Solving Distribution Planning Problems with the Interior Point Method*

Jacek Gondzio[†] Andreas Grothey[‡]

Technical Report, MS-06-001

School of Mathematics
The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ
United Kingdom.

February 18th, 2006, revised May 25th, 2006

*Supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/R99683/01.
[†]Email: J.Gondzio@ed.ac.uk, URL: http://maths.ed.ac.uk/~gondzio/
[‡]Email: A.Grothey@ed.ac.uk, URL: http://maths.ed.ac.uk/~agr/

Solving Distribution Planning Problems with the Interior Point Method

Abstract

A real-life distribution planning problem is dealt with in this paper. Modelling of this problem combines the use of graphs to describe the topology of the distribution network, dynamics to capture multi-periodicity of the planning, and uncertainty of the future demands. The combination of these factors causes the (overall) problem to be non-trivial and defy standard optimization software. The problem is an example of an Operations Research application which needs a dedicated solution approach. In this paper the problem has been modelled in such a way that its complicated structure can be exploited by a specialised optimization tool: its main difficulty has been converted into an advantage. The use of structure-exploiting modelling and solution methodology has enabled solving larger instances of the problem and has shed new light on the interpretation of results. The use of OOPS (Object-Oriented Parallel Solver) was crucial in achieving these goals.

1 Introduction

Planning problems are at the heart of Operations Research. They include every day operations of companies facing uncertainty of the market and adjusting production to changing needs of customers. Increasing computational power allows for more complicated problems to be dealt with. However, the need to model reality with a high degree of accuracy challenges existing modelling and optimization tools. The use of algebraic modelling languages such as AIMMS [6], AMPL [9] or GAMS [7] is helpful. It facilitates the development and testing of prototype models and occasionally allows for running the (smaller) real-life production models. Modelling languages access standard optimization software and this software has been subject of impressive progress in the past decade. This resulted in the development of many tools able to solve small to medium optimization problems with sizes reaching tens or hundreds of thousand (occasionally millions) of variables. Convenient access to optimization codes offered by algebraic modelling languages comes with a high price: these tools interface to *general* solvers and in consequence any particularities in the model which could have facilitated the solution process are lost in the generation phase.

We have developed a structure-exploiting optimization code called OOPS (Object-Oriented Parallel Solver) [14, 12]. OOPS (cf `http://maths.ed.ac.uk/~gondzio/parallel/solver.html`) is an implementation of the primal-dual interior point method and uses all recent algorithmic advances including multiple centrality correctors. What distinguishes it from general optimization solvers such as Cplex, LoQo, PCx or Xpress is its ability to exploit the problem structure in the linear algebra of interior point method. Its design features have been discussed in detail in [14, 12] and we will not review them in this paper. The modern design of OOPS payed off in the ability to solve problems of unequaled size exceeding one billion variables. There is neither hope nor sense to generate such problems with current algebraic modelling languages. These problems have to be generated by specialised optimization-driven modelling tools.

Attempts have been made to extend modelling languages and retrieve the underlying structure of the problem. The development of the Structure Exploiting Tool (SET for short) [10] addressed this issue in detail. SET is an addition to a modelling language which appends structural information to the general model produced by the modelling language and passes this information to the solver. We have developed a simple interface (SPI - Structure Passing Interface) to OOPS along these lines, which makes it possible to retrieve the structure in an optimization model and pass it to OOPS. We have used this integrated modelling and optimization tool to solve some otherwise intractable distribution planning problems. Another advantage coming naturally with the combined use of SPI/OOPS is the access to parallelism and the ability to use high performance computing both during the generation of the model and during the following solution of the problem.

We illustrate this approach with a case study of a real-life distribution planning problem. The model is proprietary and the Company has not agreed for us to reveal its details. Therefore we will use a generic description of this class of problems and illustrate advantages resulting from the use of different reformulations of it. These different reformulations are fed into OOPS and lead to very different behaviour and efficiency of the solver. This shows very clearly the scope for better modelling, the importance of being able to reformulate the model and finally the key role of the integrated modelling/optimization approach which enables structure exploitation.

The paper is organised as follows. In Section 2 generic structures which are combined in the distribution planning model are presented. In Section 3 the linear algebra techniques employed by the interior point method in the solution of problems with these particular structures are discussed. In Section 4 a generic distribution planning model is presented, specific features of the problem being at the origin of exploitable structures are discussed and the structures obtained in different reformulations are displayed. Our integrated modelling approach of combining OOPS with a modelling language via a SET-like approach is described in more detail in Section 5. In Section 6 the computational results are reported for attempts to solve different formulations of the problem, while finally in Section 7 we draw our conclusions.

2 Generic Structures in Distribution Planning Problem

Modelling generic distribution problems combines networks, dynamics and uncertainty. The reader interested in a background material in this area may consult [15, 17]. We discuss generic features which are used in the modelling of distribution planning problems and demonstrate the associated structures.

Networks
Graphs are commonly used to describe the topology of distribution networks [1]. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with m nodes $\mathcal{V}$ and n (directed) arcs $\mathcal{E} \subset \{(i,j) : i \in \mathcal{V}, j \in \mathcal{V}, i \neq j\}$. Let $A \in \mathcal{R}^{m\times n}$ be its node-arc incidence matrix. Each column k corresponds to an arc (i,j) with entries a_{ke} defined as

$$a_{ke} = \begin{cases} -1 & \text{if } k = i \\ 1 & \text{if } k = j \\ 0 & \text{if } k \neq i \text{ and } k \neq j. \end{cases}$$

Let $x \in \mathcal{R}^n$ be a network flow, where x_{ij} is the flow passing through arc (i,j). The general flow constraint can then be expressed in a form of linear equation $Ax = b$. Its right-hand-side vector $b \in \mathcal{R}^m$ is the generalized demand/supply. It is common to deal with distribution centers (depots) which are large supply points and numerous demand nodes which receive the goods shipped through the network.

The node-arc incidence matrix displays a specific *structure*: All its entries are $0, +1$ or -1 and each column contains only two nonzero entries. This structure may be exploited by the solution algorithm. Indeed, there exist many specialized approaches which originate from standard methods and make good use of these features. For example, the simplex method exploits the fact that the basis matrices are triangular and can be represented as spanning trees in the graph [1].

