On Bridging the Gap Between Stochastic Integer Programming and MIP Solver Technologies

Gyana R. Parija • Shabbir Ahmed • Alan J. King

Mathematical Sciences Department, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.
School of Industrial & Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive, Atlanta, GA 30332, USA.
Mathematical Sciences Department, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA.

parija@us.ibm.com • *sahmed@isye.gatech.edu* • *kingaj@us.ibm.com*

April 12, 2001
Revised: April 30, 2002, October 3, 2002

Stochastic integer programs (SIP) represent a very difficult class of optimization problems arising from the presence of both uncertainty and discreteness in planning and decision problems. Although applications of SIP are abundant, nothing is available in the way of computational software. On the other hand, commercial software packages for solving *deterministic* integer programs have been around for quite a few years, and more recently, a package for solving stochastic *linear* programs has been released. In this paper, we describe how these software tools can be integrated and exploited for the effective solution of general purpose SIPs. We demonstrate these ideas on four problem classes from the literature, and show significant computational advantages.

(Stochastic programming; Integer Programming; Branch & Bound; Software)

---

1. Introduction

Recent years have witnessed widespread use of stochastic programming to address planning problems under uncertainty (Birge & Louveaux 1997, Kall & Wallace 1994). In a typical setting, the uncertainty is resolved by specifying a set of scenarios and the problem is reduced to deterministic, albeit large-scale, mathematical program – known as the *deterministic equivalent*. For linear structures, a wide array of efficient decomposition algorithms have been proposed in the research literature (Birge 1985, Rockafellar & Wets 1991, Ruszczyński 1986, Higle & Sen 1996, Van Slyke &

Wets 1969) and have been successfully implemented in computational software (Birge, Donohue, Holmes & Svintsitski 1996, Gassmann 1990, IBM 1998).

Unfortunately standard stochastic programming decomposition algorithms break down in the presence of discrete problem variables. Such stochastic integer programming (SIP) problems arise, for example, in production planning (Dempster, Fisher, Jansen, Lageweg, Lenstra & Rinnooy Kan 1981, Lenstra, Rinnooy Kan & Stougie 1983, Bitran, Haas & Matsuo 1986), scheduling (Dempster 1982, Tayur, Thomas & Natraj 1995, Birge & Dempster 1996), routing (Spaccamela, Rinnooy Kan & Stougie 1984, Laporte, Louveaux & Mercure 1989, Laporte, Louveaux & Mercure 1992), location (Laporte, Louveaux & van Hamme 1994), capacity expansion (Ahmed, Parija & King 2002, Bienstock & Shapiro 1988), electricity production (Takriti, Birge & Long 1996, Carøe, Ruszczyński & Schultz 1997), environmental control (Norkin, Ermoliev & Ruszczyński 1998), and finance (Dert 1995). Recently quite a few new algorithms for stochastic integer programming have been proposed. These techniques rely on integer programming duality or Lagrangian duality to decompose the problem, in conjunction with branch and bound or enumeration techniques (Ahmed, Tawarmalani & Sahinidis 2000, Carøe & Schultz 1999, Carøe & Tind 1998, Laporte & Louveaux 1993, Norkin et al. 1998). Although these ideas are theoretically promising, a great number of practical issues concerning their successful software implementation remain open. For a detailed discussion of stochastic integer programming, the reader is referred to various survey articles (van der Vlerk 1995, Schultz, Stougie & van der Vlerk 1996, Stougie & van der Vlerk 1997).

With the scenarios specified, a stochastic integer program is essentially a large-scale structured integer program. Thus, at least in principle, standard deterministic mixed-integer programming (MIP) techniques implemented in commercial software, such as CPLEX (ILOG Inc. 1997), OSL (IBM 1991), XPRESS (Dash Assoc. 1999), are applicable.

General purpose MIP software packages spend a significant amount of time analyzing problem matrix data structures to generate logical relationships existing among cuts (rows) and integer variables automatically. These additional relationships help tighten the underlying formulation, thus minimizing the effort required to close the linear programming (LP) relaxation gap. However, for certain problem formulations, these relationships are too complex to be automatically detected by a commercial solver in reasonable times, thus causing the solver to perform poorly.

The scenario tree structure underlying stochastic integer programs carries complex, hard-to-detect but very useful information about important scenarios, implication lists (linking the integer variables) and branching priorities of sets of integers associated with various stages and scenarios. However, the modelling infrastructure provided by commercial MIP solvers are often inadequate to hold such rich knowledge about the underlying model. Therefore, when existing MIP solvers attempt to solve the specially structured MIPs underlying stochastic integer programs, the exercise is equivalent to discarding the structural knowledge completely, and then computationally re-discovering a small subset of the discarded knowledge at a high price. Therefore, it should not come as a surprise that straightforward application of commercial MIP solvers are doomed to perform terribly on stochastic integer programs.

In 1998, IBM released the first commercial grade stochastic optimization modeling/solver software product, IBM OSL Stochastic Extensions (IBM 1998), or OSLSE, that

1. allows easy modelling of stochastic linear, quadratic and integer programs using scenario tree data structures,
2. allows easy manipulation of tree objects and elements (nodes, scenarios etc), and
3. provides a set of specialized solver algorithms that exploit the underlying tree structure for solving stochastic linear programs effectively.

It has already been demonstrated in (King & Wright 2001) that the tree structure in stochastic *linear* programs makes them amenable to various decomposition algorithms implemented in IBM OSLSE. This motivated us to explore OSLSE as a modelling environment and its integration with a standard MIP solver (OSL MIP) for solving stochastic *integer* programs. The key challenge is to exploit the stochastic tree structure in improving the problem formulation and to guide the branch and bound process in the MIP solver for effective resolution of the problem.

The remainder of the paper is organized as follows. In Section 2, we briefly describe the underlying general strategy for attacking stochastic integer programs by integrating OSLSE and OSL MIP. The next two sections (Sections 3 and 4) describe the applications of the proposed strategy on two sets of stochastic integer programs from the literature. For both problem sets, we describe problem-specific reformulation techniques to improve the LP relaxation bounds, as well as specialized branching rules to expedite branch and bound convergence. In Section 5, we apply the proposed method to an industrial-scale test problem arising in the planning of machine tool purchases in a semiconductor manufacturing facility. Finally, in Section 6, we consider a class of stochastic dynamic capacity acquisition and assignment problems from the literature. In each of the test cases, we demonstrate significant computational advantages over using straightforward MIP techniques. Finally, Section 7 concludes the paper with some closing remarks.

