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

Annals of Operations Research 90(1999)161–184

EVPI-based importance sampling solution procedures for multistage stochastic linear programmes on parallel MIMD architectures

M.A.H. Dempster and R.T. Thompson
Judge Institute of Management Studies, University of Cambridge, Trumpington Street, Cambridge, UK
E-mail: {mahd2;rt208}@cam.ac.uk

Multistage stochastic linear programming has many practical applications for problems whose current decisions have to be made under future uncertainty. There are a variety of methods for solving the deterministic equivalent forms of these dynamic problems, including the simplex and interior-point methods and nested Benders decomposition, which decomposes the original problem into a set of smaller linear programming problems and has recently been shown to be superior to the alternatives for large problems. The Benders subproblems can be visualised as being attached to the nodes of a tree which is formed from the realisations of the random data process determining the uncertainty in the problem. This paper describes a parallel implementation of the nested Benders algorithm which employs a farming technique to parallelize nodal subproblem solutions. Differing structures of the test problems cause differing levels of speed-up on a variety of multicomputing platforms: problems with few variables and constraints per node do not gain from this parallelisation. We therefore employ stage aggregation to such problems to improve their parallel solution efficiency by increasing the size of the nodes and therefore the time spent calculating relative to the time spent communicating between processors. A parallel version of a sequential importance sampling solution algorithm based on local expected value of perfect information (EVPI) is developed which is applicable to extremely large multistage stochastic linear programmes which either have too many data paths to solve directly or a continuous distribution of possible realisations. It utilises the parallel nested Benders algorithm and a parallel version of an algorithm designed to calculate the local EVPI values for the nodes of the tree and achieves near linear speed-up.

Keywords: linear programming, dynamic stochastic programming, nested Benders decomposition, parallel algorithms, aggregation, MIMD computers

1. Introduction

The purpose of this paper is to investigate parallel decomposition algorithms for large-scale dynamic stochastic programming problems on two current multicomputers, Fujitsu's AP1000 and IBM's SP2. A farming strategy is used to solve the subproblems

of the decomposition in parallel and efficiency improvements through time stage aggregation of several subproblems are investigated. *Stochastic programming* is the extension to the field of *mathematical programming* obtained by introducing *uncertainty* into the problem data and decision process. Such models are necessary when the effectiveness of current decisions is dependent on future events. Fields in which stochastic programming models arise include economics, financial planning [8,9,20, 24,25,37,44] and telecommunications; many examples can be found in [15,23]. Given an appropriate *probability space* $(\Omega, \mathcal{F}, \mathrm{P})$, where Ω is a sample space with sample points ω, $\mathcal{F}$ is the σ-field of subsets of Ω and P is a probability measure, and taking the (von Neumann–Morgenstern) *expected utility* of the problem objective leads to a general formulation for stochastic programming.

Two main approaches to *static* stochastic programming have been developed: the *chance constrained* and *recourse* formulations. The recourse formulation or *two-stage* stochastic programming problem has a later, second stage that is dependent on the decision vector from the first stage. This model can be extended to the multistage case, when a set of decisions $\{x_1, x_2, \ldots, x_T\}$ are to be made over time up to the *horizon* T. The sequence of random vectors $\omega_1, \omega_2, \ldots, \omega_T$ yield the *discrete time* (*data*) *stochastic process*: $\omega := (\omega_1, \omega_2, \ldots, \omega_T)$ and the notation $\omega^t := (\omega_1, \omega_2, \ldots, \omega_t)$ is used to represent possible histories of this stochastic process up to time t. In the multistage problem, an observation is made and then a decision is taken, followed by another observation and another decision, and so on: $\xi_1 \rightsquigarrow x_1, \xi_2 \rightsquigarrow x_2, \ldots, \xi_{T-1}, \xi_T \rightsquigarrow x_T$. The first observation and decision are usually deterministic in these models. A *filtration* (see [4] for a full definition) can be used to introduce *nonanticipativity* [15] of the decisions into the recourse model.

If the stochastic process is *discrete* state, the expected criterion is a summation, and if all the problem functions are linear (and therefore separable), we obtain the *linear multistage recourse problem* model

$$
\begin{aligned}
\textbf{(LMRP)} \qquad & \underset{x_1}{\text{minimise}} \ \{c_1' x_1 + Q(x_1) : A_{11} x_1 = b_1\}, \\
& \text{where} \quad Q(x_t) := \mathbb{E}_{\omega_t}\left\{ \min_{x_t} c_t' x_t + Q(x_t) : \sum_{\tau=1}^{t} A_{t,\tau} x_\tau = b_t \ \text{a.s.} \right\}, \\
& \text{and} \quad Q(x_{T+1}) \text{ is specified}, \\
& \qquad l_1 \le x \le u_1, \\
& \qquad l_t \le x_t \le u_t \ \text{a.s.}, \quad t = 2, \ldots, T. \qquad (1)
\end{aligned}
$$

Here, all constraints involving random variables hold *almost surely* (a.s), i.e. with probability one. There are several overviews of the stochastic programming literature which contain information on chance constrained methods, approximating schemes and problems with infinite horizons, see [15,16,23,33]. Dupačová [20] has recently surveyed the multistage stochastic programming literature.

The *deterministic equivalent* of problem (1) can be taken by replacing the expectations by probability weighted summations over the finite number of data paths and writing deterministic constraints for each path. The size of the associated constraint matrix increases exponentially with the number of time stages and the number of outcomes for each random vector ω_t. Each of the data paths $\omega := \{\omega_1, \omega_2, \ldots, \omega_T\}$ is referred to as a *scenario*. The large-scale deterministic equivalent form of problem (1) can be solved by the simplex or interior-point method. Alternatively, decomposition methods such as *augmented Lagrangian decomposition* [16,40] or *Benders decomposition* [3,5,28,45] can be employed. Benders decomposition is more suitable for decomposing stochastic linear programmes than Dantzig–Wolfe decomposition [14], and was first developed by Van Slyke and Wets [45] into a cutting plane algorithm for the two-stage problem. This has been extended to the *nested decomposition* method by Birge [5] and Gassmann [28] for linear multistage recourse problems.

Sequential importance sampling methods are required to reduce the potentially very large dimensionality of realistic deterministic equivalent problems, whilst parallel programming allows problems to be solved more quickly or more detailed problems to be treated.

In section 2, nested Benders decomposition is reviewed and the associated algorithm explained. Section 3 gives an outline of the MSLiP nested Benders code, while section 4 describes the parallel version of the code and gives parallel results. Section 5 explains the concept of *expected value of perfect information* (EVPI) [12,16–18] and its calculation, while section 6 contains the EVPI-S sampling procedure [12,17]. The parallelisation of the EVPI algorithms is found in section 7, which includes numerical results, and conclusions are drawn in section 8.

2. Nested Benders decomposition

In the discrete case, the data sample path (scenario) space is partitioned into sets $E_t \in \mathcal{F}_t \subset \mathcal{F}_T := \Omega$, where $\mathcal{F}_t$ forms a filtration over $t = 1, \ldots, T$. The sets E_t contain realizations which are identical up to period t and are distinct at the horizon T, so that the stochastic process can be represented by a tree structure. In figure 1, the Lane tree partition matrix is shown [35], which gives the Lane node labelling $l(\omega^t, t)$ for node ω^t in period t and shows the partitioning E_t of the sets in that stage. There is a 1:1 mapping between the outcomes of the random variables ω^t and the *nodes* of the tree at stage t, for $t = 1, \ldots, T$. The realisation ω^t can thus be used to represent a node of the tree at stage t. The hierarchical structure of the nodes can be described in the same manner as a family, with ancestors, descendants, parents, children, siblings and cousins. The nested Benders method solves the problem in a recursive manner. This can be illustrated by taking a particular node of the decision tree defined by the data path ω^t and its descendants:

$$
\min\{c_t(\omega^t)' x_t(\omega^t) + Q(\omega^t, x_t(\omega^t)) : A_{t,t}(\omega^t) x_t = b_t(\omega^t) - B(\omega^t, x_t(\omega^t))\}
$$