Dynamics
Processes evolve in time and in the simplest case (discrete, linear, and stationary) their evolution can be described by the following equation

$$x_{t+1} = Ax_t + Bu_t,$$

where x_t and u_t are the state and control variables at time $t \in \mathcal{T}$ and A and B are matrices of appropriate dimensions [3]. More generally the state vector x_{t+1} may depend on states of several previous time periods, matrices involved in the equation may depend on time, etc and the dynamics are described by a more complicated equation:

$$x_{t+1} = A_{t,t+1}x_t + A_{t-1,t+1}x_{t-1} + \cdots + A_{t-p,t+1}x_{t-p} + B_t u_t.$$

After concatenating the state and control variables into one long vector $(x_1, u_1, \ldots, x_{|\mathcal{T}|}, u_{|\mathcal{T}|})$, the constraints of the model can be expressed in a usual linear programming form. The constraint matrix displays a well-known *staircase structure* as shown in Figure 1. The left figure corresponds to simple dynamics (without time lag) and the right figure corresponds to a situation with time lag $p = 2$ and displays larger inter-temporal overlaps. Note that a problem with a lag of $p > 1$ can be reformulated into one with $p = 1$ at the expense of introducing additional variables. Again such structures can be exploited by optimization algorithms, for example the simplex method can take advantage of them in the pricing and basis inverse representation [8].

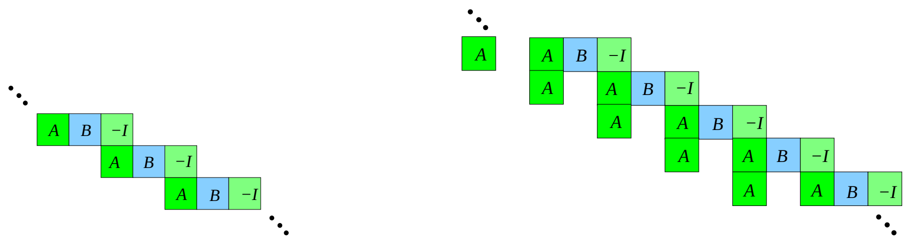


Figure 1: Staircase structure resulting from modelling of dynamics.

Uncertainty
Planning problems reflect attempts to hedge against uncertainty and are well-suited to the use of stochastic programming approaches [16]. In a simple two-stage model, a first stage decision has to be made immediately and a recourse action is postponed to stage two. This is described by the following equation

$$T_i x^1 + W_i y_i = b_i, \qquad i = 1, \ldots, S$$

where the first stage decision x^1 is common for all possible scenarios while the recourse actions $y_1, y_2, \ldots, y_S$ depend on the scenario $i = 1, 2, \ldots, S$. The decision x^1 is made in the initial stage and the second stage decisions y_i adjust to (uncertain) future described with a finite number of scenarios $\{(T_i, W_i, b_i),\ i = 1, 2, \dots, S\}$. In a more complicated situation where the decision process expands into multiple stages, event trees are used to describe the unfolding of the uncertainty over the planning period [16]. The dynamics are then captured by the equation

$$T_{l_t} x_{a(l_t)} + W_{l_t} x_{l_t} = b_{l_t},$$

where l_t is a node of the event tree in stage t and $a(l_t)$ is its ancestor (a node in stage $t-1$). A stochastic programming problem formulated as a deterministic equivalent is usually a large linear program with specially structured constraint matrix. The constraint matrix displays a nested *block-angular structure* as shown in Figure 2. The left figure corresponds to a two-stage problem and the right figure corresponds to a multi-stage one (three-stage). Again, such structures can be exploited by optimization algorithms, for example through decomposition approach [4, 11] or directly in the interior point algorithm [5, 12].

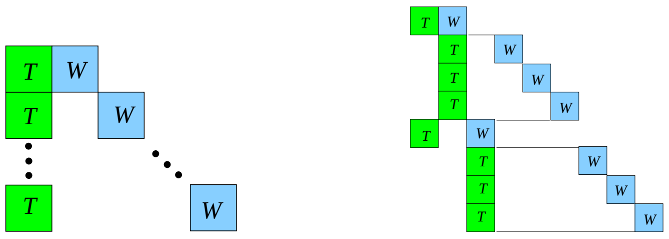


Figure 2: Block-angular structure resulting from modelling of uncertainty.

In the following section we give a brief computational view of interior point methods and show how the structures discussed in this section are exploited in the implementation of the linear algebra kernel of OOPS [14, 12].

3 Linear Algebra Techniques in Interior Point Methods

Over the years, interior point methods for linear optimization have proved to be a very powerful technique. We review basic facts of their implementation in this section and show how their implementation can take advantage of two particular structures resulting from the presence of dynamics and modelling of uncertainty in the model. The reader interested in the theoretical background of interior point methods should consult [18]; practical aspects of these algorithms are discussed in [2] and the references therein.

Consider the linear programming problem

$$\min \quad c^T x \quad \text{s.t.} \quad Ax = b,\ \ x \geq 0,$$

where $A \in \mathcal{R}^{m \times n}$ is the full rank matrix of linear constraints and vectors x, c and b have appropriate dimensions. The usual transformation in interior point methods consists in replacing inequality constraints with the logarithmic barriers to get

$$\min \quad c^T x - \mu \sum_{j=1}^{n} \ln x_j \quad \text{s.t.} \quad Ax = b,$$

where $\mu \geq 0$ is a barrier parameter. The Lagrangian associated with this problem has the form:

$$L(x,y,\mu) = c^T x - y^T(Ax-b) - \mu \sum_{j=1}^{n} \ln x_j,$$

and the conditions for a stationary point read as

$$\begin{array}{rcrcl} \nabla_x L(x,y,\mu) & = & c - A^T y - \mu X^{-1} e & = & 0 \\ \nabla_y L(x,y,\mu) & = & Ax - b & = & 0, \end{array}$$

where $X^{-1} = \mathrm{diag}\{x_1^{-1}, x_2^{-1}, \dots, x_n^{-1}\}$. After defining $s = \mu X^{-1} e$ (that is $XSe = \mu e$), and $e = (1,1,\dots,1)^T$, the first order optimality conditions (for the barrier problem) become:

$$\begin{array}{rcl} Ax & = & b, \\ A^T y + s & = & c, \\ XSe & = & \mu e \\ (x,s) & \geq & 0. \end{array} \tag{1}$$

Interior point algorithms for linear programming [18] apply Newton method to solve this system of nonlinear equations and gradually reduce the barrier parameter μ to guarantee convergence to the optimal solution of the original problem. The Newton direction is obtained by solving the system of linear equations:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_p \\ \xi_d \\ \xi_\mu \end{bmatrix}, \tag{2}$$

where

$$\xi_p = b - Ax, \qquad \xi_d = c - A^T y - s, \qquad \xi_\mu = \mu e - XSe.$$

By elimination of

$$\Delta s = X^{-1}(\xi_\mu - S\Delta x) = -X^{-1} S \Delta x + X^{-1}\xi_\mu,$$

from the second equation we get the symmetric indefinite augmented system of linear equations

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_d - X^{-1}\xi_\mu \\ \xi_p \end{bmatrix}. \tag{3}$$

where $\Theta = XS^{-1}$ is a diagonal scaling matrix [2].