2. Using OSLSE for SIP

It is well known that the performance of a branch and bound based MIP solver can be greatly enhanced with

- a "tight" problem formulation,
- an efficient strategy for solving the LP relaxations,
- and intelligent branching rules to guide the search of the branch and bound tree.

Successful application of each of the above requires careful understanding and exploitation of the underlying problem structure. As mentioned earlier, for stochastic programs expressed as deterministic equivalent mathematical programs, the underlying special structure is invisible to general purpose solvers. In the following subsections we outline ways of exploiting structure in each of the above areas.

2.1 Tightening of Problem Formulation

The matrix underlying multistage stochastic linear programs has a nested L-shaped structure for which specialized decomposition algorithms exist (Van Slyke & Wets 1969, Birge 1985). In order to tighten the formulation, automatic reformulation techniques (available in commercial solvers), such as *coefficient strengthening* and *cut generation*, tend to destroy the L-shaped structure of the underlying stochastic linear program. We take a different approach in exploiting the power of such techniques. We extract "blocks" of the matrix that correspond to a collection of nodes belonging to a subtree of the scenario tree and apply these techniques to these blocks. This scheme retains the L-shaped structure of the constraint matrix, thereby allowing the use of decomposition algorithms in solving the LP relaxation. Furthermore, the L-shaped structure present in each of these blocks allows easy exploration of logical relationships in a reasonable time and allows for further tightening during the branch-and-bound process.

2.2 Solving the LP Relaxation using the Nested L-shaped Method

The L-shaped method for linear programming is a "constraint generation" method based on a partitioning of the decision variables (Van Slyke & Wets 1969). An important aspect of the L-shaped method is that it allows breaking large problems into smaller problems of similar structure. We use the nested L-shaped method (Birge 1985, King & Wright 2001) for effective resolution of the LP relaxations of the stochastic integer programs. The nested L-shaped method, as implemented in OSLSE, is a flexible partition version of the L-shaped method that allows the user to manipulate the size and composition of the resulting subproblems. The result is that existing high-quality linear programming solvers could be exploited to a much greater extent.

2.3 Searching the Branch-And-Bound Tree

After preprocessing and solving the LP-relaxation using the Nested L-shaped decomposition algorithm, we invoke a branch and bound algorithm. The algorithm can be implemented within the MIP solver framework of any commercial solver that has the necessary infrastructure for modelling a scenario tree. The essential features of the branch-and-bound algorithm are

- **Node selection:** Select a node satisfying user-specified selection criteria from the list of open nodes (with integer infeasibilities). STOP if such a list is empty.
- **Branch selection:** Select variables for nodes of the scenario tree satisfying user-specified scenario/stage priority criteria
- **Branch direction selection:** Set branching directions according to user-specified pseudo-costs to create children nodes
- **Node LP tightening:** Tighten the node LP formulation using branching information and tree structure

- **Node LP evaluation:** Invoke the dual simplex algorithm to resolve the node LPs after branching
- **Cut generation:** Generate and add cuts separating fractional solutions from the integer polyhedron and resolve the LP.

2.4 Implementation

Each of the above ideas require the ability to manipulate the underlying scenario tree structure in stochastic integer programs. For our implementation, we decided to exploit the flexible architecture of the OSL Stochastic Extension (OSLSE) product to this effect. OSLSE allows one to input the model data by scenarios, define the integer variables as sets associated with the nodes of the scenario tree, prioritize the sets for branching purposes, and devise a node selection strategy that would quickly yield feasible integer solutions to important scenarios (in contrast to a feasible integer solution for the overall MIP). OSLSE aids the MIP preprocessor/solver in the solution effort by transforming the tree structure information into appropriate logical data structures such as implication lists, pseudo costs, sets and priorities, that can be accommodated by the MIP solver. The OSLSE architecture retains the stochastic structure of the problem, thereby allowing easy adaptation of reformulation techniques. Furthermore, the decomposition algorithms in OSLSE greatly expedites the solution of the LP relaxation sub-problems. OSLSE does not have a MIP solver of its own, rather, it uses the MIP solver algorithm in OSL to solve the deterministic equivalent of the underlying stochastic MIP. Moreover, the amount of data associated with the deterministic equivalent of a stochastic MIP can be enormous, hence, OSLSE provides a rich suite of methods for accessing and manipulating data organized by nodes of the scenario tree. Also, the flexible design of OSLSE allows one to reuse existing base OSL tuning strategies (such as branch and node selection user callbacks) in a seamless manner.

In the following four sections, we describe the use of OSLSE in solving four sets of stochastic integer programming test problems.

3. Test Set I: SIZES

3.1 Problem Description

Our first test set is a collection of stochastic product substitution problems called SIZES. Given a product that is available in a finite number of sizes (1 to N) and that a larger sized item can be cut to meet the demand of a smaller sized item, this problem consists of planning the production of each item size under demand uncertainty. A stochastic integer programming formulation of the problem as proposed by Jorjani, Scott & Woodruff (1999) is as follows:

$$\min \sum_{s \in S} p_s \sum_{t=1}^{T} \sum_{i=1}^{N} \left[(\alpha_i y_{its} + \beta z_{its}) + r \sum_{j<i} x_{ijts} \right] \tag{1}$$

s.t.

$$0 \leq y_{its} \leq M z_{its} \qquad \forall i,t,s \tag{2}$$

$$\sum_{i=1}^{N} y_{its} \leq C_{ts} \qquad \forall t,s \tag{3}$$

$$\sum_{t' \leq t} \left[\sum_{j \leq i} x_{ijt's} - y_{its} \right] \leq 0 \qquad \forall i,t,s \tag{4}$$

$$\sum_{i=1}^{N} x_{ijts} \geq d_{jts} \qquad \forall j,t,s \tag{5}$$

$$z_{its} \in \{0,1\}, \quad x_{ijts} \geq 0 \qquad \forall i,t,s \tag{6}$$

$$x, y, z \in \mathcal{N} \tag{7}$$

In the above formulation, the indices $i \in \{1,\ldots,N\}$ correspond to item sizes, $t \in \{1,\ldots,T\}$ to periods, and $s \in \{1,\ldots,S\}$ to scenarios, respectively. The decision variables y_{its} and z_{its} correspond to the production and set-up decisions for an item of size i in period t under scenario s, respectively, and the variables x_{ijts} correspond to how much of item size i to cut down to an item size $j(< i)$ in period t under scenario s. The parameters α_i, β and r represent the production, set-up, and cutting costs, respectively, and C_{ts} and d_{jts} represent the production capacity and the demand for the various item sizes. Constraints (2)-(6) connect the production and set-up decisions, enforce the capacity restrictions, establish inventory balance, assure demand satisfaction, and enforce integrality and non-negativity, respectively. The last constraint (7) enforces the non-anticipativity restrictions across the scenarios (Birge & Louveaux 1997). A complete description of the model is provided in (Jorjani et al. 1999).