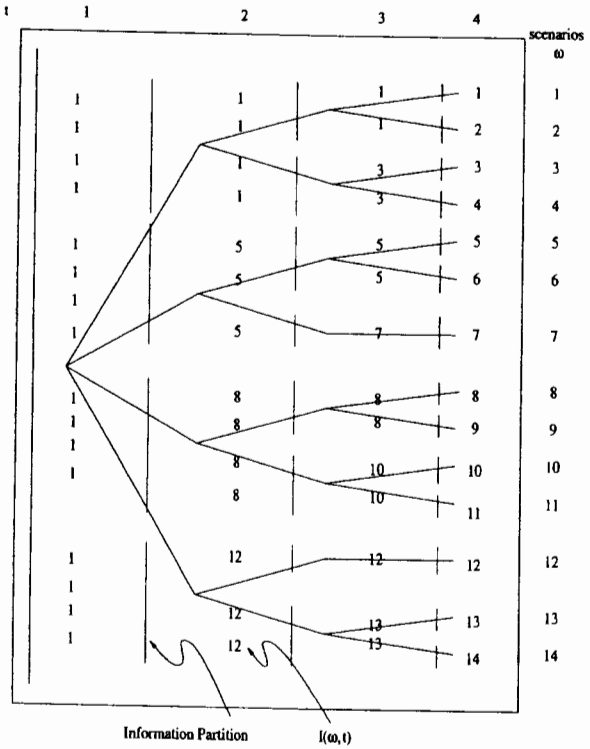

Figure 1. Tree partition matrix.

$$= \min\{c_t(\omega^t)'x_t(\omega^t) + \theta_t(\omega^t) : A_{t,t}(\omega^t)x_t = b_t(\omega^t) - B(\omega^t, x_t(\omega^t))$$
$$\theta_t(\omega^t) \geq Q(\omega^t, x_t(\omega^t))\}, \tag{2}$$

where

$$B(\omega^t, x_t(\omega^t)) := \sum_{\tau=1}^{t-1} A_{t,\tau}(\omega^\tau)x_\tau(\omega^\tau)$$

and

$$Q(\omega^t, x_t(\omega^t))$$

$$:= \text{minimise} \sum_{\omega^{t+1}|\omega^t} p(\omega^{t+1})c_{t+1}(\omega^{t+1})'x_{t+1}(\omega^{t+1})$$

$$\text{subject to } A_{t+1,t+1}(\omega^{t+1})x_{t+1}(\omega^{t+1}) = b_{t+1}(\omega^{t+1}) - B(\omega^{t+1}, x_{t+1}(\omega^{t+1})), \tag{3}$$
$$l_t \leq x_t(\omega^t) \leq u_t, \quad l_{t+1} \leq x_{t+1} \leq u_{t+1}.$$

Both the corresponding primal and dual problems can be decomposed into sets of smaller subproblems, which can be solved using the simplex method. The number of these subproblems is equal to the number of descendants of the node ω^t. The dual problems are all feasible, since the dual decision $y_{t+1}(\omega^{t+1})$ is unconstrained. If the dual problem is unbounded, then the primal problem is infeasible, and a feasibility cut can be placed in the parent node problem. Otherwise, for an optimal solution for (3), either a *single optimality cut* is placed, if weighted sums are taken, or these can be disaggregated to give a set of cuts, known as *multicuts*.

Then, having a collection of dual feasible vectors $(y^i_{t+1}(\omega^{t+1})', \lambda'^j, \mu'^i)$, $i = 1,\ldots, I_t(\omega^t)$, and directions $(y^j_{t+1}(\omega^{t+1})', \lambda'^j, \mu'^j)$, $j = 1,\ldots,J_t(\omega^t)$, we obtain the *relaxed* nodal problem

$$\begin{aligned}
&\underset{x_t(\omega^t)}{\text{minimise}} \ \{c_t(\omega^t)'x_t(\omega^t)+\theta_t(\omega^t)\} \\
&\text{subject to } A_{tt}(\omega^t)x_t(\omega^t) = b_t(\omega^t) - B(\omega^t, x_t(\omega^t)), \\
&\qquad y^j_{t+1}(\omega^{t+1})'A_{t+1,t}(\omega^t)x_t(\omega^t) \geq y^j_{t+1}(\omega^{t+1})'b_{t+1}(\omega^{t+1}) + \lambda'^j l_{t+1} \\
&\qquad\qquad - \mu'^j u_{t+1} - y^j_{t+1}(\omega^{t+1})' \sum_{\tau=1}^{t-1} A_{t+1,\tau}(\omega^\tau)x_\tau(\omega^\tau), \\
&\sum_{\omega^{t+1}|\omega^t} p(\omega^{t+1})y^i_{t+1}(\omega^{t+1})'A_{t+1,t}(\omega^t)x_t(\omega^t) + \theta_t(\omega^t) \geq \sum_{\omega^{t+1}|\omega^t} p(\omega^{t+1})\{y^i_{t+1}(\omega^{t+1})' \\
&\qquad [b_{t+1}(\omega^{t+1}) - \sum_{\tau=1}^{t-1} A_{t+1,\tau}(\omega^\tau)x_\tau(\omega^\tau)] + \lambda'^i l_{t+1} - \mu'^i u_{t+1}\}, \\
&\qquad l_t \leq x_t(\omega^t) \leq u_t, \\
&\qquad i = 1,\ldots, I_t(\omega^t), j = 1,\ldots, J_t(\omega^t).
\end{aligned} \tag{4}$$

The objective $c_t(\omega^t)'x_t(\omega^t) + \theta_t(\omega^t)$ gives a lower bound on the optimal objective value for the nodal problem, while $c_t(\omega^t)'x_t(\omega^t) + Q(\omega^t, x_t(\omega^t))$ gives an upper bound. So, if $\theta_t(\omega^t) \geq Q(\omega^t, x_t(\omega^t))$, the nodal problem is fully solved for the right-hand side supplied to it. The relaxed problem is solved iteratively, producing optimality and feasibility cuts until this condition is satisfied.

The *nested Benders decomposition* algorithm based on the above ideas can be stated as follows (cf. [5,29]):

Step 0. Solve the root node (first-stage problem). At iteration zero, in problem (4), set $\theta_1 := I_1(\omega^1) := J_1(\omega^1) := 0$ and take the first constraint to be $A_{11}x_1 = b_1$. Set $I_t(\omega^t) := J_t(\omega^t) := 0$, $\forall\, t$, ω^t. If problem (4) is infeasible, stop – the original problem is infeasible. If $\theta_1 \geq Q(x_1)$, stop – the optimal solution has been reached. Otherwise, set $t := t + 1$ and let $\tilde{x}_1$ be the current solution vector to the root problem.

Step 1. The formation of the next stage problems can be done by using the decision vectors from ancestors $x_{t-1}(\omega^{t-1})$ to form the right-hand sides of the constraints for t for all the outcomes of ω^t whose parents have an optimal solution, to give the appropriate problem (4). Some nodal problems at this time stage will not be solved due to the infeasibility of their parent or parent's siblings problems. If a nodal problem is found to be infeasible, place a feasibility cut in the parent node problem. This branch of the tree is not explored any further until the parent node problem is re-solved to optimality. There is the choice to move either forward to $t+1$ and go to step 1, or backward to $t-1$ and go to step 2. If $t = T$ or all nodal problems in the current time period are infeasible, then go to step 2.

Step 2. Set $\theta_t(\omega^t) := -\infty$ for all values which have not been previously been set in this time period. If all the descendant node problems at node ω^t have optimal solutions, a single optimality cut may be placed in the node ω^t problem. Otherwise, a feasibility cut has been previously placed. In the case of multicuts, both feasibility and optimality cuts may be placed in the node ω^t problem. There is the choice to move either forward to $t+1$ and go to step 1, or backward to $t-1$ and go to step 2. If $t = T$ or all nodal problems in the current time period are infeasible, then go to step 2. If $t = 1$, go to step 0.

Problems at any stage need only be re-solved if supplied with a new right-hand side or if $\theta_t(\omega^t) < Q(\omega^t, x_t(\omega^t))$, when new cut information is supplied to the problem at ω^t. The *sequencing protocol* determines whether to descend the tree further in step 1 (termed a *forward pass*) or return to the previous stage of the tree in step 2 (termed a *backward pass*). The *fast-forward-fast-back method* [46] suggests continuing in the current direction whenever possible and appears to be the most efficient sequencing protocol [29].

3. The MSLiP code

The MSLiP code has been developed over a number of years through the combined efforts of a variety of programmers [5, 28]. The current version (8.3) is an improvement on the version described in Gassmann's working paper [29]. The MSLiP nested Benders decomposition code (see figure 2) consists of the following modules:

(1) An LP solver (the NORMAL routine). The LP solver found in the code is a modification of the Pfefferkorn–Tomlin LP code [39], which uses the product form of the inverse. The pricing has recently been improved to include steepest edge and random pricing, as well as the default most negative cost. It has no crash-to-feasibility heuristics. Basis, primal, dual and right-hand side vectors are needed from the solution of each nodal problem.

