compensated with a corresponding positive outlier. With a small number of nodes it becomes very hard to include two outliers, while additionally fitting the mean and the variance of the distribution. With more nodes ($p \geq 9$) the kurtosis is fitted more accurately, while the worst case negative return in the scenarios tends to become more negative. It seems that only in these large event trees the risk of stocks and real estate is represented properly and hence the increasing allocation to the safe asset class bonds.

The explanation for the results in Figure 2 is tentative. It is clear however, that solving large scale models ($p \geq 9$) could be crucial for getting a reliable solution of the ALM model. In this example the solutions of the smaller models ($p \leq 8$) seem to be partly misleading, due to approximation errors in the small-sized event tree with asset returns. A full investigation of the convergence and scenario generation issues for stochastic programming models is beyond the scope of this paper. We hope however that our solution approach can provide the computational means to address these relevant questions in the future.

6 Conclusions

Dynamic asset liability management problems form an important class of real-life financial planning problems that require sophisticated techniques for risk management. Stochastic programming is one of the possible approaches that can be used to model these problems. Taking into account both uncertainty (approximating the conditional distribution of the asset returns) and the dynamic structure of the decision problem (multiple portfolio rebalancing dates) inevitably leads to an explosion of dimensionality in stochastic programming models for ALM.

This paper shows that very large instances of stochastic programming models for ALM can be solved if an efficient model generation tool and a decomposition technique for optimization are integrated in a high-performance computing environment. We solve these stochastic programming problems with a two-stage Benders decomposition algorithm that employs the interior point solver HOPDM to optimize both the restricted master problem and the second stage subproblems.

The most important finding in this paper is that given a good solution technique, the bottleneck in solving truly large problems is the available amount of memory. In order to increase the efficiency of memory management, we have developed the specialized modeling tool LEQGEN for generating stochastic linear programs. The efficiency and speed of this code allow us to regenerate subproblems during the iterations of the Benders decomposition algorithm, which results in a drastic decrease of memory use. The repeated generation of subproblems allows much larger problems to be solved and increases the overall solution time by no more than 28%.

Our integrated generation-optimization code was run on a PARSYTEC CC16 machine, using at most 13 processors with 64 MB of RAM each. On this particular parallel computer we could solve a problem with 12,5 million rows and 25 million columns in less than 5 hours. It is important, however, to note that the integrated system we developed could be run on any distributed computing platform including a cluster of PC's, a technology certainly affordable for any company.

A closer look at the optimal solutions of the ALM model revealed that the initial asset weights of stocks, bonds and real estate varied significantly between the smaller models with up to 8 nodes. The initial solutions of the larger models (with more than 8 nodes) changed in a regular and relatively stable way, probably due to reduced approximation errors in the event tree with asset returns. This result demonstrates the potential benefit of solving large stochastic programming models for practical purposes. More research in the interesting areas of convergence and scenario generation is certainly needed and we hope that the high-performance computing approach developed in this paper will provide the required tools.

References

[1] Ariyawansa K.A. and D.D. Hudson (1991), Performance of a Benchmark Parallel Implementation of the Van Slyke and Wets Algorithm for Two-stage Stochastic Programs on the Sequent/balance, *Concurrency: Practice and Experience*, vol 3, 109–128.

[2] Benders J.F. (1962), Partitioning Procedures for Solving Mixed-Variables Programming Problems, *Numerische Mathematik*, vol. 4, 238-252.

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

[4] Birge J.R. (1985), Decomposition and Partitioning Methods for Multistage Stochastic Linear Programs, *Operational Research*, vol. 33, 989-1007.

[5] Birge J.R. and F. Louveaux (1988), A Multicut Algorithm for Two-Stage Stochastic Linear Programs, *European Journal of Operational Research*, vol. 34, 384 - 392.

[6] Birge J.R. and F. Louveaux (1997), *Introduction to Stochastic Programming*, Springer-Verlag, New York.

[7] Birge J.R. and L. Qi (1988), Computing Block-angular Karmarkar Projections with Applications to Stochastic Programming, *Management Science*, vol. 34, 1472-1479.

[8] Boender G.C.E (1997), A Hybrid Simulation/optimisation Scenario Model for Asset Liability Management, *European Journal of Operational Research*, vol. 99, 126-135.

[9] Bradley S.P. and D.B. Crane (1972), A Dynamic Model for Bond Portfolio Management, *Management Science*, vol. 19, 139-151.

[10] Brooke A., D. Kendrick and A. Meeraus (1992), GAMS: A User's Guide, *The Scientific Press*, Redwood City, California.

[11] Carino D.R., T. Kent, D.H. Myers, C. Stacy, M. Sylvanus, A.L. Turner, K. Watanabe and W.T. Ziemba (1994), The Russel-Yasuda Kasai Model: An Asset Liability Model for a Japanese Insurance Company Using Multi-stage Stochastic Programming, *Interfaces*, vol. 24, 29-49.

[12] Consigli G. and M.A.H. Dempster (1998), Dynamic Stochastic Programming for Asset-Liability Management, *Annals of Operations Research*, vol. 81, 131-162.

[13] Czyzyk J., R. Fourer and S. Mehrotra (1998), Using a Massively Parallel Processor to Solve Large Sparse Linear Programs by an Interior-Point Method, *SIAM Journal on Scientific Computing*, vol. 19, 553-565.