As in (Jorjani et al. 1999), we consider two-period ($t = 1, 2$) instances of SIZES. The problems involve mixed-integer variables in both first and second stage. The test problem set includes three problems: SIZES3, SIZES5, and SIZES10, having 3, 5, and 10 scenarios, respectively. The size of the deterministic equivalent integer program for each of these test problems is presented in Table 1.

Problem	Binary Variables	Continuous Variables	Constraints
SIZES3	40	260	142
SIZES5	60	390	186
SIZES10	110	715	341

Table 1: SIZES: Problem dimensions

3.2 Initial Computations

As a first attempt, we solved the instances SIZES3, SIZES5, and SIZES10 using OSL Version 3.0 MIP solver. All computations have been carried out on an IBM Thinkpad Model 770 (with 233 MHz speed and 256 MB RAM) running Windows NT 4.0 for our computational experiments.

The LP relaxation of SIZES yields fairly good lower bounds (the LP relaxation gap being within 2%). Furthermore, generating a feasible solution for this class of problems is extremely easy – simply rounding up the LP relaxation solution restores integer feasibility. The complexity in this class of problems lies in optimizing a cost function that involves identical cost coefficients for competing integer decision variables. This causes many integer feasible solutions to lie in close proximity (with regard to the direction of the objective function) of each other. In the absence of any knowledge about the tree structure, a standard MIP processor cannot intelligently discriminate/prioritize among the candidate integer solutions. One possible approach for dealing with such symmetry issues is to reformulate the problem with many auxiliary variables (Barnhart, Johnson, Nemhauser, Savelsbergh & Vance 1998).

The results of using the standard MIP solver of OSL on the SIZES test set are presented in Table 2. As evident, the performance is quite poor. The MIP preprocessor in OSL could not recognize the tree structure present in the model, hence, had to resort to default, general-purpose branch and node selection strategies to resolve this model. It is clear that, although the problems are of modest size (no more than 110 binary variables), these are not directly amenable to standard integer programming techniques, and one must rely on exploiting problem structure.

Problem	CPU s	Nodes	Gap
SIZES3	6,000	20,000	5%
SIZES5	10,000	20,000	6%
SIZES10	18,000	20,000	8%

Table 2: SIZES: Initial Computations

3.3 Improvement Strategies

To exploit the stochastic structure of SIZES, we next utilized the OSLSE solver, as described in Section 2, along with several preprocessing/tuning strategies. The flexibility offered by OSLSE in manipulating the stochastic structures allowed for the effective implementation of these strategies, which would otherwise have been quite difficult to implement. These improvement strategies are described next.

Coefficient strengthening:

The big-M coefficient in (2) *can be tightened to* $M_{its} = \max_{z_{its}=1} \{y_{its} \mid y_{its} \text{ satisfies } (2) - (7)\}$.

The computation of M_{its} is carried out by solving a series of related stochastic linear programs and can be carried out quite efficiently using the OSLSE stochastic LP solver. Tightening of the $M's$ in (2) resulted in significantly improved lower bounds.

Branching priorities

For branching, the variables z_{its} corresponding to the $s \in \text{argmax}_s\{p_s\}$ should be considered first.

This is motivated by the fact that we should focus on the highly probable scenarios first as feasible solutions since these provide useful insights to the overall solution process.

For a given scenario s, the sets of variables z_{its} corresponding to the earlier time stages should be branched first.

This is motivated by the fact that the values of the variables appearing in the earlier time periods affect the feasibility of the later ones, hence drastically cut down the size of the solution space to search through.

For a given scenario s, and stage t, the variable z_{its} corresponding to $i \in \text{argmin}_i\ \{\alpha_i M_{its}\}$ should be branched first.

This is a greedy heuristic motivated purely by cost considerations.

Branching on number of set-ups

For a given scenario s and stage t, the aggregate variable $\sum_i z_{its}$ should be branched on.

At a given node n of the scenario tree, indexed by the scenario s and time stage t, the cost contribution of the $\{z_{its},\ i \in I\}$ variables is $\beta \sum_i z_{its}$. Hence, it is possible that we may branch on any variable of this set with very little improvement in the lower bound. In order to prevent this, we introduce the aggregate integer variable $v_{ts} = \sum_i z_{its}$, defined as the total number of set-ups at node n, and branch on this variable to obtain better lower bounds at various nodes of the branch and bound tree.

Branching directions

For a candidate branching variable z_{its}, the down-pseudocosts should be set higher than the up-pseudocosts.

OSLSE always uses a depth-first approach to find the first integer feasible solution. Since branching upwards always results in a feasible (partial or full integer) solution for the SIZES instances, there

is no incentive in doing so once we have found a feasible integer solution. Therefore, setting the pseudocosts in this prescribed manner favors either finding better solutions or hitting infeasible branch-and-bound nodes more frequently. This results in a faster convergence.

Changing branching direction

For the SIZES instances, switch to a depth-first strategy if no improvement is observed for 1000 branch-and-bound nodes.

This is motivated by the observation that the integer solutions are found very easily (since rounding up yields feasible solutions) and the improvements are observed very frequently (since the integer solutions lie very close to each other). Therefore, when no improvement is found for, say, a thousand nodes, then it is reasonable to assume that the solver is now trying to prove optimality by exploring the open nodes. At this stage, while using OSLSE, it is much faster to prove optimality by using a depth-first strategy which makes the solver algorithm zoom through the path leading to a leaf of the branch-and-bound tree.

3.4 Improved Results

With the proposed strategies, the computational performance was significantly improved. In Table 3, we compare the computational performance of the proposed method with the best known performance obtained using a specialized branch and bound algorithm (Ahmed et al. 2000). In all cases, the solution quality was as good as or better than the best known results. Furthermore, the number of branch and bound nodes required for the two larger problems were smaller in case of OSLSE. The solution times are not directly comparable owing to the difference in the computing platforms. However, considering the fact that on the LINPACK benchmark problems (Dongarra 2002) the CPU in the RS 6000 43P workstation used by Ahmed et al. (2000) is at least 3 times faster than the Pentium II processor in the IBM Thinkpad 770 used in our computations, it appears that our computational times are superior.