(2) The subproblem manager routine (NDCOM) decides which direction in the tree to move, which is dependent on the sequencing protocol. After a problem is solved it is marked infeasible or optimal. NDCOM will then decide to place a cut in the previous time stage and re-solve or form the right-hand side for the next time stage. Each nodal problem is marked if it needs to be re-solved.

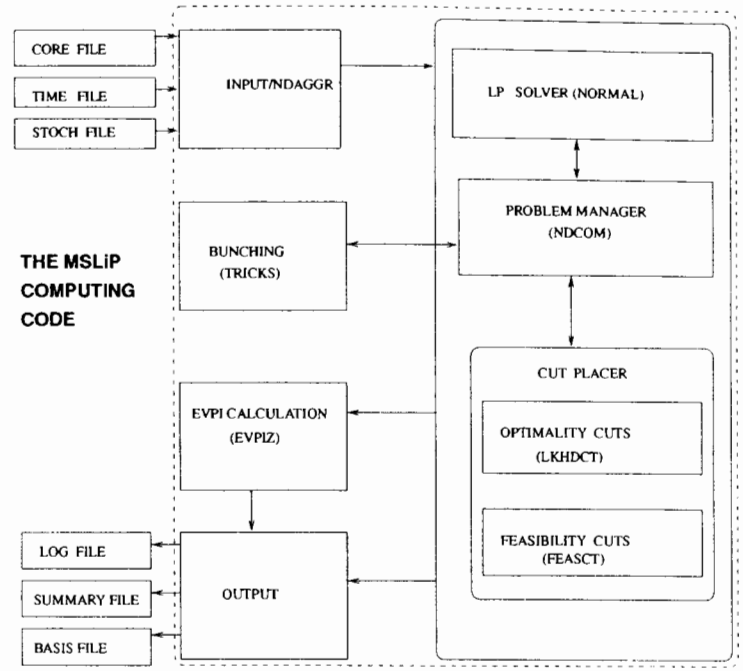

Figure 2. The MSLiP system.

(3) The cut routines form the optimality (LKHDCT) or feasibility (FEASCT) cuts to be placed in the appropriate nodal problem. The optimality cuts depend on whether the user has defined single or multicuts in the specification input file. The default in MSLiP is single cuts for the two-stage problem and multicuts for multistage problems.

(4) The bunching routine (TRICKS) tries to reduce the amount of calculation done in solving a set of LP problems that differ only on their right-hand sides. These problems may share bases and pivots, and may even have the same optimal basis (see [28]).

(5) After the solution has been reached, there is a routine (EVPIZ) for calculating the EVPI at each node of the scenario tree. This indicator is useful in determining how "stochastic" the problem actually is. Calculation of the EVPI process is explained in section 5.

There are two other Benders-based codes which have been developed for solving multistage SLPs: ND-UM [6] and SP/OSL [34]. Both are based on the IBM subroutine library OSL [31] which performs the simplex calculations within the Benders algorithm. The hybrid code MSLiP-OSL [19,42] has been developed to give a more stable and accurate solution to each nodal LP problem, as well as to allow the possibility of crash-to-feasibility and presolve procedures.

An additional routine, NDAGGR, has been added to the MSLiP and MSLiP-OSL codes, which allows stages to be aggregated, see [19,42,43] for more details. An illustration of stage aggregation can be seen in figure 3. The aggregation scheme for this example combines the first three stages, then the next three and finally the last two. This takes a 144-scenario problem down to a 36-scenario problem.

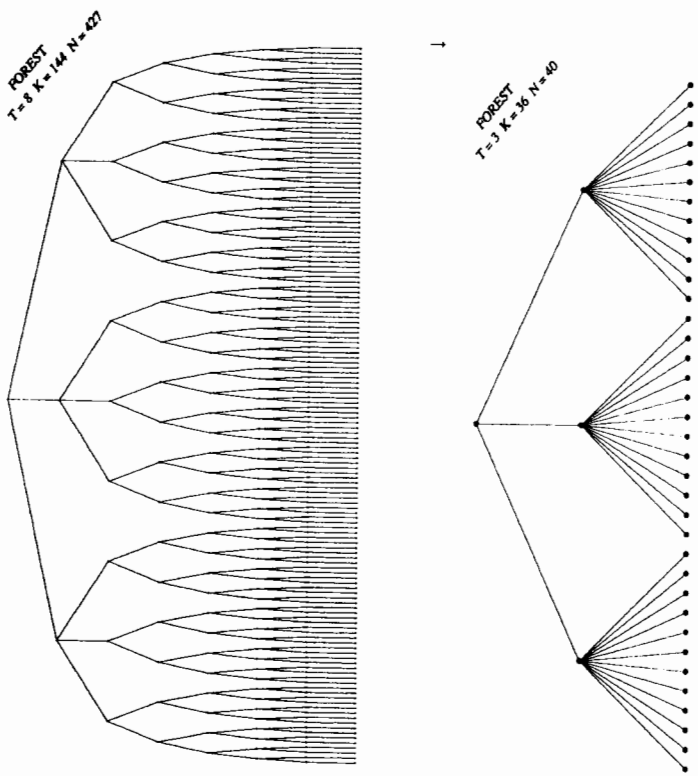

Figure 3. Aggregation of FOREST.8.144.

CPU time reduction can be obtained through stage aggregation. This may not be the case with MSLiP, where larger LP problems take an extremely long time to solve. With MSLiP-OSL, however, restructuring so that the earlier stages have larger nodal

dimensions, especially the first time stage, can both reduce the computational time considerably and increase the stability [19]. The nodal dimensions of the subproblems at other stages are dependent on the stochasticity; with higher stochasticity, more information is required from iteration to iteration and the nodal dimensions can be larger, with lower stochasticity, the nodal dimensions can be lower. Stage aggregation also reduces the number of iterations and cuts needed to reach a solution.

It has been shown [9,43] that MSLiP outperforms the simplex and interior-point methods – CPLEX was used for this testing – for large sparse problems. This, as well as the inherent parallel properties of nested Benders decomposition and the associated sampling scheme, led the authors to implement a parallel version of the MSLiP code.

4. Parallel MSLiP

Before explaining the parallel nested Benders decomposition method employed in this paper, a summary of other parallel methods is given. Parallelisation of the interior point method in effect means parallelisation of the Cholesky factorisation of the compound matrix used in calculating the iterative step. This has been done in a number of ways; by vectorising the procedure, by making use of the elimination tree to distribute the columns of the matrix to processors [36], by using a block Cholesky factorisation and distributing the blocks [13] and by using the dual Sherman–Morrison decomposition formulation for multistage recourse problems of stochastic programming due to Birge and Qi [47]. Parallelisation of the simplex method, on the other hand, is extremely difficult [26].

The augmented Lagrangian decomposition method has been parallelised by Ruszczyński [41] and Escudero et al. [25], while the similar row-action equivalent to augmented Lagrangian decomposition method used by Nielsen and Zenios [38] again decomposes by scenarios. The solution time has been shown to be reduced significantly on Thinking Machine's CM-2 when a system with more processing elements is used. These methods appear to be well structured for parallelisation, with each processor solving one particular scenario corresponding to a stochastic data process path and with non-anticipativity information being passed between processors. However, to the authors' knowledge the sequential algorithm has a relatively slow solution time when compared to methods such as nested Benders decomposition and the interior-point method.

Dantzig et al. [32] have a parallel implementation of their importance sampling based Benders decomposition code, while Ariyawansa and Hudson [2] parallelised the two-stage problem in a similar manner to the Dantzig method – the main difference being that each processor is assigned a subset of the second-stage nodal problems. Eckstein and Hiller [21] have produced a Benders-based decomposition scheme as a subsystem to a portfolio selection algorithm. Speed-up by a factor of 32 was obtained for the Benders-based portion of this algorithm on a CM-2/SUN-4 machine with 16,000 processing elements when compared with a sequential time measured on a SUN-4.

A parallel nested Benders decomposition which uses a farming strategy has been formed from the ND-UM code of Birge et al. [7]. The master processor starts the computation and initialises the slave processors. A subtree (see figure 1) is solved on each slave processor, with the root node of the subtree lying in a stage (not the first) of the whole tree structure. The master solves the portion of the tree up to the root node of the subtree. Slave processors do not branch from other slave processors.

A parallel version of MSLiP has been implemented on Fujitsu's AP1000 and IBM's SP2. See [19] for a description of the AP1000. The SP2 had 16 IBM RS6000 390 Power 2 processors, each with 128 Mbytes DRAM and double precision peak performance of 250 Mflops; therefore, any speed-up achieved is from a level which is faster than most sequential computers. The system was run under MPI and the theoretical peak for communication bandwidth was 40 Mb/s and a latency rating of 40 microseconds, with the host workstation being an RS6000 PowerPC processor.