We will focus in particular on the case where the augmented system matrix in (3) displays a block-sparsity pattern (possibly after reordering). A standard tool when analysing eliminations in block matrices is the notion of *Schur complement* which we recall here, since it will be used in the rest of the paper.

We consider a symmetric matrix H given in block form

$$H = \begin{bmatrix} K & M^T \\ M & C \end{bmatrix}.$$

If the leading block K is nonsingular we can use it as a pivot in a block-elimination leading to the block-decomposition

$$H = \begin{bmatrix} I & \\ MK^{-1} & I \end{bmatrix} \begin{bmatrix} K & \\ & S \end{bmatrix} \begin{bmatrix} I & K^{-1}M^T \\ & I \end{bmatrix}$$

of H, where $S = C - MK^{-1}M^T$ is called the *Schur complement.* Computing factorizations $K = L_K L_K^T, S = L_S L_S^T$ of K and S thus gives the block-factorization of H as

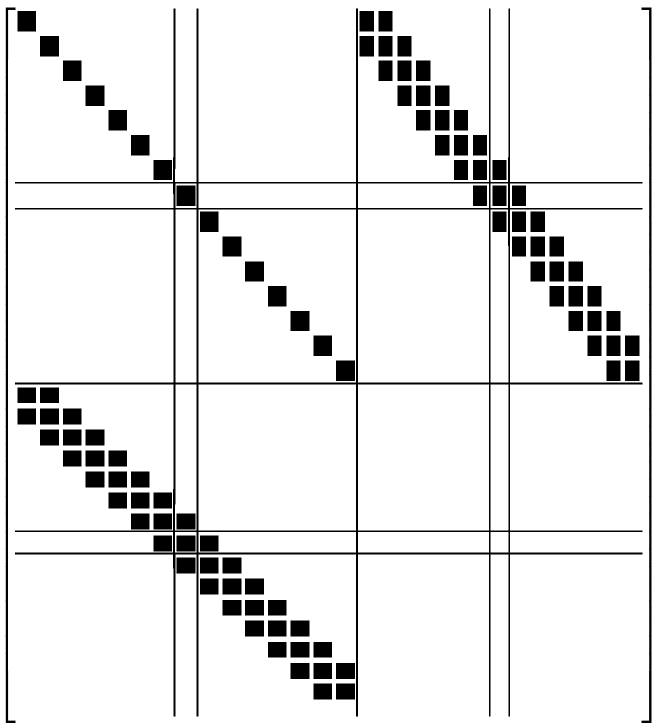
$$H = \begin{bmatrix} L_K & \\ ML_K^{-T} & L_S \end{bmatrix} \begin{bmatrix} L_K^T & L_K^{-1}M^T \\ & L_S^T \end{bmatrix}.$$

In the following two sections we discuss particular forms of augmented system matrices obtained for problems which model *dynamics* and *uncertainty.*

3.1 Dynamics

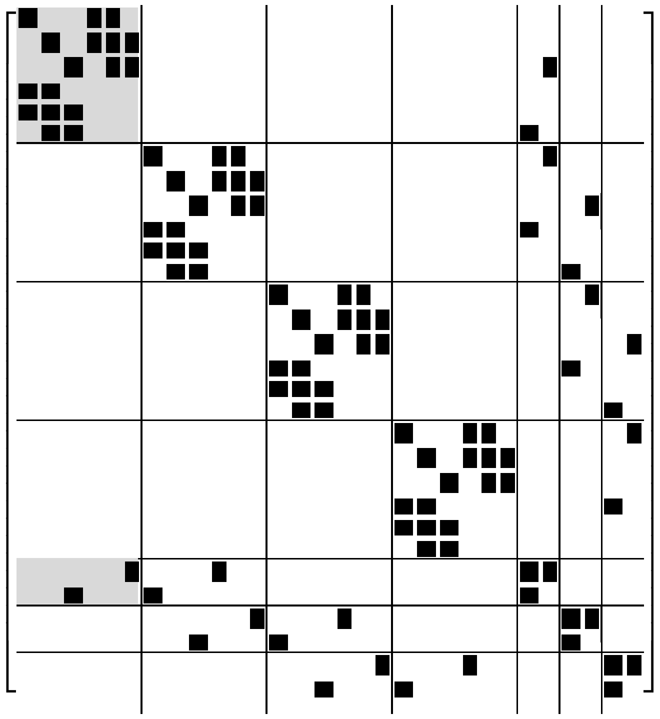
We provide an example of the augmented system (3) corresponding to dynamic structure in the LP constraint matrix. Assume that matrix A has banded block-diagonal sparsity pattern as detailed in the previous section and matrix Θ^{-1} is diagonal (block-diagonal in the figure), leading to an augmented system Matrix H of the form

$H \quad =$

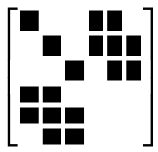
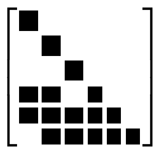
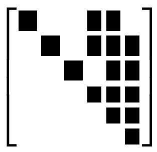


The dynamic structure in this problem can be broken by reordering the rows and columns of the matrix. If we number rows and columns of H as $\{1, 2, \ldots, 30\}$ and the rearrange them in the order $\{1, 2, 3, 16, 17, 18;\ 5, 6, 7, 20, 21, 22;\ 9, 10, 11, 24, 25, 26;\ 13, 14, 15, 28, 29, 30;\ 4, 19;\ 8, 23;\ 12, 27\}$

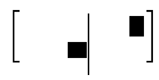
we obtain the following

$PHP^T =$ 

Consider a diagonal block in a form of a (smaller) augmented system, for example the one corresponding to the leading 6×6 diagonal block in PHP^T and its generalized symmetric LDL^T factorization. In the figure below we display the 6×6 block and its triangular factor L

$(PHP^T)_{11} =$  , and $L_{11} =$  , $L_{11}^T =$ 

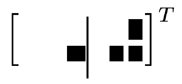
The elimination of this 6×6 block creates a Schur complement contribution to the bottom lower block corresponding to the original row and column blocks $\{4, 19;\ 8, 23;\ 12, 27\}$. We will look closer at the block-sparsity pattern contributed to the leading 2×2 block in it, namely to the Schur complement part corresponding to rows and columns $\{4, 19\}$. Let $V \in \mathcal{R}^{2\times 6}$ denote the block-matrix corresponding to rows 4 and 19 and columns 1, 2, 3, 16, 17 and 18. Its block-sparsity pattern has the following form

$V =$ 

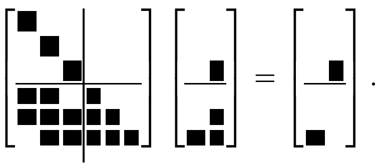
and its Schur complement contribution is $VL_{11}^{-T}L_{11}^{-1}V^T$. This contribution can be computed in several different ways depending on the order of matrix operations or equations to be solved. Although the result is always the same

$$VL_{11}^{-T}L_{11}^{-1}V^T = ((VL_{11}^{-T})L_{11}^{-1})V^T = (VL_{11}^{-T})(L_{11}^{-1}V^T) = V(L_{11}^{-T}(L_{11}^{-1}V^T)),$$

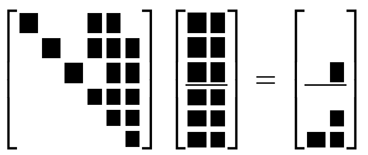
the cost of performing all arithmetic operations may differ significantly. First we note that the first and the last possibility are identical apart from a transposition. Observe that $U^T = L_{11}^{-1}V^T$ has the following block-sparsity pattern

$U^T = L_{11}^{-1}V^T =$  ^{T}.