Problem	OSLSE			Ahmed et al. (2000)		
	CPU[†] s	Nodes	Gap	CPU[‡] s	Nodes	Gap
SIZES3	1,836	5,894	0%	70.7	1885	0%
SIZES5	5,200	10,321	0%	7,829	108,782	0%
SIZES10	12,000	15,000	0.11%	10,000	36,700	0.21%

[†] IBM Thinkpad 770

[‡] IBM RS 6000 Model 43P

Table 3: SIZES: OSLSE Performance

4. Test Set II: SCAP

4.1 Problem Description

Our second problem set is a class of stochastic multi-period multi-facility capacity expansion problem (SCAP) defined over scenario trees. This problem consists of determining the capacity expansion schedule of multiple facilities under fixed-charge expansion costs, and cost and demand uncertainty. A mathematical formulation of the problem, as proposed by Ahmed et al. (2002), is as follows:

$$\min \quad \sum_{n\in\mathcal{T}} p_n \left[\sum_{i\in\mathcal{I}} (\alpha_{in}x_{in} + \beta_{in}y_{in})\right] \tag{8}$$

$$\text{s.t.} \quad 0 \le x_{in} \le M_{in}y_{in} \qquad n \in \mathcal{T};\ i \in \mathcal{I} \tag{9}$$

$$\sum_{m\in\mathcal{P}(n)} \sum_{i\in\mathcal{I}} x_{im} \ge d_n \qquad n \in \mathcal{T} \tag{10}$$

$$y_{in} \in \{0,1\} \qquad n \in \mathcal{T};\ i \in \mathcal{I}. \tag{11}$$

In the above formulation, the realizations of uncertain cost $(\alpha_{in}, \beta_{in})$ and demand (d_n) parameters are modelled in the form of a scenario tree $\mathcal{T}$ with T stages. A node n in stage (or level) t of the tree constitutes the state of the world that can be distinguished by information available up to time stage t. The probability associated with the state of the world in node n is p_n. The path from the root node to a node n will be denoted by $\mathcal{P}(n)$. If n is a terminal (leaf) node then $\mathcal{P}(n)$ corresponds to a *scenario*, and represents a joint realization of the problems parameters over all periods $1, \ldots, T$. There are S leaf nodes corresponding to S scenarios. The non-negative variable x_{in} corresponds to the level of capacity expansion of facility i in node n, and the binary variable y_{in} corresponds to the yes-no decision of capacity expansion. Constraint (9) enforces the dependence of capacity expansion level to the expansion decision, and constraint (10) ensures the capacity available (cumulative expansion) at node n is at least as much as the demand. A complete description of the formulation appears in Ahmed et al. (2002).

We consider four problem instances of (SCAP). The dimensions of these instances are presented in Table 4. The table presents the problem name, number of time stages, number of nodes in the scenario tree, number of scenarios, number of facilities, number of binary variables, number of continuous variables, and the number of rows, in each of the four problem instances.

No.	T	$\lvert\mathcal{T}(0)\rvert$	S	$\lvert\mathcal{I}\rvert$	Bin.	Cont.	Rows
P_5_1	5	121	81	1	121	121	242
P_5_2	5	121	81	2	242	242	363
P_5_3	5	121	81	3	363	363	484
P_5_4	5	121	81	4	484	484	605

Table 4: SCAP: Problem Dimensions

4.2 Initial Computations

As before, we first attempted to solve the problem instances using the standard MIP solver in OSL v.3. The results of these runs are presented in Table 5. As can be observed that, three of the four problem instances could not be solved to optimality even after thousands of branch-and-bound nodes and CPU seconds.

Problem	CPU s	Nodes	Gap
P_5_1	48	869	0%
P_5_2	1000	8000	1%
P_5_3	1786	8000	3%
P_5_4	4000	8000	5%

Table 5: SCAP: Initial Computations

4.3 Improvement Strategies

In this subsection, we discuss four tree preprocessing strategies implemented using the OSLSE solver that helped us resolving the SCAP instances effectively.

SOS Type 1 Branching

This branching strategy is based on the following result which is proved in the appendix.

Proposition 1 *If all cost components are strictly positive, at any node n of the scenario tree, at most one facility may be expanded, i.e., $\sum_{i \in I} y_{in} \leq 1$.*

This result is very powerful as it forms the basis of a strong branching strategy (based on SOS Type 1 sets) which cuts down the number of branch-and-bound nodes to be explored significantly.

Variable Fixing

At node n of the scenario tree, if $(\alpha_{in}, \beta_{in}) \geq (\alpha_{jn}, \beta_{jn})$, then facility j dominates facility i and therefore facility j must be expanded before facility i can be expanded, i.e, $y_{jn} \geq y_{in}$.

This indicates that, at node n, it is more expensive to open or increase capacity at facility i than it is at facility j. This result can be combined with the result of Proposition 1 to fix y_{in} at 0. This result greatly reduces the number of free integer variables to be optimized.

Coefficient Strengthening

The big-M coefficient M_{in} in (9) *can be tightened to the residual demand for the subtree originating at node n, expressed as* $(\max_{m \in \mathcal{T}(n)}\{D_m\} - D_{a(n)})$*, where $a(n)$ is the ancestor of node n and*

$D_m = \sum_{n \in \mathcal{P}(m)} d_n$.

This improves the LP relaxation considerably by forcing the fractional values for the binary variables closer to 1. This is an important preprocessing step that exploits the knowledge of the underlying scenario tree structure to tighten the coefficients. It is noteworthy here that traditional coefficient strengthening methods fail to come off with such tight coefficients as they tend to look at only a small subset of the constraints for such analysis.

Heuristic Branching

At node n of the scenario tree let $l = \text{argmin}_{i \in I}\{\alpha_{in} d_n + \beta_{in}\}$. Then the binary variable y_{ln} has the highest branching priority.

This rationale is motivated by local (at any node) cost considerations and it reinforces the result of Proposition 1 to serve as an effective heuristic branching rule.

4.4 Improved Computations

With the proposed strategies, the performance, improved significantly. In Table 6 we compare the performance of the proposed method against the specialized branch and bound algorithm of Ahmed et al. (2002). In both cases, all four instances were solved to within an absolute optimality gap of 10^{-5}. However, the number of branch and bound required by OSLSE is significantly smaller than those in Ahmed et al. (2002). Although the computational platforms are different, the LINPACK benchmark (Dongarra 2002) indicates that the CPU in the RS 6000 590 workstation used by Ahmed et al. (2002) is at least 4 times faster than the Pentium II processor in the IBM Thinkpad 770 used in our computations. This suggests that the OSLSE computational times are significantly smaller.