The *parallel algorithm* described below is more similar to the methods of [2, 22, 32] than to that of [7]. The idea is to achieve load balancing, which cannot be achieved by splitting the tree into a set of subtrees, at the possible expense of communication overhead. It uses the *farming* technique, with a *master* and a number of *slaves* decided by the user. In this technique, the master processor allocates work to each of the slave processors and then receives information from each slave processor. The master processes this data and decides how the algorithm should proceed. The master sends LP information to a slave processor when that slave becomes available. The method can be described for a multicomputer attached to a host processor as follows.

Host

Step H: 0. Send master module to processor 0. Send slave module to processors $s = 1, 2, \ldots, S$. Send data to processor 0.

Step H: 4. Receive optimal solution from processor 0.

Master processor

Step M: 0. Receive module from host. Receive data from host.

Step M: 1. Solve master problem, obtaining trial solution.

Step M: 2. Assuming there are $m \geq S$ nodes to be solved in the current time stage, send information for the first S subproblems to slave processors $s = 1, 2, \ldots, S$.

Step M: 3. When an optimal solution is returned from a processor $s \in \{1, 2, \ldots, S\}$, send the information to solve another nodal problem to processor s unless there are no problems left to be sent. If there are no nodal problems remaining to be sent, go to step M: 4. If an infeasible solution is returned from a processor, go to step M: 4. If there is only one nodal problem left to be sent and there is only one slave available, this problem is solved on the master processor.

Step M: 4. When all nodal problems have been solved or a problem is found to be infeasible, the master determines whether to move forward or backward in the tree. The master forms the cuts for all nodes.
On reaching the first stage in the tree and finding there are no more subproblems to be solved in the tree, the optimal solution has been reached. It is returned to the host, and the slave processors are informed to stop. Otherwise, return to step M: 2.

Slave processor

Step S: 0. Receive module from host.

Step S: 2. Receive and solve LP problem sent from master.

Step S: 3. Return solution to master and return to step S: 2.

Step S: 4. Receive message from master to stop, and stop.

A slave processor requires new constraints (cuts), new right-hand sides and basis information to be sent from the master processor, while the master processor requires a new primal vector, a new dual vector, basis information and the status of the LP problem – feasible or optimal – from a slave. The slave processors operate as LP solvers in an asynchronous manner. A synchronising step occurs when the master processor changes the stage and forms the new cut information. Dynamic load-balancing occurs within this framework, as when a slave processor becomes available, the next nodal LP problem to be solved is automatically sent to that processor.

If the nodal subproblems are large enough, this algorithm should have reasonable load balancing properties, though there may be a communications bottleneck while slave processors try to communicate with the master. The sequential portion of the algorithm is dependent on the structure of the problem, but is relatively small when there are a large number of scenarios. Increasing the number of stages does not necessarily reduce this portion, as cut generation is performed on the master processor only. Amdahl's Law [1] does show that speed-up is generally possible for the test problems covered (see [43]).

Figures 4 and 5 plot speed-up against the number of slave processors. Problems throughout this paper are labelled in the following way: *Prob.Per.Scen*, where *Prob* gives the problem name, *Per* gives the number of periods and *Scen* gives the number of scenarios in the problem. Most of the problems listed in table 1 are stochastic versions of various well-known NETLIB problems, while the SGPF and WATSON problem sets [11, 27] are both portfolio management problems. The table gives the deterministic equivalent dimensions of each problem (rows, columns and nonzeros), the objective value (*Obj. val.*), the associated tree structure (*Tree*), which for these balanced trees is given by the amount of branching at each time stage, the *stochasticity* (this is defined in section 5) (*EVPI*), and their sequential solution times on the SP2 (*MSLiP*).The notation used to denote the branching of a balanced scenario tree is best

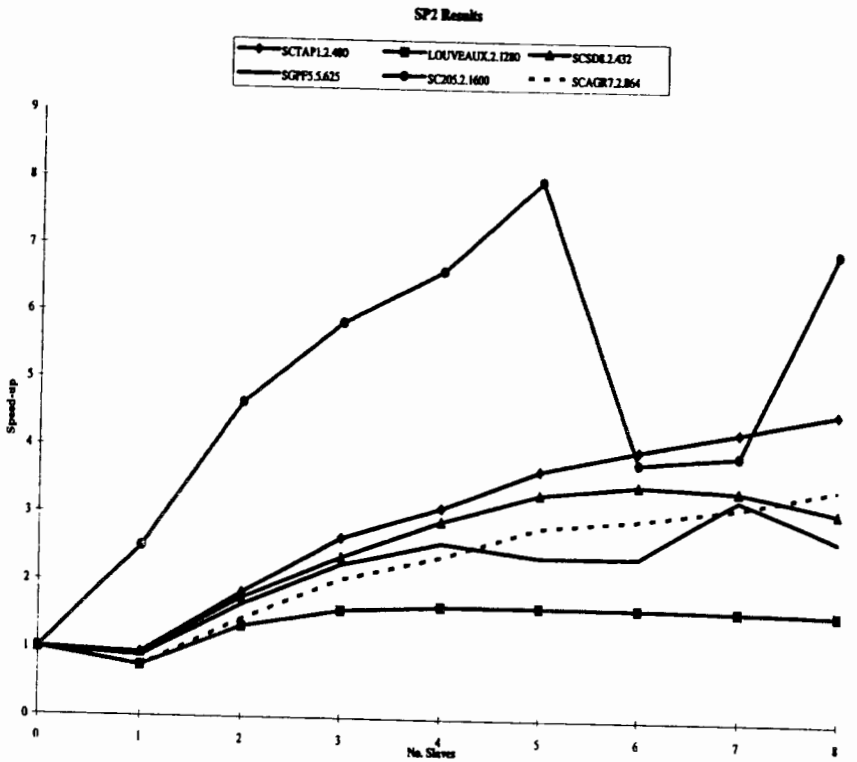

Figure 4. SP2 – various problem sets.

explained by example. Thus, $4^3 \cdot 2^3 \cdot 1$ means that from each node emanate four branches for the first three stages and two branches for the next three stages, with no branching at the seventh (penultimate) stage. Only a subset of the problems tested are given in this paper, but they show the general pattern for all the parallel performances that can be found in [43].

It is noticeable that the parallel algorithm generally performs better on two-stage problems than on multistage problems. Super-linear speed-up occurs on the SC205 problem set (which occurs because of anomalous behaviour), near-linear speed-up is obtained on the two-stage SCSD8 and SCTAP1 problem sets on 4 processors, while speed-up was achieved on both the multistage SGPF and WATSON problem sets. The nodal problem sizes for the SGPF and WATSON problems are larger than for the FOREST multistage problems, which have poorer parallel performance. It should be noted that for two-stage problems, our algorithm is equivalent to both the Birge et al. and Dantzig et al. parallelisation techniques [7,32].

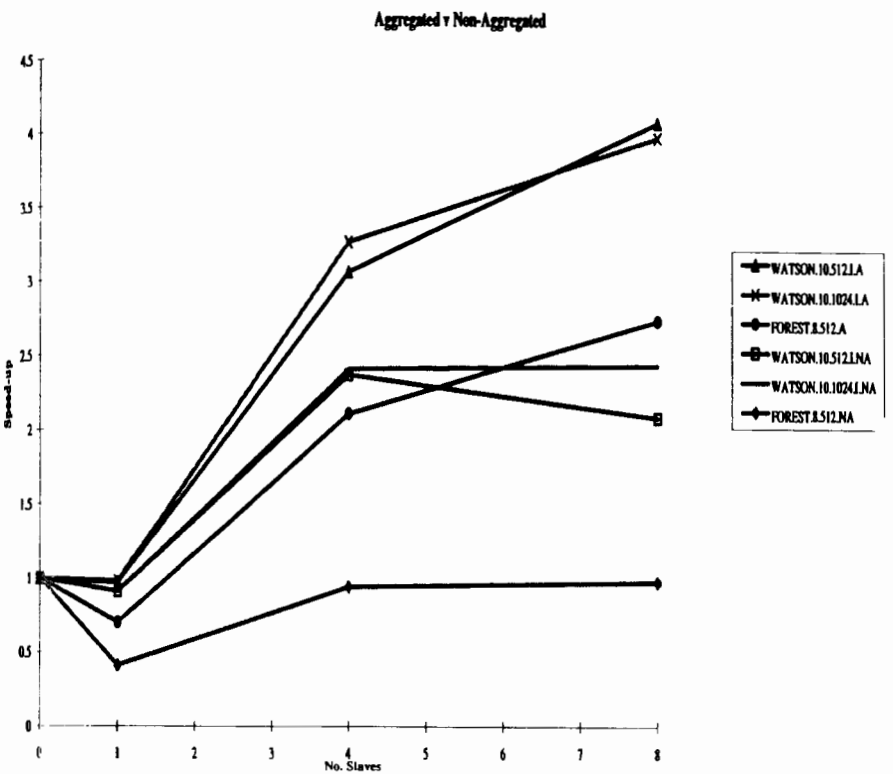

Figure 5. SP2 – various problem sets.

