International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

Multi-item capacitated lot-sizing with demand uncertainty

Paolo Brandimarte

To cite this article: Paolo Brandimarte (2006) Multi-item capacitated lot-sizing with demand uncertainty, International Journal of Production Research, 44:15, 2997-3022, DOI: 10.1080/00207540500435116

To link to this article: https://doi.org/10.1080/00207540500435116

International Journal of Production Research,
Vol. 44, No. 15, 1 August 2006, 2997–3022

Taylor & Francis
Taylor & Francis Group

Multi-item capacitated lot-sizing with demand uncertainty

PAOLO BRANDIMARTE*

Dipartimento di Sistemi di Produzione ed Economia dell'Azienda,
Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

(*Revision received November 2004*)

We consider a stochastic version of the classical multi-item Capacitated Lot-Sizing Problem (CLSP). Demand uncertainty is explicitly modeled through a scenario tree, resulting in a multi-stage mixed-integer stochastic programming model with recourse. We propose a plant-location-based model formulation and a heuristic solution approach based on a fix-and-relax strategy. We report computational experiments to assess not only the viability of the heuristic, but also the advantage (if any) of the stochastic programming model with respect to the considerably simpler deterministic model based on expected value of demand. To this aim we use a simulation architecture, whereby the production plan obtained from the optimization models is applied in a realistic rolling horizon framework, allowing for out-of-sample scenarios and errors in the model of demand uncertainty. We also experiment with different approaches to generate the scenario tree. The results suggest that there is an interplay between different managerial levers to hedge demand uncertainty, i.e. reactive capacity buffers and safety stocks. When there is enough reactive capacity, the ability of the stochastic model to build safety stocks is of little value. When capacity is tightly constrained and the impact of setup times is large, remarkable advantages are obtained by modeling uncertainty explicitly.

Keywords: Capacitated lot-sizing problem; Stochastic programming; Scenario generation

1. Background and motivation

One of the critical issues in MRP/ERP systems is the support in developing a good Master Production Schedule (MPS). In principle, Mixed-Integer Linear Programming (MILP) models, like variants of the multi-item capacitated lot-sizing problem (CLSP), could be helpful. In general, optimized production planning models could yield significant improvements in some situations. While such models have been around in the academic community for quite some time, their practical application is far from widespread. Among the reasons for this, the computational burden of solving a MILP model is probably the most commonly mentioned. However, the increase in computing power at affordable prices and the astonishing progress in commercial optimization software have been so remarkable that such models are now a viable alternative, at least if one accepts the idea of a somewhat suboptimal solution obtained by some heuristic truncation

*Email: paolo.brandimarte@polito.it

ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540500435116

of the branch-and-bound search tree. Indeed, many Advanced Planning and Scheduling (APS) modules within commercial ERP systems include some form of optimization algorithm (Stadtler and Kilger 2002).

Nevertheless, this does not mean that applying such models in practice is now easy. On the one hand, the need for organizational support in terms of high-quality data, education, and managerial commitment cannot be overemphasized. On the other hand, at least two issues still need to be properly addressed: demand uncertainty and the end-of-horizon effect. These are two facets of a general issue: we may solve, within a reasonable accuracy, an optimization model which is only a simplified representation of reality; but how does the 'optimal' solution fare in a real-world context?

For instance, it is well known that, in the case of uncapacitated single-item lot-sizing models with deterministic demand, the optimal solution is easily found by exploiting the Wagner–Whitin property (see, e.g., Stadtler (2000)). Yet, this 'optimal' solution, when applied within a rolling horizon framework, whereby new time buckets are added to the current planing horizon, can be outperformed by heuristic strategies. Some authors, such as Fisher *et al*. (2001), suggest ways to add a terminal value term in order to overcome the apparent myopic behaviour of the basic optimization model, which takes only a trajectory cost into account. The issue can be considered as a consequence of a modeling error: the basic optimization model assumes that the demand will be zero (the factory is closed down) beyond the planning horizon. It can be argued that, in a multi-item capacitated model, this issue is somehow mitigated by the lack of the Wagner–Whitin property, which makes the solution less 'extreme'. Still, the effect of changing demand patterns can also cause nervousness effects (Kazan *et al*. 2000), the cost of which may be difficult to quantify, but which may be disruptive in practice.

Neglecting demand uncertainty can also be considered a modeling error. We often substitute the uncertain demand by an expected value, possibly obtained by some forecasting procedure, but this modeling error can cause an increase in costs and, even worse, it can result in lost sales and unsatisfied customers. There is some literature on stochastic lot-sizing models and a review is given, for example, by Sox (1999). Many such models are variants of statistical inventory management strategies, which may be used to address supply chain management within a decentralized framework; however, they are not so useful for capacity constrained master production scheduling. Among the possible approaches to tackle the MPS problem, multistage stochastic programming (Birge and Louveaux 1997) is potentially interesting. However, we must face serious issues.

- Multi-stage stochastic programming is computationally intensive, even in the continuous case; adding the complexity of mixed-integer programming models such as CLSP does not help; so, we need strong formulations and mathematical programming-based heuristics.
- We need a parsimonious but effective way to represent uncertainty; this raises issues linked to scenario generation and sampling uncertainty.
- Last, but not least, a largely ignored issue is how well does a multi-stage planning approach fare in a real context. We need to simulate the behaviour of the planning system in out-of-sample scenarios within a rolling horizon framework.