Problem	OSLSE		Ahmed et al. (2002)	
	CPU† s	Nodes	CPU‡ s	Nodes
P_5_1	0.56	0	5.9	193
P_5_2	5.05	10	16.94	492
P_5_3	19.51	56	90.15	2142
P_5_4	230.11	1486	271.91	4728

† IBM Thinkpad 770

‡ IBM RS 6000 Model 590

Table 6: SCAP: Improved Computations

5. Test Set III: SEMI

5.1 Problem Description

Our third problem set is a class of multi-stage stochastic integer programs corresponding to optimal tool-planning in a semiconductor fabrication facility. This multi-period problem consists of

scheduling the purchase of fabrication machine tools and the allocation of tool capacity in order to meet the demand of various families of semiconductor products. Together with the high capital cost of semiconductor machines, the uncertainty surrounding demand growth, technology obsolescence and procurement lead times, makes tool purchasing decisions extremely complex. Barahona, Bermon, Günlük & Hood (2001) developed the following multi-stage stochastic integer programming formulation for the problem.

$$\min \qquad \sum_{s} \pi_s \sum_{p,t} U_{s,p,t} + \sum_{i,t} q_i N_{i,t}, \tag{12}$$

s.t.

$$\gamma_{p,t} W_{s,p,t} + U_{s,p,t} = d_{s,p,t} \qquad \forall\ s,\ p,\ t \tag{13}$$

$$\sum_{p} b_{j,p,t} W_{s,p,t} = \sum_{i \in I(j)} O_{s,j,i,t}, \qquad \forall\ s,\ j,\ t \tag{14}$$

$$\mu_{i,t} + c_{i,t} \sum_{k=1}^{t} N_{i,k} - \sum_{j \in J(i)} h_{i,j,t} O_{s,j,i,t} \geq 0 \qquad \forall\ s,\ i,\ t \tag{15}$$

$$\sum_{i} m_{i,t} N_{i,t} \leq \beta_t \qquad \forall\ t \tag{16}$$

$$U_{s,p,t} \leq \alpha_{s,p,t} \qquad \forall\ s,\ p,\ t \tag{17}$$

$$U_{s,p,t},\ W_{s,p,t},\ O_{s,j,i,t} \geq 0 \qquad \forall\ s,\ p,\ j,\ i,\ t \tag{18}$$

$$N_{i,t} \in \mathbb{Z}_+ \qquad \forall\ i,\ t. \tag{19}$$

In the above formulation, the indices i, j, p, s, t correspond to machine tool groups, processing operations, product families, scenarios and time periods, respectively. For a given time period t and scenario s, the problem variables $U_{s,p,t}$ correspond to the quantity of unmet demand of product p; $W_{s,p,t}$ correspond to the production of product p; and $O_{s,j,i,t}$ correspond to the volume of operations j carried out on machine i. The variables $N_{i,t}$ correspond to the number of tools of tool group i to purchase in period t. The model parameters π_s correspond to the probability of scenario s; q_i correspond to a penalty for buying a tool of tool group i; $\gamma_{p,t}$ is the yield factor for the manufacturing process for product p in period t; $d_{s,p,t}$ correspond to the demand for product p under scenario s in period t; $b_{j,p,t}$ is the required number of passes of operation j for product p in period t; $\mu_{i,t}$ is the initial capacity in hours/day for tool group i in period t; $c_{i,t}$ is the additional capacity for an additional tool in hours/day for tool group i in period t; $h_{i,j,t}$ corresponds to the processing requirement (in hours) for operation j on tool group i in period t; $m_{i,t}$ is the cost of a tool in tool group i bought for period t; β_t is the budget available in period t; and $\alpha_{s,p,t}$ is an upper bound for the unmet demand for product p under scenario s in period t. The index sets $I(j)$ correspond to the set of tool groups capable of operation j, and $J(i)$ corresponds to the set of operations that can be carried out on tool group i. The objective function (12) is to minimize the expected unmet

demand and machine tool penalty; constraint (13) equates the production and shortage to the demand; constraint (14) relates the processing requirements to the production volumes; constraint (15) enforces that the required processing capacity should be less than the installed tool capacity; constraint (16) corresponds to the budget condition; constraint (17) enforces an upper bound on the unmet demand; and finally, constraints (18) and (19) enforces the non-negativity and integrality of the problem variables, respectively. Further details of the above formulation is described in Barahona et al. (2001).

We consider three instances of SEMI problem set. These consist of 2, 3, and 4 scenarios respectively. The sizes of the deterministic equivalent problems for these instances are presented in Table 7. Note that these MIPs involve general integer variables.

Problem	Scenarios	Columns	Integer variables	Rows
SEMI2	2	20216	612	11686
SEMI3	3	30018	612	17528
SEMI4	4	39820	612	23370

Table 7: SEMI: Problem Dimensions

5.1 Initial Computations

Table 8 reports on a straightforward application of OSL version 3.0 MIP solver on the SEMI instances. The LP relaxation bounds for these problems are extremely weak. For lower bounding we use the expected value of the wait-and-see solutions (Birge & Louveaux 1997), i.e., the probability weighted sum of the optimal objective values of the subproblems corresponding to the individual scenarios. The wait-and-see bound proved to be stronger than the LP relaxation bound and could not be improved in the course of the branch and bound algorithm. As is evident from Table 8, the performance is quite disappointing. Thousands of branch-and-bound nodes and CPU seconds were spent before solutions of acceptable quality could be found. Note that, owing to the poor quality of the lower bound, we believe that the optimality gap presented in Table 8 is a gross over-estimate of the true optimality gap for the reported solutions.

Problem	UB	LB	% Gap	CPU s	Nodes	Iterations
SEMI2	1629.0	1280.0	27.3	23,043	99,129	1,874,335
SEMI3	1814.3	1457.0	24.5	30,204	91,747	1,653,069
SEMI4	2203.1	1748.0	26.0	666,142	116,730	3,929,277

Table 8: SEMI: Initial Computations using the OSL Version 3.0 MIP Solver

5.2 Improvement Strategies

In solving the SEMI problems using the OSL v.3 MIP branch-and-cut solver, we observed that much of the computational effort is spent early on in finding poor quality integer feasible solutions that are in close proximity of each other. Prior experience indicates that for integer programs with many feasible integer solutions with similar objective function values, computational efficiencies may be achieved by increasing the size of the branching grid. Consequently, we used a larger-sized branching grid during the initial phase of the branch-and-bound search, and gradually decreased the grid size towards the end. The OSL MIP solver allows control of the branching grid through the parameter `Rimprove`. Unfortunately, when this strategy was implemented within the standard OSL MIP solver framework, the desired improvement was not observed.