Table 1

Problem characteristics – a summary.

Name	Rows	Cols	Nonz	Obj. val	Tree	EVPI	MSLiP CPU
SC205.2.1600	35212	35214	96030	0.0	1600	–	31.25
SCSD8.2.432	8650	60550	190210	25.80	432	0.24	13.73
SCAGR7.2.864	32847	34580	108906	– 834446	864	0.01	29.67
SCTAP1.2.480	28830	46128	169068	248.50	480	0.20	25.59
LOUV.2.1280	8962	20484	40968	482.64	1280	9.35	60.34
FOREST.8.384	15667	14927	62795	– 43759	$4^2 \cdot 3 \cdot 2^3 \cdot 1$	2.75	32.86
FOREST.8.512	20771	19791	83307	– 43869	$4^3 \cdot 2^3 \cdot 1$	2.51	40.64
SGPF5.5.625	49202	61759	165570	– 5201.282	5^4	10.84	41.07
WATSON.10.512.1	67069	128001	350728	– 1959.63	2^9	44.63	201.96
WATSON.10.1024.1	134127	255987	701428	– 1926.79	$4 \cdot 2^8$	46.53	409.97

While Amdahl's Law would indicate that the obtainable speed-up is less for multistage problems [43] than for two-stage problems, the theoretical speed-ups are not achieved. This is partly due to the tree structure; multistage problems cannot be totally parallelised in later stages as there are not enough processors, while in the earlier stages, there may be more processors than nodal problems to be solved, resulting in some processors remaining idle. In the multistage problems, cuts have to be passed from master to slave, and on large problems where many cuts may be created, this increases the communication time significantly. If the communication-to-calculation ratio becomes too high, a bottleneck develops, with too many slave processors attempting to communicate with the master processor.

To reduce the communication-to-calculation ratio for multistage problems, stage aggregation has been employed (as shown in figure 3). Bigger nodal subproblems will lead to larger intervals between communication. A reduction in the number of cuts formed by the master should also ensue from this aggregation, reducing the number of sequential operations. Figure 5 compares aggregated speed-ups with the non-aggregated equivalent, with similar schemes as that shown in figure 3. The aggregated problems always perform better and it can be seen that for both the FOREST and WATSON problem sets, greater speed-ups can be achieved. The parallel algorithm is reasonably efficient on up to eight processors for the SP2.

Anomalous effects have occurred with certain problem sets where feasibility cuts have been placed, noticeably with the SC205 problem set, but also with the SCAGR7 set. With more than one slave, a different ordering in the return of nodal subproblems may lead to a different solution path. This occurs in other parallel applications, such as the branch and bound algorithm for mixed integer programming. This type of CPU reduction could be introduced into the serial algorithm by randomising the order in which nodal subproblems are solved within a time stage. Possible anomalies are classified as acceleration anomalies, deceleration anomalies and detrimental anomalies. *Acceleration anomalies* occur when the solution time achieved is faster than expected, while *deceleration anomalies* are exactly the opposite. *Detrimental anomalies* are severe deceleration anomalies where the solution time achieved is slower than the sequential solution time. A similar observation has been made by Hajian et al. [30]. (FOREST.8.512 does not experience a detrimental anomaly – there is no speed-up because the communication-to-calculation ratio is too high.)

5. Expected value of perfect information: EVPI

For a multistage recourse problem, each node of the tree has an associated *expected value of perfect information* value, called the *local* EVPI, defined as

$$\mathrm{EVPI}_t(\omega^t) := \pi_t(\omega^t) - \phi_t(\omega^t),$$

where $\pi_t(\omega^t)$ denotes the optimal value of the problem with root node defined by ω^t and

$$\phi_t(\omega^t) := \mathbb{E}_{\omega|\omega^t}\pi^t(\omega)$$

is the expected value of having perfect information about the future – i.e. the optimal value $\pi^t(\omega)$ along each data path ω from time t onwards from the data history path ω^t. The local EVPI values form a stochastic process, termed the *EVPI process* $\{\mathrm{EVPI}(\omega^t) : t = 1, \ldots, T\}$, which is a *supermartingale* [17]. The supermartingale property mathematically states the intuitive result that information becomes less important on average as t increases, i.e. closer to the horizon T. This leads to the following observations about the EVPI process:

(1) If the EVPI at a node is zero, all its descendant nodes have zero EVPI.

(2) A node in the tree whose descendants have no branching has zero EVPI.

(3) At the root node, the process value is the EVPI of the problem.

(4) The *local stochasticity* of a node of the tree is the local EVPI normalised by the optimal value of the whole problem. For the root node, this ratio is termed the *stochasticity* of the problem.

For each path which branches at, or after, the node ω^t, the optimal value is calculated with all nonanticipativity conditions ignored. These paths are all in the set $E_t(\omega^t) \in \mathcal{F}_t \subset \mathcal{P}(\Omega)$, the set of all subsets of scenarios at time t. The local EVPI value at a node of the tree at stage t is calculated by solving to find $\pi^t(\omega)$ for all $\omega \in E_t(\omega^t)$. Each $\pi^t(\omega)$ is an LP problem conditional on ω^{t-1} with all the data from ω^t along the path ω. This is multiplied by the conditional probability at ω^t for each path before the sum over $\omega \in E_t(\omega^t)$ is taken to obtain ϕ_t. This value is then subtracted from the actual optimal value $\pi_t(\omega^t)$ at the node to give the local EVPI.

To calculate the values of the EVPI process, this procedure has to be repeated for every node of the tree except for those in the last period (which have zero EVPI).

This procedure is performed in MSLiP as follows:

Step 1. Define an array called *EVPI*, say, with length equal to (or exceeding) the number of nodes in the tree. Set all values in the array to zero.

Step 2. Take the first scenario in the tree and set $t := T - 1$.

Step 3. Calculate the optimal value of the current path by a call to the LP solver and multiply it by the conditional probability of the occurrence of this path at the current node. Add this value to the corresponding position for this node in the *EVPI* array. Set $t := t - 1$. If $t < 1$, go to step 4, otherwise repeat step 3.

Step 4. Find the next path of the tree, set $t := T - 1$ and go to step 3. If all paths of the tree have been traversed, go to step 5.

Step 5. These calculations lead to the position where for each node ω^t, $t < T$, the value $\phi_t(\omega^t)$ has been calculated in the array *EVPI*, i.e. $EVPI_t(\omega^t) = \phi_t(\omega^t)$.

For all nodes, reset the value in the *EVPI* array by subtracting the calculated value from the nodal optimal value, i.e. in programming notation

$$EVPI_t(\omega^t) := \pi_t(\omega^t) - EVPI_t(\omega^t).$$

The values $\phi_t(\omega^t)$ are stored in the same vector as the $EVPI_t(\omega^t)$ values to save memory. This implementation applies to both the MSLiP and MSLiP-OSL codes.

6. EVPI-S sampling

Corvera Poiré and Dempster [12, 17] developed a method which can be considered to be based on artificial intelligence ideas to approximate the true EVPI and therefore the optimal solution of the problem. More precisely, their method utilizes *sequential importance sampling heuristics*, using the EVPI process as the importance criterion to push the sampled approximations towards the true optimal value. This is accomplished by increasing the number of sampled branches from a node of $\mathcal{T}_{n-1}$ with high relative EVPI in $\mathcal{T}_n$ and decreasing or resampling the branches with negligible EVPI relative to the sample problem EVPI. These heuristics can be used in a variety of ways to give a whole family of *EVPI- sampling* algorithms, which are termed EVPI-S. The sampling method employed by all the EVPI-S algorithms represents the data path tree $\mathcal{T}$ associated with the problem by sampling a sequence of trees $\mathcal{T}_1$, $\mathcal{T}_2, \ldots, \mathcal{T}_n$, where $\mathcal{T}_n$ is always more representative of $\mathcal{T}$ than $\mathcal{T}_{n-1}$.