Given such serious challenges, it is no surprise that many stochastic planning models are actually two stage (see, e.g., Bakir and Byrne (1998)). Basically, even if the problem is multi-period, it is assumed that the decisions for the overall planning horizon will not be adapted progressively as more and more information is collected. This is not plausible, and to actually model a dynamic decision-making process we need a truly multi-stage approach. An exception is Haugen *et al*. (2001), where a heuristic approach based on progressive hedging is applied (progressive hedging is basically a Lagrangian decomposition scheme based on solving and coordinating single scenario subproblems; see Kall and Wallace (1994)). However, they consider an uncapacitated single-item problem. A branch-and-cut approach is described for the single-item uncapacitated problem in Guan *et al*. (2004), where cutting planes for the deterministic problem are extended to the stochastic case. A branch-and-price approach is proposed in Lulli and Sen (2002) for a specific capacitated, but single-item, version of the batching problem. Here we consider a simple version of multi-item CLSP, which is certainly not a full-fledged model for master production scheduling, but it is closer to our aim.

While *ad hoc* heuristic approaches, such as adaptations of progressive hedging, may be valuable, it should be noted that exploiting commercially available solvers has some merit. On the one hand, LP-based heuristics, or truncated branch-and-bound schemes, may be more flexible than Lagrangian heuristics, which exploit specific problem structures (see, e.g., Maes *et al*. (1991)). Furthermore, using commercial solvers as building blocks allows us to leverage their remarkable continuous progress. Thus, in the following we will consider MILP models solved approximately by heuristics relying on pieces of commercial code (namely, ILOG CPLEX) as building blocks.

Within this framework, the aims and the plan of the paper are the following.

- In section 2 we propose a relatively strong formulation of the Stochastic CLSP (SCLSP). SCLSP is clearly a simplified version of a real-life problem, but it should provide a good starting point. The need for a strong formulation is justified by the application of LP-based solution methods which are sensitive to the integrality gap (Wolsey 1998).
- Since the model cannot be solved to optimality even for medium-sized problem instances, we propose in section 3 a solution strategy which is basically a version of the fix-and-relax approach of Dillenberger *et al*. (1994).
- Section 4 considers different scenario generation strategies. This is quite a thorny issue, as we need a very parsimonious way to model uncertainty. The standard approach is based on a scenario tree, which may easily grow beyond a tractable limit in the case of multi-stage problems. However, we should note that the real aim of the scenario tree is not really to represent uncertainty as faithfully as possible, but rather to generate a good and robust solution. We mean 'good' in a very limited but practical way: we are satisfied if the scenario tree is able to yield a production plan outperforming what we get by solving a deterministic CLSP based on expected values of demand.
- In section 5 we describe different computational experiments. One aim is to assess the quality of the heuristic solution approach and its computational burden. More important, we want to simulate the application of the proposed approach within a rolling horizon framework in order to compare

the performance of the stochastic planner with a considerably simpler planner based on an expected value model. In fact, we should not take for granted that a stochastic model works better than a simpler deterministic model. More so, when even our model of uncertainty is subject to approximation (the scenario tree is limited) and errors (the probability distribution and its parameters do not match those of the 'true' demand data process). So, we must also consider the robustness of the approach to such modeling errors, and this issue can only be addressed within a proper simulation architecture.

- Finally, section 6 is devoted to conclusions, critical remarks, and further research directions.

In this paper we do not address end inventory conditions. We believe that this is a severe limitation of this paper (and of most lot-sizing literature, by the way). However, our computational experience suggests that a stochastic formulation is *per se* able to cope with this issue better than a deterministic formulation. See section 6 for some considerations on this topic.

2. A plant-location-based SCLSP model formulation

Our mathematical formulation of Stochastic CLSP (SCLSP) stems from two starting points: a strong, plant location formulation of the deterministic model, and a tree-based representation of uncertainty. Many readers will be familiar with the plant location formulation, but we prefer to recall its basics both for the sake of readability and because it is necessary to have a graphical grasp of it in order to understand its stochastic extension.

It is well known that natural model formulations of CLSP are not practically solvable due to the large integrality gap. This means that the lower bounds obtained by relaxing the binary setup variables to continuous values and solving the resulting LP problem are very weak and strongly underestimate the true objective value at the optimal integer solution; this makes branch and bound very slow as many useless nodes are explored (Wolsey 1998). If we denote by $x_{it} \geq 0$ the amount of item i produced during time bucket t and by $s_{it} \in \{0, 1\}$ the corresponding binary setup variable, we must enforce a constraint such as $x_{it} \leq Ms_{it}$, where the big-M constant is typically taken as the sum of the future demand, $\sum_{\tau=t}^{T} d_{i\tau}$, where T is the planning horizon, i.e. the number of time buckets we consider in the model. As this big-M may indeed be too big, weak linear programming relaxations are obtained, and this results in a poorly pruned branch-and-bound tree.

Alternative model formulations have been proposed, based on a plant location or a shortest route framework. The first alternative looks a bit more flexible; in fact, it has also been used in multi-level lot-sizing models (see, however, Stadtler (2003) for a discussion). Such strong model formulations are also fundamental in developing LP-based heuristics, since a reduced integrality gap makes rounding strategies more effective (Maes *et al.* 1991, Alfieri *et al.* 2002).