Next, we tried the OSLSE framework of Section 2 along with the above branching strategy. As the results in the Section 5.3 indicate, the computational performance improved significantly. The difference in the role of the control parameter `Rimprove` on the performance of the OSL MIP solver and the OSLSE solver can be explained as follows. In its branch-and-cut algorithm, OSL uses the value of `Rimprove` for tightening the objective function cutoff value (for effective pruning of the tree) and for probing (to fix some of the integer variables). Probing is often accompanied by building and exploiting the logical relationships tabulated in implication lists. This is accomplished by processing linear programs underlying the subproblems corresponding to the nodes of the branch-and-bound tree. Thus, for branch-and-bound nodes associated with larger linear programs, the probing process could and does consume too much computational resources to justify the potential benefits. Therefore, additional logic has been implemented in commercial codes like OSL to automatically turn off such probing heuristics when certain threshold criteria are met. Consequently, owing to the large LP relaxations in case of standard MIP techniques for stochastic integer programs, the probing heuristics are ineffective. On the other hand, in the OSLSE framework, the original LP is decomposed into smaller subproblems by exploiting the underlying scenario tree structure, thereby allowing efficient probing.

5.3 Improved Results

The performance of the OSLSE framework along with the specialized branching process outlined above, are presented in Table 9. As is evident, better solutions than those obtained using the OSL MIP solver, were found using the proposed approach. Furthermore, the successful use of the `Rimprove` control parameter resulted in significantly improved lower bounds, and therefore the estimated optimality gap. More importantly, the better solutions and lower bound estimates were obtained at significantly smaller computational effort.

Problem	UB	LB	% Gap	CPU s	Nodes	Iterations
SEMI2	1545.5	1457.1	6.1	2,164	7,153	93,118
SEMI3	1808.1	1745.4	3.6	8,914	28,410	383,787
SEMI4	2197.2	2148.4	2.3	27,519	61,192	1,125,986

Table 9: SEMI: Improved Computations using OSLSE

6. Test Set IV: DCAP

6.1 Problem Description

Our final problem set consists of a class of stochastic dynamic capacity acquisition and assignment problems. Given a set of m resources and n tasks with stochastic processing requirements, the dynamic resource acquisition and assignment problem seeks a minimum expected cost schedule of capacity acquisitions for the resources and the assignment of resources to tasks, over a given planning horizon of T periods. Using a finite set of scenarios to model uncertain problem parameters, Ahmed & Garcia (2002) proposed the following two-stage stochastic integer programming formulation for the problem:

$$\min \quad \sum_{t=1}^{T}\sum_{i=1}^{m} f_{it}(x_{it}) + \sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{i=0}^{m}\sum_{j=1}^{n} p^s c^s_{ijt} y^s_{ijt} \tag{20}$$

$$\text{s.t.} \quad x \in X \subseteq \mathbb{R}^{mT} \tag{21}$$

$$\sum_{j=1}^{n} d^s_{jt} y^s_{ijt} \leq \sum_{\tau=1}^{t} x_{i\tau} \quad \forall\ i,\ t,\ s \tag{22}$$

$$\sum_{i=0}^{m} y^s_{ijt} = 1 \qquad \forall\ j,\ t,\ s \tag{23}$$

$$y^s_{ijt} \in \{0,1\} \qquad \forall\ i,\ j,\ t,\ s. \tag{24}$$

In the above formulation x_{it} denotes the capacity acquisition of resource i in period t and y^s_{ijt} denotes the $0-1$ decision of assigning resource i to task j in period t under scenario s. The function $f_{it}(\cdot)$ is the expansion cost for resource i in period t, c^s_{ijt} is the cost of assigning resource i to task j in period t under scenario s, d^s_{jt} is the processing requirement of task j in period t under scenario s, and p^s is the probability of scenario s. Typically the capacity expansion costs are modelled as fixed-charge cost functions, i.e.

$$f_{it}(x_{it}) := \alpha_{it} x_{it} + \beta_{it} u_{it},$$

with

$$u_{it} = \begin{cases} 1 & \text{if } \ x_{it} > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where α_{it} and β_{it} are the variable and fixed cost, respectively, of capacity acquisition of resource i in period t, and u_{it} is the indicator variable for capacity acquisition. The set X denotes the constraints on capacity acquisitions. The second constraint reflects that the processing requirement of all tasks assigned to a resource in any period cannot exceed the installed capacity up to that period. The third constraint enforces that each task needs to be assigned to exactly one resource in each of the periods. The final constraint enforces the binary restrictions on the assignment variables y^s_{ijt}. Note that a dummy resource $i = 0$ with infinite capacity is included, so that the assignment problem is always feasible. The cost c^s_{0jt} denotes the penalty of failing to assign a resource to task j. The dummy resource enforces the *complete recourse* property (Birge & Louveaux 1997), which guarantees that there is a feasible second-stage assignment in all periods and all scenarios for any capacity acquisition schedule.

We considered 12 instances of the above problem. The number of resources, tasks, time periods, and scenarios, as well as the size of the associated deterministic equivalent problems for these instances are presented in Table 10.

Name	m	n	T	S	Rows	Columns	Binaries	Non-zeroes
DCAP_233_200	2	3	3	200	3012	5418	5406	10230
DCAP_233_300	2	3	3	300	4512	8118	8106	15330
DCAP_233_500	2	3	3	500	7512	13518	13506	25530
DCAP_243_200	2	4	3	200	3612	7218	7206	13230
DCAP_243_300	2	4	3	300	5412	10818	10806	19830
DCAP_243_500	2	4	3	500	9012	18018	18006	33030
DCAP_332_200	3	3	2	200	2412	4818	4806	9627
DCAP_332_300	3	3	2	300	3612	7218	7206	14427
DCAP_332_500	3	3	2	500	6012	12018	12006	24027
DCAP_342_200	3	4	2	200	2812	6418	6406	12427
DCAP_342_300	3	3	2	300	4212	9618	9606	18627
DCAP_342_500	3	3	2	500	7012	16018	16006	31027

Table 10: DCAP: Problem Dimensions

6.2 Initial Computations

Table 11 reports on a straightforward application of OSL version 3.0 MIP solver on the DCAP instances. The computations were carried out on an IBM Thinkpad T-20 (700 MHz), with a relative optimality gap tolerance of 1% and a CPU limit of 1800 seconds. As evident from the table, the performance is quite unsatisfactory. None of the problem instances could be solved to the desired optimality tolerance within the allotted time. Furthermore, a huge number of branch and bound nodes are required.