This can easily be obtained after analysing the block-sparsity pattern in equation $L_{11}U^T = V^T$:

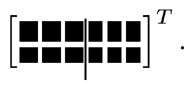


Our following options are to compute the Schur complement contribution $VL_{11}^{-T}L_{11}^{-1}V^T$ as $VL_{11}^{-T}U^T$ or as UU^T. Note that the block-sparsity pattern of $Z^T = L_{11}^{-T}U^T = L_{11}^{-T}L_{11}^{-1}V^T$ is computed by analysing the equation $L_{11}^TZ^T = U^T$:



hence the block-sparsity pattern of Z^T is completely dense:

$$Z^T = L_{11}^{-T}L_{11}^{-1}V^T =$$



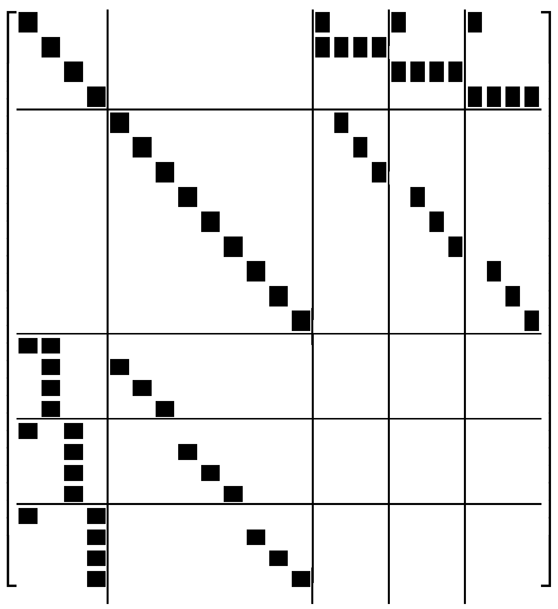
Therefore it is not efficient to form the matrix $L_{11}^{-T}L_{11}^{-1}V^T$; it is preferable to compute U and then form the Schur complement contribution as UU^T for example by outer products of the sparse column vectors of U.

Similar operations have to be performed when other blocks in the 6×6 Schur complement are computed. It is advisable to analyse each of such operations and organise the computations in a form which guarantees maximum efficiency. These issues have been addressed with great care in the implementation of OOPS [14, 12].

3.2 Uncertainty

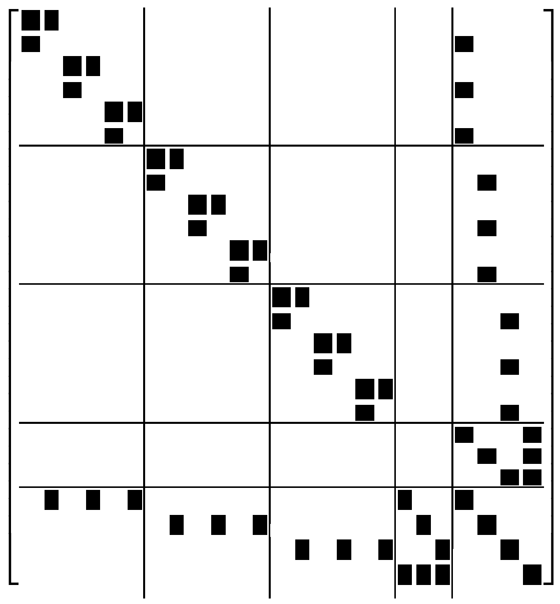
As a second example we consider a block-structure originating from the modelling of uncertainty. The following matrix structures correspond to a three stage stochastic program with 3 second stage and 9 third stage realisations. Matrix A has a block-tree sparsity pattern and matrix Q is diagonal (block-diagonal in the figure).

$H =$



If we number rows and columns of H as $\{1, 2, \ldots, 25\}$ and rearrange them in the order $\{5, 15, 6, 16, 7, 17;\ 8, 19, 9, 20, 10, 21;\ 11, 23, 12, 24, 13, 25;\ 14, 18, 22;\ 2, 3, 4;\ 1\}$ we arrive at

$PHP^T =$



Which as before achieves to break the links between different parts of the model, by delaying them into the right and bottom block-borders. The implementation in the linear algebra kernel of OOPS [14, 12] does not actually physically reorder rows, but rather (equivalently) use the tree-sparse structure of the stochastic programming problem to guide the order of the elimination process.

The exploitation of special structures in the linear algebra operations of interior point method offers a number of advantages:

- it improves the overall efficiency,

- it reduces memory needs (through the use of implicit inverse representation),
- it allows for a straightforward parallelisation.

We demonstrate these advantages when OOPS is applied to a class of real-world distribution planning problems.

4 Formulations of Distribution Problem and Their Structures

In this paper we are concerned with the optimization of a general distribution planning problem. This section gives a general description of possible problem formulations. Note that these are kept fairly general to demonstrate the adaptability of our approach. We discuss a generic distribution planning model and skip details which are proprietary and which the Company has not agreed to be revealed.

4.1 Deterministic Formulation

We are concerned with the distribution of a (storable) utility (gas, oil, electricity) over a network. The network is described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and arcs of the graph respectively. Let $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{E}|}$ be the node-arc incidence matrix of the network.

The network is considered over a sequence of time periods $t \in \mathcal{T} = \{1, \ldots, |\mathcal{T}|\}$ (typically one per day over one or several years). The time periods are partitioned into *seasons* $s \in \mathcal{S}$, where each season consists of the time periods $t \in S(s)$. The model is cyclic, so that time period $t = 1$ is identified with $t = |\mathcal{T}|$.

Different time periods are linked in the following manner:

- Some arcs in the network exhibit a time lag: in-flow into an arc will need several time periods before it appears as the out-flow of the same arc. In terms of the problem formulation, these are captured by the delay matrices $B^{(-\tau)}$ that contain those -1 entries of A corresponding to the sources of arcs with a delay of τ time periods. As pointed out earlier, the problem can always be modelled as one with a maximum delay of $\tau = 1$.
- At certain nodes in the network there are storage facilities that allow to siphon off some of the flow through this node and re-inject it at a later (variable) time period. Use of this storage facility will usually incur a loss of the utility.