The basic idea of the plant location formulation is disaggregating the production variable x_{it} into decision variables y_{itp}, denoting the amount of item i produced during time bucket t in order to meet demand in the current or in a future time

bucket p ($p \geq t$). The model can be visualized as the network flow depicted in figure 1 for a single item. Supply and demand nodes are indexed by time periods. The variable x_{it} is the flow entering a supply node, which is routed to meet the demand which flows out of demand nodes. The disaggregate y_{itp} variables are just the flows on the intermediate arcs between the two arrays of nodes. This formulation can be interpreted as a plant location problem where the fixed cost of opening a plant corresponds to the setup cost during a time bucket, and the commodities are shipped in time rather than in space, incurring inventory instead of transportation costs. Assuming a deterministic demand d_{it}, this results in the following model:

$$\begin{aligned}
&\min \sum_i \sum_t \sum_{p \geq t} h_i(p-t) y_{itp} + \sum_i \sum_t f_i s_{it}, \\
&\text{s.t.} \sum_{t \leq p} y_{itp} \geq d_{ip}, \quad \forall\, i, p, \\
&\quad y_{itp} \leq d_{ip} s_{it}, \quad \forall\, i, t, p \geq t, \\
&\quad \sum_i \sum_{p \geq t} r_i y_{itp} + \sum_i r_i' s_{it} \leq R, \quad \forall\, t, \\
&\quad y_{itp} \geq 0, \qquad s_{it} \in \{0, 1\},
\end{aligned} \tag{1}$$

where h_i and f_i are the inventory and setup costs, respectively, for item i; R is the availability of the capacitated resource (we consider one resource, say the bottleneck, the capacity of which does not vary with time; the model can be trivially extended to multiple capacitated resources with time-varying capacity); and r_i and r_i' are the unit processing and setup time for item i, respectively. Initial inventories may be dealt with by netting off the demand. Note that no end-of-horizon issue is taken into account; the end inventory will be zero in the optimal solution. The key advantage of this model formulation is that the big-M in constraint (1) linking production and setup variables is smaller than in the natural formulation (it is the demand in one time bucket, rather than the sum of future demands). This results in remarkable

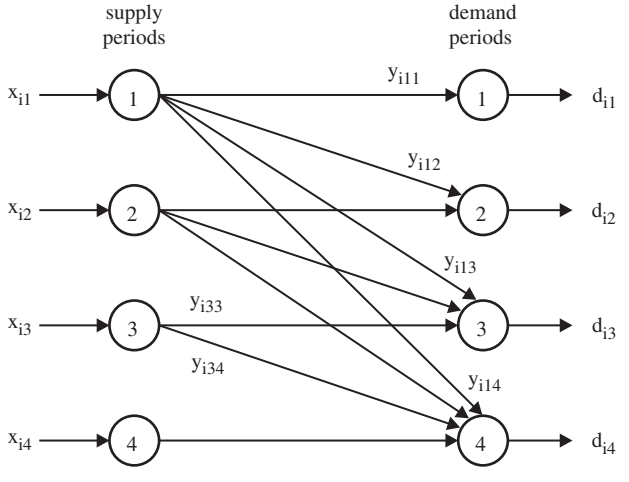

Figure 1. Graphical interpretation of the plant location formulation.

advantages when using LP-based solution algorithms, but it has the unpleasing effect that coming up with a stochastic model is more difficult than with the natural formulation.

To obtain a stochastic model, we need a way to represent demand uncertainty. The typical choice in the stochastic programming literature is a scenario tree, such as that depicted in figure 2. The tree can be regarded as a discretization of a possibly complex probability distribution, where a demand value for all items is associated with each node. The scenario tree may also be shaped according to some expert opinion in the case where the lack of historical data precludes statistical analysis. We have a unique root node representing the current state, where we must take a first-stage planning decision. Then, a decision is associated with each node: contingent on the realized demand value, we will take future planning decisions in a closed-loop manner. In the figure we see a six-period tree, with a branching structure [1, 3, 2, 2, 1, 1]; this means that the first node has three successors, that each of the successors is followed by two nodes, and so on. A sequence of nodes is a scenario, i.e. a realization of the corresponding stochastic process. Note that, in the last two periods, there is no real uncertainty: for instance, if we get to node 22, then we know exactly what the future demand will be for the remaining nodes in the scenario, since the branching factor is 1. Such an information structure may not reflect the real uncertainty, but adding a few nodes, possibly characterized