The data scenario-subtree is formed and the corresponding problem is converted to the standard format SMPS using STOCHGEN (see [12]). It can then be solved with MSLiP or MSLiP-OSL and its EVPI values $EVPI_t(\omega^t)$ computed. Given that the problem corresponding to $\mathcal{T}_{n-1}$ has been solved and its local EVPI values computed, one of the following may occur at each node of the tree $\mathcal{T}_{n-1}$ to form $\mathcal{T}_n$:

(1) Where the nodal EVPI is above a certain zero tolerance level and its descendents have not been considered previously (there was no branching from them in the tree $\mathcal{T}_{n-1}$), the tree is *expanded* – to a depth of one or more stages – by independent sampling conditional on the appropriate nodal data so that more multiple branches to descendent nodes are considered in $\mathcal{T}_{n-1}$.

(2) At nodes where the local EVPI value is below the zero tolerance, a new subtree is generated by *resampling* the set of paths conditional on the path to the particular node, with or without replacement. The true local EVPI value may not be zero and so the local EVPI is thus re-estimated.

(3) At nodes where the local EVPI value is found to be repeatedly below the zero tolerance over a sequence of iterations, the subtree is *collapsed*. It becomes more certain with each iteration that this local value is indeed zero or very small. This collapsing is in fact resampling of a single path done from the parent node of the node found to have near-zero local EVPI. A period before which EVPI values

below the zero tolerance are not collapsed is required, with the period possibly being changed after each iteration.

In [12], the ratio of local EVPI to root optimal value – which corresponds to the *local stochasticity* – is the tolerance criterion used (1 or 5 percent). The algorithm can be stated as follows:

Step 1. Set the number of iterations to be performed, N say. (Alternatively, a termination condition could be used, such as when the rate of increase in EVPI decreases below a certain threshold.) Set $n := 1$.

Step 2. Sample a scenario-subtree $\mathcal{T}_n$ based on the current EVPI information carried by $\mathcal{T}_{n-1}$.

Step 3. Generate the corresponding SMPS files for the problem corresponding to the tree $\mathcal{T}_n$, using STOCHGEN.

Step 4. Solve the multistage recourse problem defined by the SMPS files with MSLiP or MSLiP-OSL and obtain the nodal EVPI values.

Step 5. If $n < N$, set $n := n + 1$. (In the finite scenario case, transfer the sample EVPI information from $\mathcal{T}_n$ to the full tree $\mathcal{T}$.) Otherwise stop, the sampling has terminated and an estimate of the optimal solution obtained.

The set of sampling procedures which are defined as above can greatly reduce the size of the problem to be solved so that more detailed problems, or problems with a continuous distribution of data paths, may be solved. Examples of these large reductions can be found in [12], which reports, for example, that SGPF5.3125 can be reduced from 3,125 to 677 scenarios, with a solution value within 1% of that of the full problem being obtained in 10% of the CPU time.

7. Parallel EVPI

In producing a fast EVPI-S sampling system which is unbiased in the small sample, the solution times of the nested Benders decomposition and EVPI calculations are highly important. Therefore, a parallel EVPI method has been developed for MSLiP. As the parallel EVPI is within the parallel MSLiP system, the algorithm is presented below as a continuation of the nested Benders parallel algorithm.

The host processor is as before, with step H: 4 relabelled as step H:7. It receives all the EVPI information.

Master processor

Step M: 0–Step M: 3. As before.

Step M: 4. When all nodes have been solved, the master determines whether to move forward or backward in the tree. The master forms the cuts for all nodes.

On reaching the first stage in the tree and finding there are no more subproblems to be solved in the tree, the optimal solution has been reached. Send message to slaves to perform EVPI calculation and go to step M: 5. Otherwise, return to step M: 2.

Step M: 5. Take the first scenario in the tree and set $t := T - 1$. Assuming there are $m \geq S$ paths in the tree, send information defining the first S paths to slave processors $s = 1, 2, \ldots, S$.

Step M: 6. When a solution for each LP defined along the path has been returned from a processor $s \in \{1, 2, \ldots, S\}$, add the value of each node in the path to the corresponding position for each node in the *EVPI* array. Send the information defining the next path to processor s unless there are no paths left to be sent. If there are no paths left to be sent, go to step M: 7.

Step M: 7. The EVPI for each node in the tree is calculated by subtracting the appropriate *EVPI* array value from the nodal optimal value.

The optimal solution has been reached and the EVPI values calculated for nodes of the tree. This information is returned to the host, and the slave processors are informed to stop.

Slave processor

Step S: 0–Step S: 3. As before.

Step S: 4. Receive message from master to perform EVPI calculation.

Step S: 5. Receive path information or termination condition from the master. If the termination condition has been received, then stop. Otherwise, calculate the optimal value of this path, multiply by the conditional probability of the occurrence of this path at the current node and store the value. Let $t = t - 1$ and repeat step S:5. Otherwise, go to step S: 6.

Step S: 6. If $t < 1$, the set of optimal values, one for each node (apart from the leaf node) along the path, are sent to the master processor. Return to step S: 5.

The communication required between the master and slave processors when there are more than two stages is much less for this algorithm than for the nested Benders algorithm. The load-balancing property observed in the nested Benders parallel code is maintained, with each newly available processor receiving the latest information. The results reported here are for the SP2 implementation running under MPI. Large problems were again run on the SP2 on up to 8 slaves processors. The solution of two-stage problems does not gain from the addition of extra processors and, as EVPI is appropriate only for multistage problems in terms of the sampling algorithm (a two-stage problem has only a single EVPI value at the root node), results are not shown here. The algorithm appears to perform reasonably on three-stage problems and is

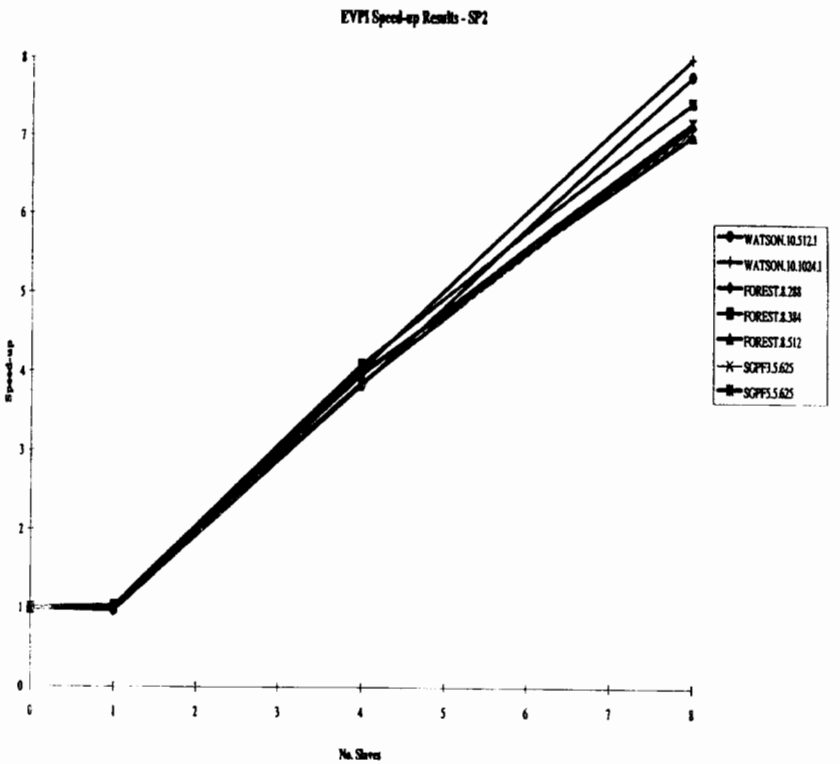

Figure 6. EVPI speed-up on the SP2.

Table 2

SP2 EVPI results.

Name	Slaves								
	Serial	1	2	3	4	5	6	7	8
WATSON.10.512.I	1.00	0.97	1.95	2.92	3.81	4.84	5.80	6.77	7.62
WATSON.10.1024.I	1.00	1.00	2.02	3.03	4.01	5.03	6.04	6.99	7.97

approximately 50% efficient on 8 slave processors. For problems of more than three stages, its efficiency and speed-up are high (>90% efficiency), especially for the FOREST, WATSON.I and SGPF problem sets, see figure 6. Table 2 shows the speed-up on 1 to 8 slaves processors for problems WATSON.10.512.I and WATSON.10.1024.I.

It can be seen that they both achieve near-linear speed-up. Amdahl's effect can be seen in the efficiency increase for the larger problem.