Storage can be easily modelled by the above matrices A, $B^{(-\tau)}$, where the loss of utility is accounted for by a weight $\gamma \in (-1, 0)$ in position of the target entry.

The network has to satisfy deterministic daily demands d_t. The utility can be obtained from a set of n_s sources in the network. Let $Q_s \in \mathbb{R}^{n_s \times |\mathcal{V}|}$ be the matrix that projects the space of all nodes onto the space of sources and $\tilde{\phi}_t \in \mathbb{R}^{n_s}$ be the value of the sources at time period t, then $Q_s^T \tilde{\phi}_t$ is the current injection into the network.

For all arcs a record of maximal seasonal use $\bar{x}_s$ is kept. Pricing is done both by actual daily use of arcs as well as by maximal seasonal use. The injection $\tilde{\phi}_t$ is decomposed into a fixed component ϕ_0 plus a daily varying component ϕ_t. Typically, the price p_0 of the utility obtained through the fixed component is lower than the price p_t of the daily varying component.

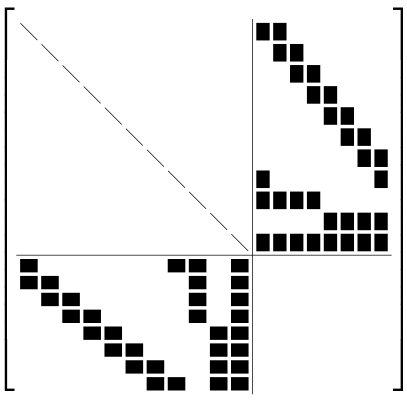
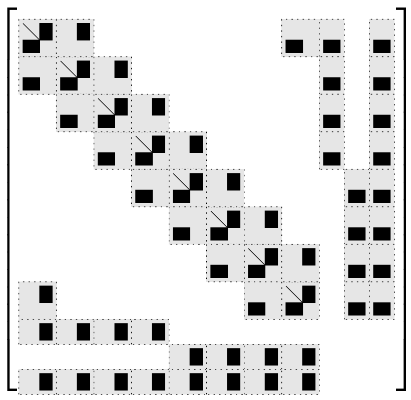
The complete model description is as follows

$$\begin{array}{llcll}
\min & \sum\limits_{t\in\mathcal{T}} (c_t^T x_t + p_t^T \phi_t) + \sum\limits_{s\in\mathcal{S}} c_s^T \bar{x}_s + p_0^T \phi_0 & & & \\
\text{s.t.} & A x_t + \sum\limits_{\tau=1}^{\bar{\tau}} B^{(-\tau)} x_{t-\tau} + Q_s^T \phi_0 + Q_s^T \phi_t & = & d_t & t \in \mathcal{T} \\
 & x_t & \leq & \bar{x}_s & t \in S(s), s \in \mathcal{S}
\end{array} \tag{4}$$

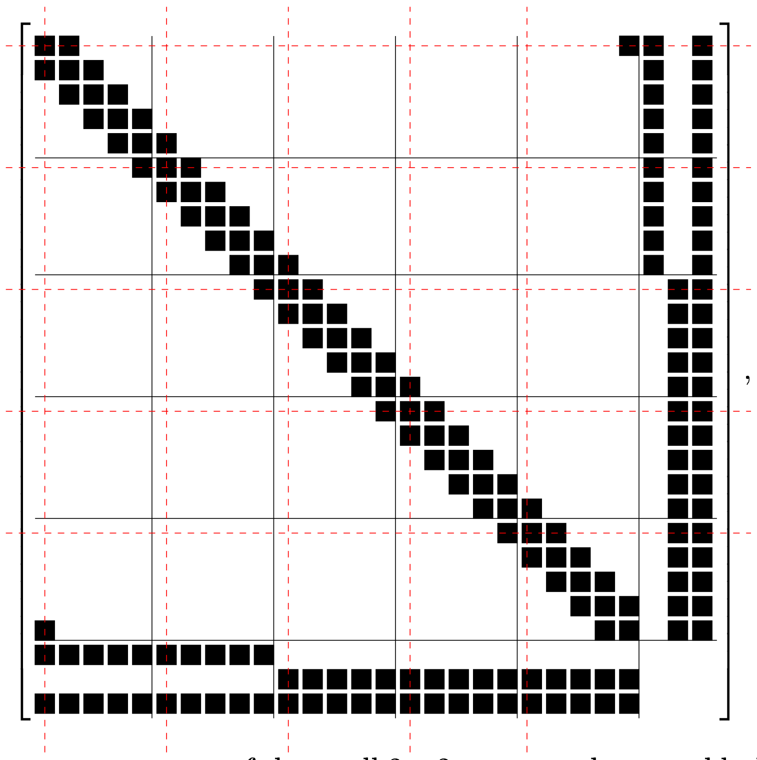
If we concatenate the variables in the order $(x_1, \phi_1, \ldots, x_{|\mathcal{T}|}\phi_{|\mathcal{T}|}, \bar{x}_1, \ldots, \bar{x}_{|\mathcal{S}|}, \phi_0)$, the structure of the constraint matrix (for lag $\bar{\tau} = 1$) is as follows

$$\left[\begin{array}{ccccccccccccc}
A & Q^T & & & & & & & B & & & & Q^T \\
I & & & & & & & & & & -I & & \\
B & & A & Q^T & & & & & & & & & Q^T \\
 & & I & & & & & & & & -I & & \\
 & & B & & \ddots & \ddots & & & & & & & \vdots \\
 & & & & \ddots & & & & & & & & \\
 & & & & \ddots & & A & Q^T & & & & & Q^T \\
 & & & & & & I & & & & & -I & \\
 & & & & & & B & & A & Q^T & & & Q^T \\
 & & & & & & & & I & & & -I &
\end{array}\right]. \tag{5}$$

This is a cyclic dynamic structure with a "dense" column border block. The corresponding augmented system matrix H can be reordered into structured form PHP^T as outlined below (the 19 rows and columns are in the order $\{1, 12, 2, 13, 3, 14, 4, 15, 5, 16, 6, 17, 7, 18, 8, 19;\ 9, 10, 11\}$)