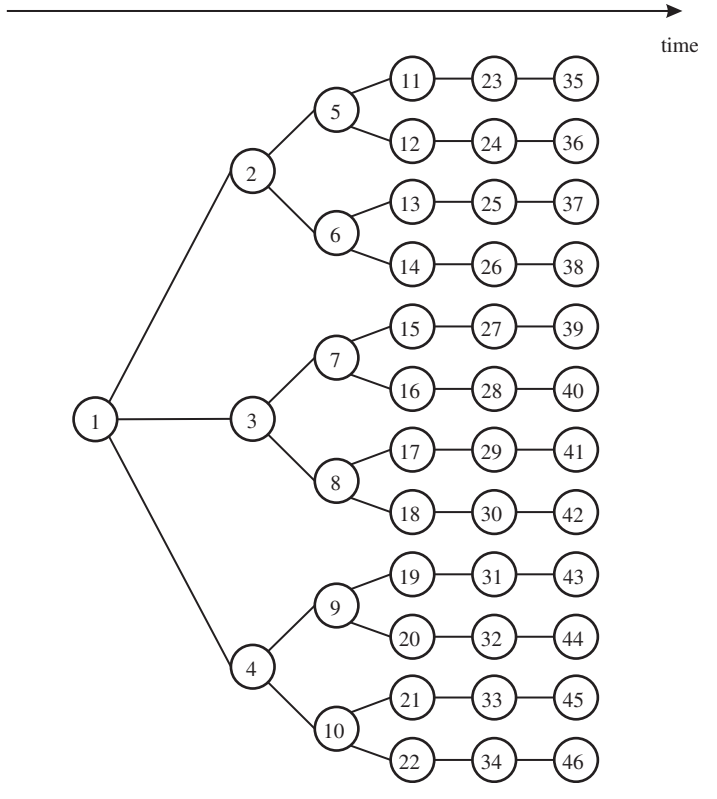

Figure 2. Tree-based representation of demand uncertainty.

by the expected value of demand, according to this pattern, could make sense to overcome end-of-horizon effects, at least partially: it is just a way to increase the planning horizon without increasing the number of scenarios. Each branch in the tree has a conditional probability, which need not be uniform (i.e. if the branching factor is 2, the conditional probability of each branch need not be 0.5). We can easily compute the unconditional probability of occurrence for all nodes in the tree. These probabilities are used in order to define an *expected* value of total costs, which will be our objective function; note that, by using the expected value of the cost, we assume a risk-neutral attitude on the part of the decision maker, which may be reasonable in some multi-period production planning problems (in a companion paper (Brandimarte 2004) we deal with a problem in which taking risk into account may be fundamental).

To write the stochastic multi-stage model, let us introduce the following notation:

- $n \in \mathcal{N}$ is a generic node of the scenario tree; 1 is the root node; $\mathcal{T}$ is the set of terminal nodes in the tree; in figure 2, $\mathcal{N} = \{1, 2, \ldots, 46\}$ and $\mathcal{T} = \{35, 36, \ldots, 46\}$;
- $T(n)$ is the time period for node n; for instance, $T(1) = 1$ and $T(19) = 4$;
- $a(n)$ is the immediate predecessor for node n, $n \neq 1$; in our figure, $a(6) = 2$;
- $\Omega(n, t)$ is the (unique) ancestor of node n at time period t ($n \neq 1$, $t < T(n)$); for instance, $\Omega(27, 2) = 3$;
- $\Sigma(n, t)$ is the *set* of successor nodes of n at time period t ($n \notin \mathcal{T}$, $t > T(n)$); in the figure, $\Sigma(4, 5) = \{31, 32, 33, 34\}$.

The demand data are represented by $d_i^{[n]}$, the demand for item i in node n. We denote the unconditional probability of node n by $p^{[n]}$. Other data are just as in the deterministic model. Note that decisions are taken based only on information concerning the past demand values; they are non-anticipative (Birge and Louveaux 1997).

Before proceeding in the description of the model, it is important to note that different uncertainty profiles are possible. We must clarify the exact flow of information on which the decisions are based. One possibility is that we must take a production decision without even knowing the demand for the *current* time bucket. This is depicted in figure 3(a); we have, in a loose sense, 100% uncertainty over the whole planning horizon. However, in many practical situations, the uncertainty profile will be more like figure 3(b): uncertainty tends to increase with time. This happens when we build a master production schedule based both on customer orders and forecasts; customer orders tend to dominate at the beginning of the planning horizon, and uncertainty increases in the last time buckets where we must rely more on forecasts. In this paper, we consider a third possibility: we discover the current time bucket demand *before* planning production for the first time bucket, and we have a constant level of uncertainty for the future time buckets. This is depicted in figure 3(c), where we see that we now have no uncertainty, and 'full' uncertainty for the future. This may seem arbitrary, and admittedly it is. The point is that, in the first case (figure 3a), the only way to hedge against uncertainty is safety stock; in practice, reactive capacity is another management lever. Our case is clearly stylized, but it allows us to evaluate the interplay between these two levers.

The remaining issue is how we can merge the plant-location-based formulation with the scenario-based representation of uncertainty. In the deterministic