Problem	UB	LB	% Gap	CPU s	Nodes	Iterations
DCAP_233_200	1884.685	1800.574	4.67	1800.0	103,334	3,987,344
DCAP_233_300	1698.573	1583.473	7.26	1800.0	35,567	1,899,475
DCAP_233_500	1798.215	1699.258	5.82	1800.0	58,778	932,053
DCAP_243_200	2393.632	2280.644	4.96	1800.0	230,887	722,349
DCAP_243_300	2668.944	2502.958	6.63	1800.0	102,220	340,388
DCAP_243_500	2226.787	2100.733	6.00	1800.0	30,772	665,328
DCAP_332_200	1093.295	1013.449	7.90	1800.0	178,993	522,855
DCAP_332_300	1314.356	1206.386	8.95	1800.0	104,553	449,840
DCAP_332_500	1619.678	1510.376	7.22	1800.0	50,336	879,477
DCAP_342_200	1663.054	1583.435	5.05	1800.0	190,554	537,921
DCAP_342_300	2131.889	2001.633	6.50	1800.0	82,334	643,458
DCAP_342_500	1979.445	1859.971	6.45	1800.0	20,765	946,033

Table 11: DCAP: Initial Computations using the OSL Version 3.0 MIP Solver

6.3 Improved Results

Table 12 displays the performance of the OSLSE framework of Section 2. In all but one case, the desired optimality tolerance was achieved at the pre-processing stage. Since the OSLSE framework makes the underlying tree structure transparent to the MIP pre-processor, significant tightening of the formulation is possible.

Problem	UB	LB	% Gap	CPU s	Nodes	Iterations
DCAP_233_200	1834.695	1829.595	0.27	109.23	0	15,333
DCAP_233_300	1648.473	1643.473	0.3	194.81	0	20,478
DCAP_233_500	1741.218	1736.218	0.2	378.12	0	32,050
DCAP_243_200	2330.639	2320.639	0.4	57.23	0	22,342
DCAP_243_300	2566.955	2556.955	0.4	467.02	220	40,350
DCAP_243_500	2176.736	2166.736	0.4	666.31	0	65,313
DCAP_332_200	1063.399	1053.399	0.95	44.34	0	22,828
DCAP_332_300	1254.345	1244.345	0.8	162.35	0	49,814
DCAP_332_500	1589.711	1579.711	0.63	616.33	0	87,401
DCAP_342_200	1623.077	1613.077	0.61	103.23	0	37,939
DCAP_342_300	2071.686	2061.686	0.48	239.30	0	64,490
DCAP_342_500	1909.191	1899.191	0.52	1049.34	0	146,067

Table 12: DCAP: Improved Computations using OSLSE

7. Concluding Remarks

Stochastic integer programs remain among the most formidable problems in mathematical programming. Although progress in developing specialized algorithms is continually being made, a great number of issues remain open before these algorithmic ideas can be implemented in software

for solving general purpose SIPs. On the other hand, much has been accomplished in the way of software packages for solving *deterministic* integer programs as well as stochastic *linear* programs. In this paper, we showed how the modelling infrastructure afforded by stochastic linear programming solvers can be integrated with commercial MIP software packages to exploit the knowledge of the scenario tree structure underlying stochastic MIPs. We have demonstrated that the proposed ideas offer significant computational advantage when applied to problem sets from the literature.

Appendix

Proof of Proposition 1

By way of contradiction, assume that there exists an optimal solution to SCAP in which there are two facilities to be expanded at a node n. Let j and k be the two facilities that need to be expanded at node n in the optimal solution. Thus, we have $y_{jn} = 1$ and $y_{kn} = 1$ and the associated optimal production quantities x_{jn} and x_{kn}. Consider now the subproblem at node n:

$$\begin{aligned}
\min \quad & \sum_{i \in I} (\alpha_{in} x_{in} + \beta_{in} y_{in}) \\
\text{s.t.} \quad & 0 \leq x_{in} \leq M_{in} y_{in} && i \in I \\
& I_{a(n)} + \sum_{i \in I} x_{in} = d_n + I_n \\
& y_{in} \in \{0, 1\} && i \in I
\end{aligned}$$

where $I_{a(n)}$ is the capacity inherited from the ancestor node $a(n)$ and I_n is the capacity in node n corresponding to the optimal solution. Since it is optimal in the above problem to expand both facilities j and k, and since the upper bounds on the expansions M_{in} are sufficiently high, we must have

$$\alpha_{jn} x_{jn} + \beta_{jn} + \alpha_{kn} x_{kn} + \beta_{kn} \leq \alpha_{jn}(x_{jn} + x_{kn}) + \beta_{jn}$$

and

$$\alpha_{jn} x_{jn} + \beta_{jn} + \alpha_{kn} x_{kn} + \beta_{kn} \leq \alpha_{kn}(x_{jn} + x_{kn}) + \beta_{kn},$$

i.e. the cost of expanding both k and j is less than expanding either facility by the total amount. The above inequalities simplify to

$$\beta_{kn} \leq (\alpha_{jn} - \alpha_{kn}) x_{kn}, \tag{25}$$

and

$$\beta_{jn} \leq (\alpha_{kn} - \alpha_{jn}) x_{jn}. \tag{26}$$

If $\alpha_{jn} \leq \alpha_{kn}$, (25) implies $\beta_{kn} \leq 0$, and if $\alpha_{kn} \leq \alpha_{jn}$, (26) implies $\beta_{kn} \leq 0$. However since all cost components are strictly positive we have a contradiction. Hence, no optimal solution to SCAP may have more than one facility to be expanded at any node of the scenario tree. □

Acknowledgement

S. Ahmed's research has been supported in part by The Logistics Institute Asia-Pacific in Singapore and the National Science Foundation under Grant DMII-0099726.

References

Ahmed, S., & Garcia, R. (2002). Dynamic capacity acquisition and assignment under uncertainty. Technical Report. School of Industrial & Systems Engineering, Georgia Institute of Technology. (2002).

Ahmed, S., Parija, G. & King, A. J. (2002). A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. To appear in *The Journal of Global Optimization*.

Ahmed, S., Tawarmalani, M. & Sahinidis, N. V. (2000). A finite branch and bound algorithm for two-stage stochastic integer programs. Stochastic Programming E-Print Series, available at `http://dochost.rz.hu-berlin.de/speps/`.