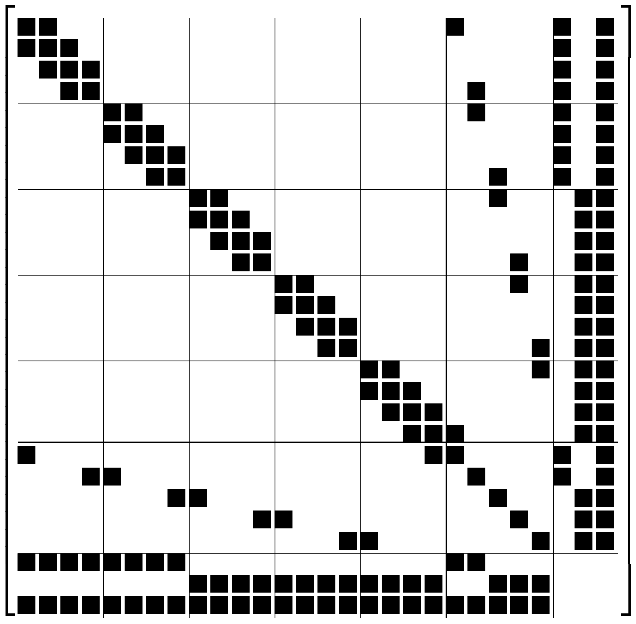
$H =$  $, PHP^T =$ 

which is again of cyclic bordered structure. As outlined in Section 3.1, the dynamic structure in the augmented system can be broken by further reordering the rows and columns. Consider for exemplifying purposes a larger augmented system of the same structure as PHP^T:



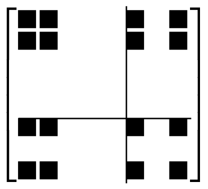
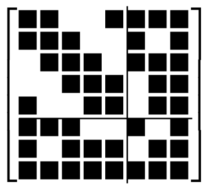
,

where each box now represents one of the small 2×2 augmented system blocks in PHP^T. The cyclic nature of the main block can be broken by ordering the marked rows and columns (i.e. numbers $\{1, 6, 11, 16, 21\}$) to the bottom and right to obtain



. (6)

The Schur complement part now consists of original rows and columns $\{1, 6, 11, 16, 21; 26, 27, 28\}$. The elimination of the leading 4×4 diagonal block in the manner exemplifyed in Section 3.1 will lead to a Schur complement contribution of the form S_1, so that the elimination of all diagonal blocks gives a Schur complement of form $C - \sum S_i$:

$S_1 =$  , $C - \sum S_i =$  (7)

which still exhibits significant block sparsity.

Traditionally the Schur complement technique is used to delay pivoting on dense rows or on linking rows which have entries in every column-block. In either case the Schur complement will be a dense matrix and is usually treated as such. As its size is typically small, there is no performance degradation in using dense linear algebra. Using the Schur complement technique to break dynamic structures however leads to a different type of Schur complement matrix. The delayed links are very sparse but there might be many of them, leading to a large but sparse Schur complement matrix. Usually a Schur complement matrix of this size would not be considered feasible, but savings are to be gained by using sparse linear algebra for the Schur complement calculations.

4.2 Stochastic Programming Formulation

Some parameters in the utility distribution problem are random. Typical examples are demands and prices that vary with (unknown) changes in weather and economics. To account for this case we can reformulate problem (4) as a stochastic programming problem. For this assume that the prices c_t, p_t and the demands d_t depend on a random variable ξ with a known underlying distribution. The prices c_s for the maximal seasonal use of a link (or storage) however are fixed. In the stochastic programming formulation the aim is to solve the stochastic version of (4)

$$\begin{array}{lrcll}
\min & \multicolumn{4}{l}{\mathbb{E}_\xi \left(\sum\limits_{t\in\mathcal{T}} (c_t(\xi)^T x_t(\xi) + p_t(\xi)^T \phi_t(\xi)) + p_0(\xi)^T \phi_0(\xi) \right) + \sum\limits_{s\in\mathcal{S}} c_s^T \bar{x}_s} \\
\text{s.t.} & Ax_t(\xi) + \sum\limits_{\tau=1}^{\bar{\mathcal{T}}} B^{(-\tau)} x_{t-\tau}(\xi) + Q_s^T \phi_0(\xi) + Q_s^T \phi_t(\xi) & = & d_t(\xi) & t \in \mathcal{T} \\
& x_t(\xi) & \le & \bar{x}_s & t \in S(s), s \in \mathcal{S},
\end{array} \tag{8}$$

where x_t, ϕ_t and ϕ_0 are recourse variables, that is their values can be determined after the value of ξ is known. The maximum usage of the links $\bar{x}_s$ on the other hand has to be decided in advance.

Assume that the distribution of ξ is discrete (or otherwise is approximated by a discrete distribution), that is ξ attains values ξ_i, $i \in \mathcal{I}$ with probabilities $\pi_i : \pi_i > 0, \sum_{i\in\mathcal{I}} \pi_i = 1$ and set $c_t^i = c_t(\xi_i), p_0^i = p_0(\xi_i), p_t^i = p_t(\xi_i), d_t^i = d_t(\xi_i)$. Then the stochastic programming formulation (8) can be written as the deterministic equivalent problem

$$\begin{array}{lrcll}
\min & \multicolumn{4}{l}{\sum\limits_i \pi_i \left(\sum\limits_{t\in\mathcal{T}} ({c_t^i}^T x_t^i + {p_t^i}^T \phi_t^i) + {p_0^i}^T \phi_0^i \right) + \sum\limits_{s\in\mathcal{S}} c_s^T \bar{x}_s} \\
\text{s.t.} & Ax_t^i + \sum\limits_{\tau=1}^{\bar{\mathcal{T}}} B^{(-\tau)} x_{t-\tau}^i + Q_s^T \phi_0^i + Q_s^T \phi_t^i & = & d_t^i & t \in \mathcal{T}, i \in \mathcal{I} \\
& x_t^i & \le & \bar{x}_s & t \in S(s), s \in \mathcal{S}, i \in \mathcal{I}.
\end{array} \tag{9}$$

If we write the constraint matrix in (5) as blocks $[XR]$, where the X part corresponds to variables $(x_1, \phi_1, \ldots, x_{|\mathcal{T}|}\phi_{|\mathcal{T}|}, \phi_0)$ and the R part to $(\bar{x}_1, \ldots, \bar{x}_{|\mathcal{S}|})$, then the constraint matrix of the stochastic problem has the structure

$$\begin{bmatrix} R & X & & & \\ R & & X & & \\ \vdots & & & \ddots & \\ R & & & & X \end{bmatrix}, \tag{10}$$

described in Section 3.2. The X matrices, even after removing the $\bar{x}$ columns, display the bordered dynamic structure described in Section 3.1. We can combine the reordering techniques described for both structures to arrive at a bordered block diagonal matrix with a sparse Schur complement similar to that described in (6),(7).

5 Conveying Structure to the Solver

The problem formulations presented in the previous section all display a nested block-structured pattern. Exploiting this block-sparsity in our solver OOPS turned out to be instrumental in being able to solve the larger instances of these problem.

So far conveying the problems and their structure to structure exploiting solvers (such as OOPS, but also various decomposition approaches) has required carefully handcrafted routines in a language such as C or Fortran to fill the required data-structures with the correct parts of the model. Clearly this approach is cumbersome and error prone especially for large models, while lacking flexibility for later changes to the model.