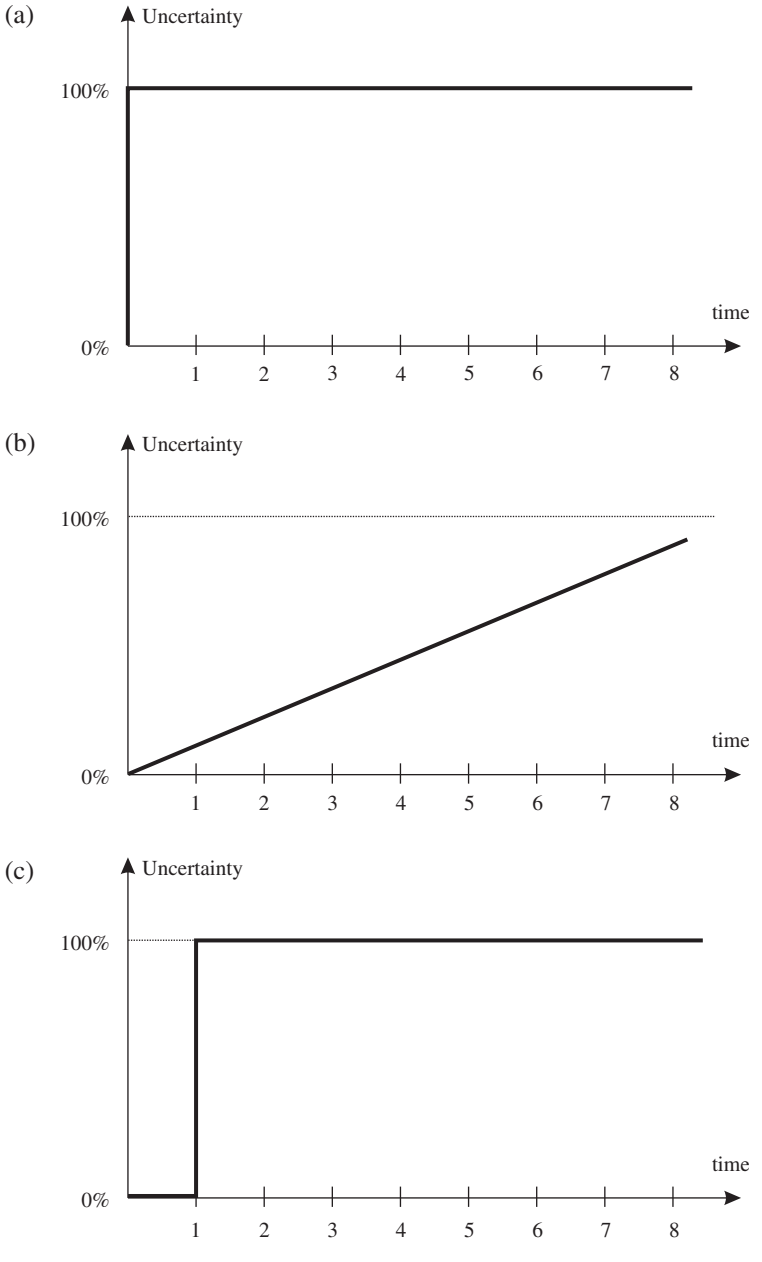

Figure 3. Different profiles of demand uncertainty.

framework we know exactly how many items we should 'ship' from a production/supply node to a demand/destination node. In the stochastic case, we must ship the same amount of commodity to a *set* of successor nodes, as the policy is non-anticipative; we do not know which node among the set of successors will really occur. With reference to figures 1 and 2, we have one demand node for the initial period, but three demand nodes for the second one, and so on. The resulting network

flow structure for each item will be three dimensional, where the scenario tree structure is superimposed on each array of production and demand nodes. Furthermore, since the items shipped to a 'destination' node will not match demand exactly, a more complex flow structure results. For instance, referring to figure 1, if more commodity than necessary is routed to a demand node in period 2, some flow will be transferred in the form of leftover inventory from this node to the demand nodes of time period 3. By the same token, if less commodity than necessary is routed to a demand node, some recourse action must be taken. This could either be a back order, which would be modeled by some flow going backward in time, or a lost sale, which may be represented as a suitably penalized inflow into each demand node. In the following, we consider the lost sale case only, since this is arguably the riskier case, where stochastic programming may be more useful. Lost sales for item i are penalized by a coefficient g_i which is suitably larger than the inventory cost.

These considerations lead us to the following choice of decision variables:

- $s_i^{[n]} \in \{0, 1\}$ is the setup variable for item i in node n;
- $y_{i\tau}^{[n]} \geq 0$ is the amount of i produced in node n to meet demand in time period $\tau \geq T(n)$; note that, because of non-anticipativity, what we ship from supply node n is the same for all its successor nodes in the scenario tree corresponding to each time layer; this is why the y variable is not indexed by the demand node, but by the corresponding time index τ (which is the time index of *demand*, not production);
- $I_i^{[n]} \geq 0$ is leftover inventory, i.e. the surplus amount of item i not consumed at node n and passed for use to *immediate* successor nodes in the tree;
- $z_i^{[n]} \geq 0$ is the lost sale, i.e. the amount of item i demand we fail to meet at node n.

The resulting model formulation of SCLSP is

$$\min \sum_{n \in \mathcal{N}} p^{[n]} \left[\sum_i \left(f_i s_i^{[n]} + h_i I_i^{[n]} + g_i z_i^{[n]} \right) \right] + \sum_{n \in \mathcal{N} \setminus \mathcal{T}} p^{[n]} \left[\sum_i \sum_{\tau > T(n)} h_i (\tau - T(n)) y_{i\tau}^{[n]} \right], \tag{2}$$

$$\text{s.t. } I_i^{[a(n)]} + \sum_{t < T(n)} y_{i,T(n)}^{[\Omega(n,t)]} + y_{i,T(n)}^{[n]} = d_i^{[n]} + I_i^{[n]} - z_i^{[n]}, \quad \forall\, i, n, \tag{3}$$