On the AP1000, up to 35 processors were used to test the performance of the parallel EVPI implementation. FOREST.8.288 achieved a speed-up of 29 on 35 slaves, an efficiency of 83%. A large SGPF problem achieved speed-up of 18 on 24 slaves, an efficiency of 75%. The WATSON problem set required too much DRAM to be solved on the Fujitsu machine.

The efficiency of the EVPI calculation suggests the use of parallelisation for the EVPI-S sampling technique. The EVPI takes a significant proportion of the computational time for the multistage problems for which the sampling technique was designed. An example is shown in figure 7 for the parallel MSLiP code. The three iterations of the parallel EVPI-S procedure have different parallel performances; as the size of the problem increases ($\mathcal{T}_n > \mathcal{T}_{n-1}$), the parallelisation improves. Since MSLiP-OSL performs the EVPI calculation much faster than MSLiP, it is to be preferred in a sequential sampling algorithm. The parallelisation of the MSLiP-OSL code would lead to lower speed-up for the WATSON.10.1024.I problem shown in figure 7, since the solution time of the EVPI calculation would play a smaller part in the overall solution time. (Unfortunately, this has not been performed as the OSL software has not been installed on either parallel machine.) However, as the solution of a whole set of problems is needed when sampling, each of which can require a large amount of CPU time, even lower speed-ups can still be extremely useful. The problem shown can be solved in parallel by EVPI-S on the SP2 with 8 processors in ~9.1 min versus solution in parallel of the full problem (with EVPI calculation) in ~11.5 min. The full problem is solved serially in ~75 min (parallel speed-up 6.5/8). For much larger problems, the advantage of the parallel EVPI-S algorithm will be still further highlighted.

8. Conclusions

As nested Benders decomposition for linear multistage recourse stochastic programming problems compares favourably with both interior-point and simplex methods, and can be stabilised for difficult problems by the use of stage aggregation, it appears to be a sensible method to parallelise, especially as there is a sampling procedure which utilises this method. This paper is mainly concerned with the parallelisation of nested Benders decomposition and the EVPI calculation within the MSLiP code. These algorithms are integrated into the EVPI-S sampling algorithm for solving linear multistage recourse problems with large, continuous state, stochastic structures.

Along with most coarse-grain parallel algorithms for solving multistage recourse problems, this parallel nested Benders algorithm uses a farming strategy, as does the parallel EVPI algorithm. Each slave processor solves an LP problem, with the master processor directing operations. This type of algorithm has good load-balancing properties. Results on a variety of multicomputing platforms show that, as with other parallel

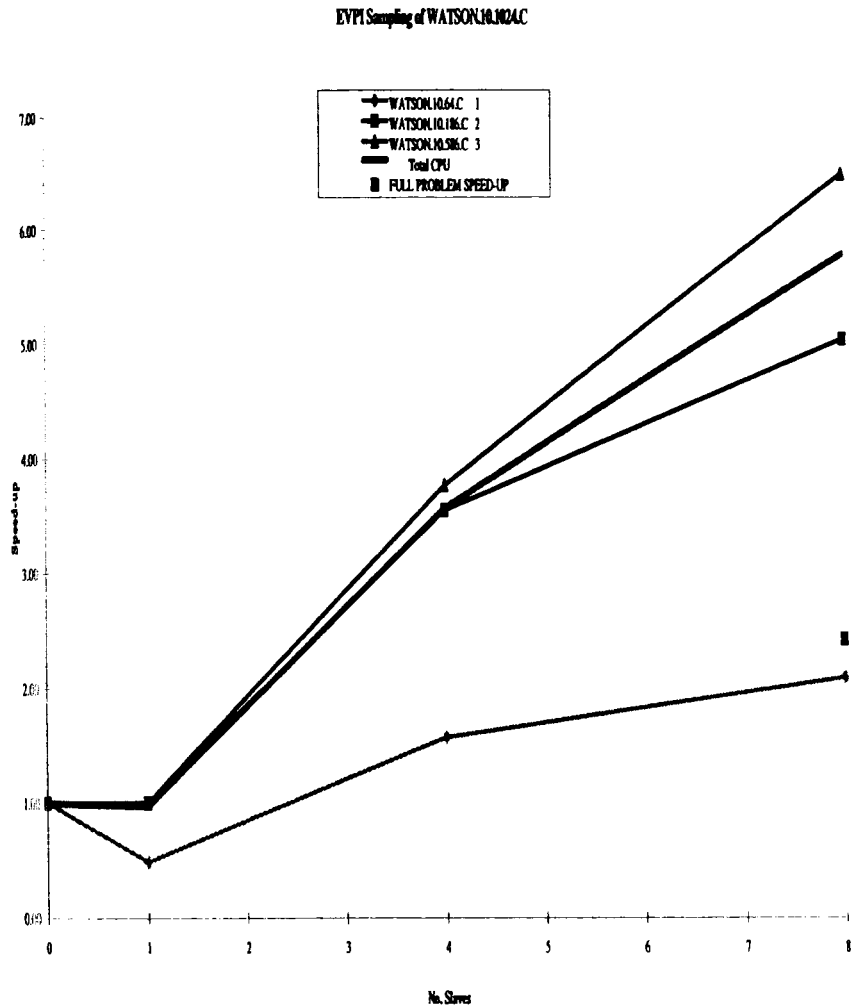

Figure 7. EVPI-S speed-up on the SP2.

decomposition strategies, the performance of the algorithm is problem dependent. It is particularly dependent on the communication-to-calculation ratio of the particular platform.

The problem sets on which the parallel algorithm performs best – with near-linear speed-up on up to 8 processors – had larger nodal LP problems to solve. The length of time spent communicating between processors is critical to the performance

of the algorithm, as is the tree structure and the percentage of the total time spent forming cuts. Through the use of aggregation, a reduction in the time spent forming cuts and an increase in nodal problem dimensions can be achieved so that higher levels of speed-up can be achieved. The algorithm may be prone to both detrimental and acceleration anomalies due to different solution paths occurring in the parallel algorithm.

As the communication-to-calculation ratio is critical to performance, and the data needs to be replicated on each processor, a further improvement to the algorithm may be made by converting the algorithm from the multicomputing to the multiprocessing environment. Shared memory machines will generally have large amounts of centralised RAM and communication will generally be much faster, improving both the parallel nested Benders and parallel EVPI algorithm.

Further methods which could improve the parallel algorithm are:

(1) A hybrid parallel code could be attempted which is a cross between the parallel algorithm presented here and the Birge et al. [7] parallel algorithm, with a slave processor solving a node and its children, say, before returning a solution to the master processor. There may, however, be a problem with transferring cut information to the appropriate processor in this case.

(2) A slave processor could solve a set of LP problems at the same stage before returning solutions to the master. This resembles the parallel EVPI calculation. The load-balancing property of the algorithm can be maintained by not initially allocating all the LPs to be solved to the slave processors.

(3) A semi-distributed load balancer could be employed with more than one processor forming cuts. Depending on the node, a particular processor may form the cut and broadcast the information to other slave processors.

The penultimate section of this paper concentrates on the EVPI-S sampling algorithm. For a further reduction in solution time and to make the whole system viable in a practical setting, the parallelisation of the EVPI calculation was needed. This is a much simpler operation than that of the parallelisation of nested Benders decomposition and has a very small percentage of sequential calculations in the multistage case. Near-linear speed-up was achieved on most problems with more than three stages. The parallel EVPI-S algorithm incorporating the parallel EVPI calculation shows reasonable speed-up on up to 8 processors.

Acknowledgements

We are grateful to Professor Tony Hey and the Southampton High Performance Computing Consortium, and Fujitsu (UK) Limited and the Imperial College Fujitsu Parallel Computing Research Centre for access to and support for the parallel computing facilities used in the experiments reported here.