[14] Fragnière E., J. Gondzio and J.-P. Vial (1998), Building and Solving Large-scale Stochastic Programs on an Affordable Distributed Computing System, Logilab Technical Report 98.11, Department of Management Studies, University of Geneva, Switzerland.

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

[16] Gondzio J. (1995), HOPDM (version 2.12) - A Fast LP Solver Based on a Primal-Dual Interior Point Method, *European Journal of Operational Research*, vol. 85, 221-225.

[17] Gondzio J. (1998), Warm Start of the Primal-Dual Method Applied in the Cutting Plane Scheme, *Mathematical Programming*, vol. 83, 125-143.

[18] Gondzio J. and R. Sarkissian (1996), Column Generation with a Primal-Dual Method, Logilab Technical Report 96.6, Department of Management Studies, University of Geneva, Switzerland.

[19] Gondzio J. and J.-P. Vial (1999), Warm Start and Epsilon-subgradients in the Cutting Plane Scheme for Block-angular Linear Programs, *Computational Optimization and Applications*, vol. 14, 1-20.

[20] Høyland K. and S.W. Wallace (1999), Generating Scenario Trees for Multi Stage Problems, *Management Science*, to appear.

[21] Jessup E.R., D. Yang and S.A. Zenios (1994), Parallel Factorization of Structured Matrices Arising in Stochastic Programming, *SIAM Journal on Optimization*, vol. 4, 833-846.

[22] Kall P. and S.W. Wallace (1994), *Stochastic Programming*, John Wiley & Sons, Chichester.

[23] Kouwenberg R. (1998), Scenario Generation and Stochastic Programming Models for Asset Liability Management, working paper, Erasmus Centre for Financial Research, Erasmus University Rotterdam (to appear in *European Journal of Operational Research*).

[24] Kouwenberg R. (1999), LEQGEN: A C-tool for Generating Linear and Quadratic Programs, User's Manual, Working Paper, Econometric Institute, Erasmus University Rotterdam, The Netherlands.

[25] Kusy M.I. and W.T. Ziemba (1986), A Bank Asset and Liability Model, *Operations Research*, vol. 34, 356-376.

[26] Mulvey J. and A. Ruszczyński (1995), A New Scenario Decomposition Method for Large Scale Stochastic Optimization, *Operations Research*, vol. 43, 477-490.

[27] Mulvey J.M. and H. Vladimirou (1992), Stochastic Network Programming for Financial Planning Problems, *Management Science*, vol. 38, 1642-1664.

[28] Mulvey J.M. and W.T. Ziemba (1998), Asset and Liability Management Systems for Long-Term Investors: Discussion of the Issues, in *Worldwide Asset and Liability Modelling*, eds: J.M. Mulvey and W.T. Ziemba, Cambridge University Press, 3-38.

[29] Ruszczyński A. (1984), A Regularized Decomposition Method for Minimizing a Sum of Polyhedral Functions, *Mathematical Programming*, vol. 33, 309-333.

[30] Van Slyke R. and R.J.B. Wets (1969), L-shaped Linear Programs with Application to Optimal Control and Stochastic Programming, *SIAM Journal of Applied Mathematics*, vol. 17, 638-663.

[31] Vladimirou H. and S.A. Zenios (1997), Parallel Algorithms for Large-Scale Stochastic Programming, in *Parallel Computing in Optimization*, eds.: Migdalas S., P. Pardalos and S. Storøy, Kluwer Academic Publishers, 413-469.

[32] Yang D. and S.A. Zenios (1997), A Scalable Parallel Interior Point Algorithm for Stochastic Linear Programming and Robust Optimization, *Computational Optimization and Applications*, vol. 7, 143-158.

[33] Zenios S.A. (1995), Asset/Liability Management under Uncertainty for Fixed-Income Securities, *Annals of Operations Research*, vol. 59, 77-97.

Appendix A: Dictionary for the ALM Model

Definition of variables:

A_{tn}	:	portfolio value at the end of year t,
X^h_{itn}	:	amount invested in asset class i at the beginning of year t,
X^b_{itn}	:	amount bought of asset class i at the beginning of year t,
X^s_{itn}	:	amount sold of asset class i at the beginning of year t,
m^l_{tn}	:	cash lent at the beginning of year t,
m^b_{tn}	:	cash borrowed at the beginning of t,
cr_{tn}	:	the contribution rate at the beginning of year t,
Z_{tn}	:	deficit relative to the minimum funding ratio in year t,
cr^{end}_n	:	contribution rate at the planning horizon needed to lift the funding level to F^{end}.

Definition of random coefficients:

B_{tn}	:	benefit payments in year t,
L_{tn}	:	liabilities at the end of year t,
S_{tn}	:	total wages of the participants at the beginning of year t,
r_{tn}	:	return on risky assets in year t,
r^l_{tn}	:	lending rate in year t,
r^b_{tn}	:	borrowing rate in year t.

Definition of model parameters:

X^{ini}_i	:	initial amount invested in asset class i,
m^{ini}	:	initial cash position,
cr^{ini}	:	initial contribution rate,
$cr^{lo/up}$	:	lower/upper bound of the contribution rate,
$dcr^{lo/up}$	:	lower/upper bound of the decrease/increase of the contribution rate,
$\gamma^{b/s}_i$	:	proportional transaction costs on buying/selling of asset class i,
F^{min}	:	minimum funding ratio,
F^{end}	:	target funding ratio at the planning horizon,
$w^{lo/up}_i$	:	lower/upper bound of the weights of the asset mix,
λ	:	parameter for specifying risk aversion.