$$y_{i\tau}^{[n]} \leq \left(\max_{j \in \Sigma(n,\tau)} d_i^{[j]} \right) s_i^{[n]}, \quad \forall\, i, n, \tau > T(n), \tag{4}$$

$$y_{i,T(n)}^{[n]} \leq d_i^{[n]} s_i^{[n]}, \quad \forall\, i, n, \tag{5}$$

$$\sum_i \sum_{\tau \geq T(n)} r_i y_{i\tau}^{[n]} + \sum_i r_i' s_i^{[n]} \leq R, \quad \forall\, n, \tag{6}$$

$$y_{i\tau}^{[n]}, I_i^{[n]}, z_i^{[n]} \geq 0; \quad s_i^{[n]} \in \{0, 1\}.$$

The first term of the objective function (2) accounts for the expected value of setup costs, leftover inventory (which is the surplus of production with respect to demand and stays in inventory for one time period), and lost sales. The second term is the expected inventory cost, just like in the deterministic model: it corresponds to flows from supply to demand nodes. It involves non-terminal nodes only, i.e. nodes in $\mathcal{N}$ which are not in $\mathcal{T}$, denoted by the set difference $\mathcal{N} \setminus \mathcal{T}$; in fact, terminal nodes do not ship anything to the future intentionally (apart from leftover inventory at the end of the planning horizon, which is zero for the deterministic model, but not necessarily in the stochastic case).

Constraint (3) is just a flow balance constraint in each demand node. The inflow is the sum of the leftover inventory from the immediate predecessor node in the tree, plus the sum of all the shipments from ancestor nodes, plus the production for immediate consumption. The outflow is the demand, plus leftover inventory to immediate successor nodes, minus lost sales. Strictly speaking, we should write a different balance constraint for the initial node, since its predecessor is not defined. In this case, $I_i^{[a(n)]}$ should be considered as the starting inventory, which is part of the problem data.

When linking setup and production variables, some care must be taken in selecting the right big-M. In (4) we must take the maximum over the possible demands of the successor nodes we are shipping items to. In the case of (5), we are dealing with production for immediate use, and in this case the demand is known (given our assumptions). The capacity constraint and the variable definition constraints are self-explanatory.

We should mention that a similar formulation is used, within a different context, in Ahmed *et al*. (2003).

3. Solution strategies

The above model is, for a realistic problem, a large-scale MILP model. NP-hardness issues compound with the difficulties of multi-stage stochastic programming models, resulting in an intractable problem. We should also mention that our SCLSP model formulation has a couple of additional difficulties with respect to its deterministic counterpart, leaving its sheer size aside. One is that, in constraint (4), we must take the maximum over the set of demand values over the successor nodes; this obviously weakens the formulation. A less obvious point is that it is possible to route items through both 'intentional' shipment arcs in the graph (those corresponding to decision variables $y_{i\tau}^{[n]}$) and leftover inventory $I_i^{[n]}$. This creates multiple equivalent optima which correspond to the same practical production plan, but have the effect of making branch and bound less effective. So, we must resort to heuristic solution strategies.

In order to keep the size of the model to a manageable size, suitable scenario generation strategies must be considered, and this is the topic of the next section. Here we consider mathematically motivated heuristic solution strategies. We want to stick to LP-based strategies, such as rounding strategies or truncated partial enumeration, since such strategies are not too sensitive to problem structure, unlike Lagrangian relaxation approaches.

In Alfieri *et al.* (2002), naive rounding strategies are found to be quite effective, when coupled with strong model formulations, to solve deterministic lot-sizing problems where setup times are negligible. The idea is to solve a sequence of LP-relaxations, where setup variables set to zero or one by the simplex algorithm, possibly within some integrality tolerance, are 'frozen' to their integer value, and the largest setup variable is rounded to one. The rationale is that if we have a large fractional setup variable for an item in a time bucket, then it is probably a good idea to produce that item in that time period. In the case of no setup times, this procedure is guaranteed to find a feasible integer solution if there is one. Unfortunately, when a realistic model with setup times is considered, the problem is much more difficult; indeed, even the feasibility problem is NP-complete when setup times are non-negligible (Maes *et al.* 1991). In fact, setting a setup variable to zero may result in later infeasibilities. So, even if feasibility is guaranteed in our model formulation by the fact that we leave room for lost sales, we need a better strategy.

One possibility is the fix-and-relax approach proposed by Dillenberger *et al.* (1994). The idea is to partition the set of integer variables into subsets, and solve a sequence of MILP problems with respect to selected subsets of integer variables, relaxing the remaining ones to continuous values. In the first subproblem, all subsets but the first are relaxed to continuous values, which results in a reduced MILP model. In the second subproblem, variables in the first subset are fixed to the value obtained by solving the first subproblem, and a MILP problem is solved with respect to variables in the second subset, relaxing the remaining variables to continuous values, and so on.

In our case, a natural choice is to partition the setup variables with respect to the time index, resulting in a sort of forward time-sweep strategy which enforces integrality of the setup variables one time-layer at a time.

(1) Set $k = 1$.
(2) Solve SCLSP by branch and bound with fixed setup variables for $t < k$ and relaxed setup variables for $t > k$ (possibly within a relative suboptimality gap).
(3) Set $k = k + 1$ and repeat step 2 until the end of the planning horizon is reached.