The Structure Exploiting Tool (SET: [10]) shows a possible solution to this problem which we have largely followed: The usual output of an algebraic modelling language is a problem description via sparse data structures together with a *dictionary*: a mapping of row and column indices to the names for constraints and variables used in the model. Our Structure Passing Interface (SPI) uses a *structure definition file* that describes the block-structure of the model in a nested text format. Basic sparse blocks are described by matching expressions for relevant row and column names. Special constructs allow the combination of basic blocks into structured matrices and finally the complete model. SPI analyses the structure definition file by comparing it with the dictionary and builds a skeleton problem description (consisting of the matrix block structure and a list of row and column indices for each block, but no matrix entries). This is then used by the solver to set up its data structures. The solver now has an internal representation of the structure of the problem, but as yet without the actual matrix entries. These are passed in a second step, via the use of a call-back-function supplied to the solver as part of the description of the basic blocks. The second phase essentially only involves communication between the solver and the modelling language, bypassing SPI, much in the same way as the traditional passing of problem data from modelling language to solver.

An important advantage of the two-phase approach is that the second phase is able to take advantage of the structure in the problem. When a parallel solver is used the setting up of the solver-internal data structures involves dividing the problem among processors. In the second phase each processor requests only request the relevant parts of the problem from the modelling language, resulting in important speed-up during the problem generation phase.

Our implementation of this approach is modular in its nature, it is flexible both about which modelling language to use as well as which solver to link to. Currently we have interfaces to MPS and AMPL. So far the only solver linked into this approach is OOPS, although interfaces to other structure exploiting solvers could be produced without significant overhead.

However, it should be kept in mind that this approach is not fully satisfactory: the structure of the model is first lost in the generation process and an additional effort is needed to restore it afterwards. Another disadvantage of this approach is the inevitable loss of efficiency and potential parallelisability in the processing of the model by the modelling language. Indeed, as noted in [10], the generation of a problem by an algebraic modelling language can sometimes take longer than the subsequent solution phase. An example of such inefficiency was given in [13], where a small stochastic linear program with 17,741 constraints, 35,484 variables and 92,800 non-zero elements in the constraint matrix was generated by GAMS in 1630 seconds; the same model was generated by a specialised generator written in the `C` programming language in just 2 seconds. Generally speaking, modelling nontrivial problems with current algebraic modelling languages integrated with general optimization software faces a number of difficulties: (i) the structure of the problem is lost, (ii) the generation process is inefficient. To address these issues in their entirety would require the development of a dedicated modelling language.

Several variants of the distribution planning problem have been considered in this study. We discuss the computational experience in the following section.

6 Numerical Results

We have tested the above methodology on the set of distribution planning problems available to us. In total there are five different problems. Three of these are deterministic problems: `D1Yl` models one year and takes time lags into account, `D1Yn` is identical to this apart from not having time lags and `D7Yn` is a 7 year problem without time lags. The other two correspond to the stochastic model: `S7` is a small network with 36 scenarios, whereas `S321` is a stochastic version of `D1Yn` modelled with 7 scenarios. The problem characteristics are summarized in Table 1.

name	variables	constraints	planning period	nodes	arcs	scenarios
D1Yl	850,324	484,355	365d	321	763	det
D1Yn	850,324	484,355	365d	321	763	det
D7Yn	5,880,190	3,390,485	2555d	321	763	det
S7	459,980	341,640	365d	7	10	36
S321	4,939,945	3,390,485	365d	321	763	7

Table 1: Problem characteristics for the five test problems

Here the column *planning period* gives the length of the planning period of the model in days (i.e. 1 or 7 years), *nodes* and *arcs* give the dimensions of underlying distribution network, and the column *scenarios* gives the number of probability scenarios considered (`det` for a deterministic problem). As a base for further comparisons we will state the performance of CPLEX 9.1 barrier code on these problems in Table 2. We report the values used to judge the suitability of our approach, namely apart from the time and IPM iterations to reach the convergence tolerance (10^{-4} apart from `S7` which CPLEX could only solve to 10^{-3}), the peak memory

name	time(s)	memory	IPM iters	nz in factor
D1Yl	1448	917MB	60 (1e-4)	62,023,149
D1Yn	894	808MB	49 (1e-4)	48,512,375
D7Yn	-	OoM	-	594,452,075
S7	161	262MB	162(1e-3)	2,601,576
S321	-	OoM	-	530,000,000

Table 2: CPLEX 9.1 barrier code solution statistics (OoM = Out of Memory)

requirement and the total numbers of nonzeros in the inverse representation of the matrix to be inverted at every iteration (Cholesky factor of AA^T for CPLEX). The number of nonzeros can serve as an indication of necessary peak memory and solution time per iteration independent of implementational issues.

Our solution approach is based on breaking up the dynamic structure of the problem by delaying column elimination wherever necessary. This is also done for the dense columns corresponding to the first stage decision variables in the stochastic formulations. We have experimented with a different number of cuts of the dynamic part of the problem in the pursuit for a reordering that minimizes both solution time and memory requirement. Table 3 summarizes the characteristics of each of these cutting schemes.

name	div	vars/div	n(S)	nz(S)	nz(L_S)	nz(A)	nz(L_A)	tt nz
D1Yl	0	237	1119	626K (100%)	626K	3.3M	8.5M	9.1M
	2	237	1593	1,019K (80%)	1,268K	1.6M	3.9M	9.1M
	2a	237	605	183K (100%)	183K	1.814M	9.3M	18.8M
	5	237	2304	1,767K (67%)	1,873K	633K	1,4M	8.9M
D1Yn	0	83	1119	626K (100%)	626K	3,224K	7.4M	8.0M
	2	83	1285	581K (70%)	581K	1,610K	3.5M	7.6M
	5	83	1534	776K (66%)	778K	626K	1.3M	7.3M
D7Yn	7	83	8328	5.6M (16.3%)	5.7M	3.3M	7.4M	57.5M
	7a	83	1412	OoM		3.7M	17.1M	>120M
	14	83	8909	4.1M (10.3%)	4.3M	1.6M	3.6M	54.7M
	14a	83	1993	OoM		1.8M	8.9M	>125M
	35	83	10652	5.4M (9.5%)	6.3M	664K	1.42M	56.0M
S7	36	0	8	28 (100%)	28	29K	43K	1.5M
S321	7	0	577	167K (100%)	167K	1.56M	7.3M	51.5M
	14	214	2075	1.2M (55%)	1.2M	772K	3.3M	46.7M
	35	93	3818	2.54M (35%)	2.64M	308K	1.22M	45.3M