References

[1] G.M. Amdahl, Validity of the single processor approach to achieving large scale computing abilities, AFIPS Conference Proceeding 30(1967)483–485.
[2] K.A. Ariyawansa and D.D. Hudson, Performance of a benchmark parallel implementation of the Van-Slyke and Wets algorithm, Concurrency: Practice and Experience 3(1991)109–128.
[3] J.F. Benders, Partitioning procedures for solving mixed-variable programming problems, Numerische Mathematik 4(1962)238–252.
[4] P. Billingsley, *Probability and Measure*, 2nd ed., Wiley, New York, 1980.
[5] J.R. Birge, Decomposition and partitioning methods for multistage stochastic linear programs, Operations Research 33(1985)989–1007.
[6] J.R. Birge, C.J. Donohue, D.F. Holmes and O.G. Svintsitski, ND-UM version 1.0 computer code for the nested decomposition algorithm, Department of Industrial and Operations Engineering, University of Michigan, August 1994.
[7] J.R. Birge, C.J. Donohue, D.F. Holmes and O.G. Svintsitski, A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs, Technical Report 94-1, Department of Industrial and Operations Engineering, University of Michigan, 1994.
[8] G.R. Carino, T. Kent, D.H. Myers, C. Stacy, M. Sylvanus, A. Turner, K. Watanabe and W.T. Ziemba, The Russell–Yasuda Kasai model. An asset/liability model for a Japanese insurance company using multistage stochastic programming, Interfaces 24(1994)29–49.
[9] G. Consigli and M.A.H. Dempster, Dynamic stochastic programming for asset–liability management, Annals of Operations Research 81(1998)131–161.
[10] G. Consigli, M.A.H. Dempster and N. Hicks Pedrón, Comparative evaluation of algorithms for dynamic stochastic programming, Working Paper, Judge Institute of Management Studies, University of Cambridge, 1999.
[11] G. Consigli, M.A.H. Dempster, D.G. Richards and R.T. Thompson, The WATSON pension fund management problems, http://www-cfr.jims.cam.ac.uk/research/sptp.html (1996).
[12] X. Corvera Poiré, Model generation and sampling algorithms for dynamic stochastic programming, Ph.D. Thesis, Department of Mathematics, University of Essex, September 1995.
[13] P. D'Ambra and M.I. Sales, Concurrent banded Cholesky factorization on workstation networks, Working Paper, Department of Mathematics and Applications, University of Naples and IBM ECSEC, Rome, 1993.
[14] G.B. Dantzig and P. Wolfe, Decomposition principle for linear programs, Operations Research 8(1960)101–111.
[15] M.A.H. Dempster (ed.), *Stochastic Programming*, Academic Press, London, 1980.
[16] M.A.H. Dempster, On stochastic programming II: Dynamic problems under risk, Stochastics 25 (1988)15–42.
[17] M.A.H. Dempster, Sequential importance sampling algorithms for dynamic stochastic programming, Research Papers in Management Studies WP 32/98, Judge Institute of Management Studies, University of Cambridge, 1998.
[18] M.A.H. Dempster and H.I. Gassmann, Using the expected value of perfect information to simplify the decision tree, *Abstracts 15th IFIP Conference on System Modelling and Optimization*, IFIP, Zurich, 1991, pp. 301–303.
[19] M.A.H. Dempster and R.T. Thompson, Parallelization and aggregation of nested Benders decomposition, Annals of Operations Research 81(1998)163–187.
[20] J. Dupačová, Multistage stochastic programs: The state-of-the-art and selected bibliography, Kybernetika 31(1995)151–174.
[21] J. Eckstein and R.S. Hiller, Stochastic dedication: Designing fixed income portfolios using massively parallel Benders decomposition, Working Paper 91-02, Harvard Business School, 1991.
[22] R. Entriken, The parallel decomposition of linear programs, Technical Report SOL 89-17, Department of Operations Research, Stanford University, 1989.
[23] Yu. Ermoliev and R.J-B Wets (eds.), *Numerical Techniques for Stochastic Optimization*, Springer, Berlin, 1988.

[24] L.F. Escudero, P.V. Kamesan, A.J. King and R.J-B Wets, Production planning via scenario modelling, Annals of Operations Research 43(1993)311–335.
[25] L.F. Escudero, J.L. de la Fuente, C. Garcia and F.J. Prieto, A parallel computation approach for solving multistage stochastic network problems, Annals of Operations Research (1999), this volume.
[26] J.J.H. Forrest and J.A. Tomlin, Vector processing in simplex and interior methods, Annals of Operations Research 22(1990)71–100.
[27] K. Frauendorfer, C. Marohn and M. Schrüle, SG-portfolio test problems for stochastic multistage linear programming, Technical Report, Institute of Operations Research, University of St. Gallen, 1995.
[28] H.I. Gassmann, MSLiP: A computer code for the multistage stochastic linear programming problem, Mathematical Programming 47(1990)407–423.
[29] H.I. Gassmann, MSLiP 8.2 User's Guide, School of Business Administration, Dalhousie University, 1992.
[30] M.T. Hajian, I. Hai and G. Mitra, A distributed processing algorithm for solving integer programs using a cluster of workstations, Research Report, Department of Mathematics, Brunel University, September 1994.
[31] *IBM Optimization Subroutine Library (OSL)*, 4th ed., IBM Corporation, 1992.
[32] G. Infanger, *Planning Under Uncertainty. Solving Large Scale Stochastic Linear Programs*, Boyd and Fraser, Danvers, 1994.
[33] P. Kall and S.W. Wallace, *Stochastic Programming*, Wiley, 1994.
[34] A. King, *SP/OSL Version 1.0 Stochastic Programming Interface User's Guide*, IBM Research Division, Yorktown Heights, New York, 1994.
[35] M. Lane and P. Hutchinson, A model for managing a certificate of deposit portfolio under uncertainty, in: [16], pp. 473–496.
[36] R. Levkovitz and G. Mitra, Solutions of large-scale linear programs: A review of hardware, software and algorithmic issues, in: *Optimization in Industry*, eds. T.A. Ciriani and R.C. Leachman, Wiley, 1993.
[37] J.M. Mulvey and H. Vladimirou, Stochastic network programming models for financial planning problems, Management Science 38(1992)1642–1664.
[38] S.S. Nielsen and S.A. Zenios, Solving multistage stochastic network programs on massively parallel computers, Mathematical Programming 73(1996)227–250.
[39] C.E. Pfefferkorn and J.A. Tomlin, Design of a linear programming system for ILLIAC-V, Technical Report SOL 76-8, Department of Operations Research, Stanford University, and Technical Report 5487, NASA-Ames Institute for Advanced Computation, Sunnyvale, CA, 1976.
[40] R.T. Rockafellar and R.J-B Wets, Scenarios and policy aggregation in optimization under uncertainty, Mathematics of Operations Research 16(1991)309–333.
[41] A. Ruszczyński, Parallel decomposition of multistage stochastic programming problems, Institute of Automatic Control, Warsaw University of Technology, 00665 Warsaw, Poland, August 1991.
[42] R.T. Thompson, MSLIP-OSL 8.3 User's Guide, Judge Institute of Management Studies, University of Cambridge, 1997.
[43] R.T. Thompson, Fast sequential and parallel methods for solving multistage stochastic linear programmes, Ph.D. Thesis, Department of Mathematics, University of Essex, 1997.
[44] C. Vassiadou-Zeniou and S.A. Zenios, Robust optimization models for managing callable bond portfolios, European Journal of Operational Research 91(1996)264–273.
[45] R. Van Slyke and R.J-B Wets, L-shaped linear programs with applications to optimal control and stochastic programming, SIAM Journal of Applied Mathematics 17(1969)638–663.
[46] R. Wittrock, Dual nested decomposition of staircase linear programs, Mathematical Programming Study 24(1985)65–86.
[47] D. Yang and S.A. Zenios, A scalable parallel interior point algorithm for stochastic linear programming and robust optimization, Computational Optimization and Applications (1997).

Annals of Operations Research 90(1999)185–202

Parallel local search for Steiner trees in graphs

M.G.A. Verhoeven[a] and M.E.M. Severens[b]

[a]*Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

E-mail: marcov@win.tue.nl

[b]*Océ Technologies, P.O. Box 101, 5900 MA Venlo, The Netherlands*

This paper discusses sequential and parallel local search for the Steiner tree problem in graphs. We introduce novel neighborhoods whose computational time and space complexity is smaller than those known in the literature. We present computational results for benchmark instances from [3] and instances derived from real-world traveling salesman problem instances, which contain up to 18,512 vertices and 325,093 edges. These results show that good-quality solutions can be obtained in moderate running times. Furthermore, we present a parallel local search algorithm based on multiple-step parallelism and an optimal polynomial-time combination function. Computational results show that good speed-ups can be obtained without loss in quality of final solutions.

1. Introduction

In the Steiner tree problem in graphs, a minimum weight tree has to be found that includes a prespecified subset of vertices of a graph. The Steiner tree problem occurs in several practical applications, such as the design of telephone, pipeline, and transportation networks, and the design of integrated circuits. Although the Steiner tree problem is NP-hard [8], several sophisticated optimization algorithms for it exist, which are able to solve instances with up to 2,500 vertices and 62,500 edges at the cost of substantial amounts of computation time, viz., several hours on a powerful workstation or supercomputer. In addition, many heuristics have been proposed, most of which have running times that are polynomially bounded at the risk of finding suboptimal solutions. In [8], an overview of both optimization algorithms and heuristics is presented.