The motivation for this heuristic stems from its interpretation as a sort of dynamic programming algorithm whereby the cost-to-go is approximated by the LP-relaxation of the future setup variables. Note that, in step 2, only the setup variables of time layer k are restricted to integer values. This eases the computational burden considerably. Furthermore, there are additional degrees of freedom, since each MILP subproblem may be solved within some suboptimality tolerance (say 1 or 5%; a relative suboptimality tolerance ε allows us to fathom a branch-and-bound node not only when the estimated lower bound LB is larger than or equal to an upper bound UB corresponding to the cost of a feasible solution, which ensures optimality, but also when $(1 + \epsilon)LB \geq UB$, which may drastically cut the CPU time, while ensuring a guaranteed near-optimal solution). This tolerance need not be the same for all time periods, since the first few time buckets are arguably more important than the later ones. After all, what we really need is the production plan for the first time bucket (in our simplified case; in a true master production

schedule, we must anticipate orders for subassemblies and components). So, there is room for flexible performance adjustments.

4. Scenario generation and simulation architecture

We have considered a simple case where demand is stationary, normally distributed (with arbitrary correlations), and there is no inter-temporal dependence. In this setting, sampling the demand process is rather straightforward. However, since we must keep the size of the scenario tree quite limited, we should be careful in choosing a suitable strategy.

Given a scenario tree structure in terms of branching factor per node at each stage, the simplest approach is to draw a few samples from the multivariate normal distribution using standard methods (see, e.g., Law and Kelton (1999)). We will refer to this approach as pure random sampling. Even in this case, there are some choices to be taken. Consider again the tree of figure 2. When sampling demand in nodes (5, 6), (7, 8), and (9, 10) should we use the same demand samples in the three pairs or not? The first idea, i.e. replicating scenarios across the tree, is statistically equivalent to the second, since demand is independent of the predecessor nodes. However, on the one hand, one may argue that using different samples is preferable, given the very limited size of the sample: if you take a 'bad' sample, it is not repeated. On the other hand, one may argue that using different samples may result in spurious time-dependence. *A priori*, which is the better strategy is not clear. Furthermore, when a node is the unique successor of another node (as nodes 23 and 35 in figure 2), it may make sense to use the expected value of demand rather than a random sample.

Since the number of samples is so limited, one could apply variance reduction strategies. One possibility is antithetic sampling, a well-known variance reduction strategy (see, e.g., Brandimarte (2001)). In this case, antithetic sampling is obtained by taking pairs of symmetric samples with respect to the expected value. When the branching factor is an odd number, one sample is always set to the expected value. We see that doing so allows us to match expected value and skewness (the normal distribution we assume is symmetric, so skewness is zero). Also in this case we may or may not replicate scenarios across the tree.

Another possibility is Latin hypercube sampling, which is a form of stratification. We have used the `lhsnorm` function provided by the Statistics Toolbox in MATLAB. We have used different variations of Latin hypercube sampling, resulting from two choices. As in the previous approaches, we may or may not replicate scenarios across the tree. Furthermore, when the branching factor is 1, we may set the demand value to the expected value, or we may sample the distribution. We should also mention that Latin hypercube sampling is somewhat related to descriptive sampling (Saliby 1990), which is based on a deterministic selection of the input values, followed by a random permutation; this approach has been proposed by Ponce-Ortega *et al.* (2004) for stochastic programming.

There is still another possibility, which is to generate a set of 'optimized' scenarios by trying to match the known moments as much as possible (Hoyland and Wallace 2001). This approach is intuitively appealing, even though counterexamples have been provided showing that completely different distributions

may match the first few moments (Hochreiter and Pflug 2002). In our case, we assume to know the expected values μ_i of demand for item i, as well as the variance σ_i^2 and the set of covariances σ_{ij} for each pair (i,j) of items ($\sigma_{ii} = \sigma_i^2$). Furthermore, since we assume a normal distribution, we know that skewness $\xi = E[(\tilde{d} - \mu)^3/\sigma^3]$ should be zero and that kurtosis $\chi = E[(\tilde{d} - \mu)^4/\sigma^4]$ should be 3.

We assume that the conditional probabilities of each node within a branching are equal and that, given independence, we generate demand scenarios successively over time. In general, one may consider the probabilities of each branch as a decision variable. Let us denote by $d_i^{[s]}$ the demand for item i in node s belonging to a certain branching of size S. Natural requirements are:

$$\frac{1}{S}\sum_s d_i^{[s]} \approx \mu_i, \quad \forall\, i,$$

$$\frac{1}{S}\sum_s \left(d_i^{[s]} - \mu_i\right)\left(d_j^{[s]} - \mu_j\right) \approx \sigma_{ij}, \quad \forall\, i,j,$$

$$\frac{1}{S}\sum_s \frac{\left(d_i^{[s]} - \mu_i\right)^3}{\sigma_i^3} \approx 0, \quad \forall\, i,$$

$$\frac{1}{S}\sum_s \frac{\left(d_i^{[s]} - \mu_i\right)^4}{\sigma_i^4} \approx 3, \quad \forall\, i.$$