Table 3: Problem characteristics for different cutting schemes

Here *div* gives the number of cuts in the dynamic structure and *vars/div* the number of rows and columns that are moved to the Schur complement for every cut in the dynamic structure; additional Schur complement variables represent the dense columns corresponding to the $\bar{x}, \phi_0$ variables in (4). $n(S), nz(S), nz(L_S)$ give the dimension, number of nonzeros and nonzeros in the Cholesky factors of the Schur complement, respectively. For $nz(S)$ we also report the density of the Schur complement (per cent in brackets). $nz(A), nz(L_A)$ give corresponding values for the resulting diagonal blocks: the number of nonzero elements in the block and its Cholesky factor, respectively. Note that due to the cyclic nature of the dynamic structure the number of cuts equals to the number of resulting diagonal blocks. In our default setting all dense columns of the dynamic blocks are included in the Schur complement. When dividing `D1Y` into 2 blocks, or `D7Y` into 7 or 14 blocks, however there is no link between the seasonal maxima $\bar{x}_s$, so these columns could be assigned to the corresponding diagonal block, resulting in a smaller Schur complement. These reorderings have been tried, corresponding to the `D1Yl-2a`, `D7Yn-7a/14a` rows of Table 3.

As can be seen for all problems we are able to improve dramatically on the total size of the factors and hence on the memory requirements for the method. We are buying this advantage at the cost of an increase in computational effort for large Schur complements. However, it can be seen that for the problems with a large Schur complement, this is fairly sparse and does not fill in considerably during factorization. The `a` versions of the reordering do indeed lead to a smaller Schur complement, however the dense columns lead to much more difficult to factor diagonal blocks, resulting in a much higher number of total nonzeros. For the larger problem `D7Yn` these versions ran out of memory in the factorization phase.

In Table 4 we present our results on the various formulations of the problem

name	div	time	memory	IPM iters	time/iter
D1Yl	0	120m	377MB	46(1e-6)	2.6m
	2	119m	378MB	48(1e-4)	2.5m
	2a	108m	1396MB	52(1e-4)	2.1m
	5	136m	388MB	56(1e-4)	2.5m
D1Yn	0	151m	363MB	70(1e-4)	2.1m
	2	142m	362MB	67(1e-4)	2.1m
	5	156m	362MB	69(1e-4)	2.2m
D7Yn	7	-	OoM	-	-
	14	-	OoM	-	-
	35	-	OoM	-	-
S7	36	10.8m/11.6m	184MB	67 (1e-3), 72(1e-4)	9.6s
S321	7	1223m	2.25GB	162 (1e-4)	7.5m
	14	1488m	2.21GB	166 (1e-4)	9.0m
	35	1318m	2.27GB	163 (1e-4)	8.0m

Table 4: OOPS solution statistics

In all cases memory requirement is less than for CPLEX (apart from `D1Yl-2a`). Problem `S321` can be solved by OOPS, although CPLEX ran out of memory. Unfortunately we are buying this advantage at the cost of an increase in computational effort. Even with this more efficient memory use problem `D7Yn` is not solvable on one processor. The final Table 5 gives solution times for this problem on 2 processors.

We end this section by indicating the parallel peformance that is achievable with OOPS. In Table 6 we compare the time spent on the first 10 iterations of OOPS. The columns of the table give the serial time (1p) in seconds, the parallel time (np), the number of processors used in the parallel run (n) and the resulting parallel efficiency (pe).

name	div	time	memory	IPM iters	time/iter
D7Yn	7	921m	3.5GB total	77 (1e-4)	11.96m
	14	1161m	3.4GB total	79 (1e-4)	14.7m
	35	1260m	3.4GB total	84 (1e-4)	15.00m

Table 5: OOPS solution statistics: 2 processors

name	div	time (1p)	time (np)	n	pe
D1Yl	2	1236	721	2	85.7%
D1Yn	2	1133	690	2	82.1%
D1Yn	5	1161	330	5	70.4%
S7	36	72	37	2	97.3%
S7	36	72	31	3	77.4%
S7	36	72	20	6	51.4%
S123	7	6322	1616	7	55.8%
S123	14	7111	1474	7	68.9%

Table 6: Parallel performance

As can be seen OOPS is displaying good parallel efficiency. The main reasons for not achieving perfect speed-up are the non-trivial work of factorizing the Schur complement (which is not paralellised) as well as load-balancing issues due to the problems not decomposing into equal sized parts.

7 Conclusions

We have presented a study of modelling and solving a challenging utility distribution planning problem. Real life instances of the problem defy the industry standard solver CPLEX 9.1. A dedicated modelling environment together with the structure exploiting interior point solver OOPS enabled us to exploit the structure present in the problem. Important memory savings could be achieved and these have resulted in the ability to solve at least one of the challenging real life problems. Breaking the dynamics structure present in the problem resulted in large but sparse Schur complement matrices. While using sparse linear algebra in the Schur complement calculations made the memory requirements for Schur complements with up to 10,000 columns feasible, there is a performance degradation associated with it.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*, Prentice-Hall, New York, 1993.

[2] E. D. Andersen, J. Gondzio, C. Mészáros, and X. Xu, *Implementation of interior point methods for large scale linear programming*, in Interior Point Methods in Mathematical Programming, T. Terlaky, ed., Kluwer Academic Publishers, 1996, pp. 189–252.

[3] J. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, 2001.

[4] J. R. Birge, *Decomposition and partitioning methods for multistage stochastic linear programs*, Operations Research, 33 (1985), pp. 989–1007.

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

[6] J. Bisschop and R. Entriken, *AIMMS: The modeling system.* Paragon Decision Technology, 1993.

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

[8] R. Fourer, *Staircase matrices and systems*, SIAM Review, 26 (1983), pp. 1–70.

[9] R. Fourer, D. Gay, and B. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, San Francisco, California, 1993.

[10] E. Fragnière, J. Gondzio, R. Sarkissian, and J.-P. Vial, *Structure exploiting tool in algebraic modeling languages*, Management Science, 46 (2000), pp. 1145–1158.

[11] H. Gassmann, *Mslip: A computer code for the multistage stochastic linear programming problems*, Mathematical Programming, 47 (1990), pp. 407–423.

[12] J. Gondzio and A. Grothey, *Exploiting structure in parallel implementation of interior point methods for optimization*, Technical Report MS-04-004, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK, December 2004.

[13] J. Gondzio and R. Kouwenberg, *High performance computing for asset liability management*, Operations Research, 49 (2001), pp. 879–891.

[14] J. Gondzio and R. Sarkissian, *Parallel interior point solver for structured linear programs*, Mathematical Programming, 96 (2003), pp. 561–584.

[15] F. Hillier and G. Lieberman, *Introduction to Operations Research*, McGraw Hill, Singapore, 7th ed., 2001.

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

[17] W. Winston, *Operations Research Applications and Algorithms*, Duxbury, Belmont, 3rd ed., 1994.

[18] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.