Barahona, F., Bermon, S., Günlük, O. & Hood, S. (2001). Robust capacity planning in semiconductor manufacturing. *Optimization on-line eprint,* `http://www.optimization-online.org`.

Barnhart, C., Johnson, E., Nemhauser, G., Savlesbergh, M.W.P. & Vance, P. (1998). Branch-and-Price: Column generation for solving huge integer programs. *Operations Research*, Vol. 46, pp. 316–329.

Bienstock, D. & Shapiro, J. F. (1988). Optimizing resource acquisition decisions by stochastic programming. *Management Science*, Vol. 34, pp. 215–229.

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

Birge, J. R. & Dempster, M. A. H. (1996). Stochastic programming approaches to stochastic scheduling. *Journal of Global Optimization*, Vol. 9,(3-4) pp. 417–451.

Birge, J. R. & Louveaux, F. (1997). *Introduction to Stochastic Programming*. Springer, New York, NY.

Birge, J. R., Donohue, C. J., Holmes, D. F. & Svintsitski, O. G. (1996). A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs. *Mathematical Programming*, Vol. 75,(2) pp. 327–352.

Bitran, G. R., Haas, E. A. & Matsuo, H. (1986). Production planning of style goods with high setup costs and forecast revisions. *Operations Research*, Vol. 34, pp. 226–236.

Carøe, C. C. & Schultz, R. (1999). Dual decomposition in stochastic integer programming. *Operations Research Letters*, Vol. 24, pp. 37–45.

Carøe, C. C. & Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, Vol. 83, pp. 451–464.

Carøe, C. C., Ruszczyński, A. & Schultz, R. (1997). Unit commitment under uncertainty via two-stage stochastic programming. In *Proceedings of NOAS 97,* Carøe *et al.* (eds.), Department of Computer Science, University of Copenhagen, pp. 21–30.

Dash Assoc. (1999). *XPRESS-MP: Extended Modeliing and Optimisation Subroutine Library,* Release 11.

Dempster, M. A. H. (1982). A stochastic approach to hierarchical planning and scheduling. In *Deterministic and Stochastic Scheduling,* Dempster *et al.*(eds.), D. Riedel Publishing Co., Dordrecht, pp. 271–296.

Dempster, M. A. H., Fisher, M. L., Jansen, L., Lageweg, B. J., Lenstra, J. K. & Rinnooy Kan, A. H. G. R. (1981). Analytical evaluation of hierarchical planning systems. *Operations Research*, Vol. 29, pp. 707–716.

Dert, C. L. (1995). *Asset Liability Management for Pension Funds, A Multistage Chance Constrained Programming Approach*. PhD thesis. Erasmus University, Rotterdam, The Netherlands.

Dongarra, J. J. (2002). Performance of various computers using standard linear equations sofwtare. Technical Report, University of Tennessee.
URL `http://www.netlib.org/benchmark/performance.ps`.

Gassmann, H. (1990). MSLIP: A computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, Vol. 47, pp. 407–423.

Higle, J. L. & Sen, S. (1996). *Stochastic Decomposition: A Statistical Method for Large Scale Stochastic Linear Programming*. Kluwer Academic Publishers, Dordrecht.

IBM (1991). *Optimization Subroutine Library Guide and Reference. Release 2*. International Business Machines Corporation. Kingston, NY.

IBM (1998). *Optimization Library Stochastic Extensions Guide and Reference*. `http://service2.boulder.ibm.com/es/oslv2/features/Stoch Ext/stochexu.htm`.

ILOG Inc. (1997). *CPLEX 6.0 User's Manual*. Incline Village, NV.

Jorjani, S., Scott, C. H. & Woodruff, D. L. (1999). Selection of an optimal subset of sizes. *International Journal of Production Research*, Vol. 37, pp. 3697–3710.

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

King, A. J. & Wright, S. E. (2001). A flexible-partition, nested L-shaped method for linear programming. Working paper, IBM T.J. Watson Research Center.

Laporte, G. & Louveaux, F. V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, Vol. 13, pp. 133–142.

Laporte, G., Louveaux, F. V. & Mercure, H. (1989). Models and exact solutions for a class of stochastic location-routing problems. *European Journal of Operational Research*, Vol. 39, pp. 71–78.

Laporte, G., Louveaux, F. V. & Mercure, H. (1992). The vehicle routing problem with stochastic travel times. *Transportation Science*, Vol. 26, pp. 161–170.

Laporte, G., Louveaux, F. V. & van Hamme, L. (1994). Exact solution of a stochastic location problem by an integer L-shaped algorithm. *Transportation Science*, Vol. 28,(2) pp. 95–103.

Lenstra, J. K., Rinnooy Kan, A. H. G. & Stougie, L. (1983). A framework for the probabilistic analysis of hierarchical planning systems. *Technical report*. Mathematisch Centrum, University of Amsterdam.

Norkin, V. I., Ermoliev, Y. M. & Ruszczyński, A. (1998). On optimal allocation of indivisibles under uncertainty. *Operations Research*, Vol. 46, pp. 381–395.

Rockafellar, R. T. & Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, Vol. 16,(1) pp. 119–147.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, Vol. 35, pp. 309–333.

Schultz, R., Stougie, L. & van der Vlerk, M. H. (1996). Two-stage stochastic integer programming: A survey. *Statistica Neerlandica. Journal of the Netherlands Society for Statistics and Operations Research*, Vol. 50,(3) pp. 404–416.

Spaccamela, A. M., Rinnooy Kan, A. H. G. & Stougie, L. (1984). Hierarchical vehicle routing problems. *Networks*, Vol. 14, pp. 571–586.

Stougie, L. & van der Vlerk, M. H. (1997). Stochastic integer programming. In *Annotated Bibliographies in Combinatorial Optimization,* M. Dell'Amico *et al.* (eds), John Wiley & Sons, New York, pp. 127–141.

Takriti, S., Birge, J. R. & Long, E. (1996). A stochastic model of the unit commitment problem. *IEEE Transactions on Power Systems*, Vol. 11, pp. 1497–1508.

Tayur, S. R., Thomas, R. R. & Natraj, N. R. (1995). An algebraic geometry algorithm for scheduling in the presence of setups and correlated demands. *Mathematical Programming*, Vol. 69,(3) pp. 369–401.

van der Vlerk, M. H. (1995). *Stochastic programming with integer recourse*. PhD thesis. University of Groningen, The Netherlands.

Van Slyke, R. & Wets, R. J.-B. (1969). L-Shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, Vol. 17, pp. 638–663.