Note that we divide by S since the parameters are known *a priori* and not estimated from the data. Approximate moment matching is obtained by minimizing the squared error

$$w_1 \sum_i \left[\frac{1}{S}\sum_s d_i^{[s]} - \mu_i\right]^2 + w_2 \sum_{i,j} \left[\frac{1}{S}\sum_s \left(d_i^{[s]} - \mu_i\right)\left(d_j^{[s]} - \mu_j\right) - \sigma_{ij}\right]^2$$
$$+ w_3 \sum_i \left[\frac{1}{S}\sum_s \left(\frac{d_i^{[s]} - \mu_i}{\sigma_i}\right)^3\right]^2 + w_4 \sum_i \left[\frac{1}{S}\sum_s \left(\frac{d_i^{[s]} - \mu_i}{\sigma_i}\right)^4 - 3\right]^2 \qquad (7)$$

over non-negative demand values $d_i^{[s]}$. The objective function includes four weights w_k which may be used to fine-tune performance. When using scenario optimization, scenarios are obviously replicated across the tree.

Of course, scenario generation is (at best) an approximation of the true uncertainty, and a proper way to evaluate performance must simulate the application of the model to out-of-sample scenarios, according to a rolling horizon strategy. In fact, only the first-stage decision variables would be implemented. To this end, we have built a simulation architecture, illustrated in figure 4, based on MATLAB and ILOG CPLEX/AMPL (Fourer *et al.* 2002). Given the problem features, we use MATLAB both to simulate a certain demand history (out-of-sample scenarios) and to generate the AMPL data files for the optimization models, including the demand scenarios. The MATLAB Optimization Toolbox is used for optimized scenario generation; we have used the `fmincon` function; the initial solution is provided by Latin hypercube sampling. Then an AMPL script implements the time-sweep heuristic by invoking the CPLEX solver. The application of the first-stage

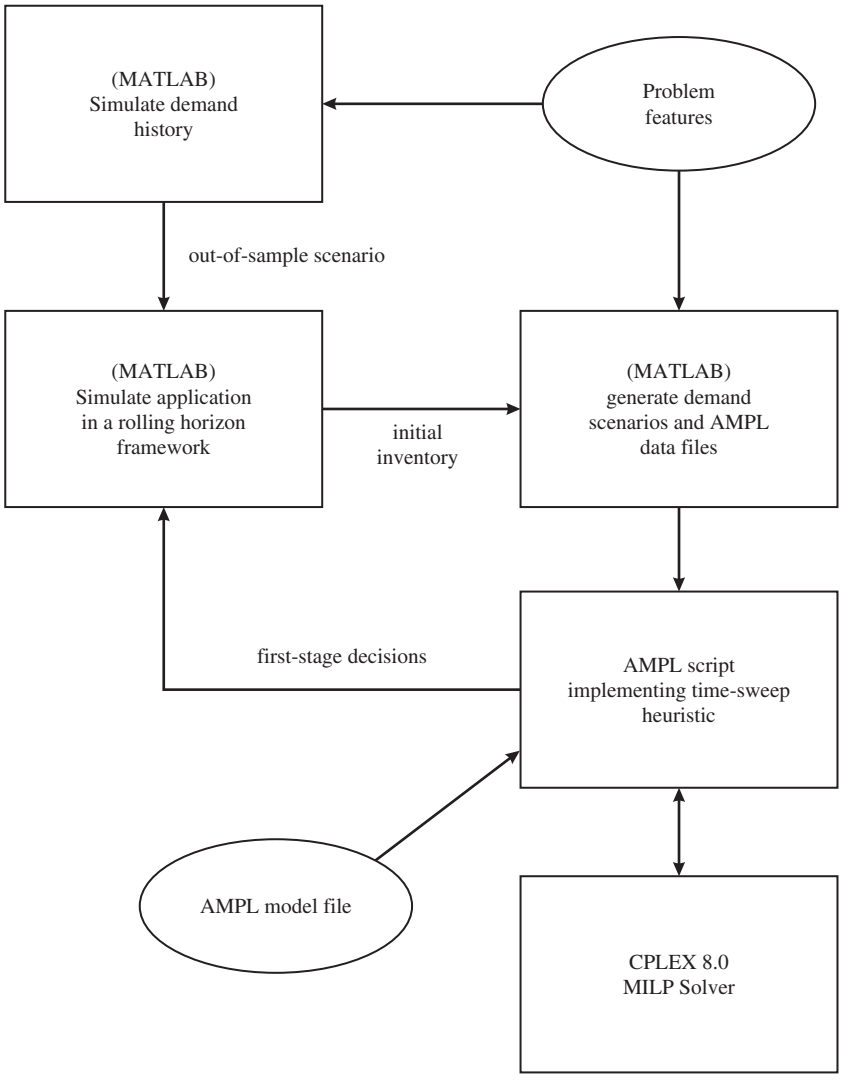

Figure 4. Simulation architecture for computational experiments.

decision is simulated on the 'true' demand data for the current period. Note that, given our assumptions, the demand for the first period is the same in the simulated scenarios and in the scenario tree. The link between the current and next simulated period is represented by the leftover inventory resulting from the planned production after meeting demand; this will be the initial inventory for the next simulation step.

The same demand history is simulated for the stochastic and the deterministic model, coherently with the principle of variance reduction based on common random numbers (Law and Kelton 1999). The results of the simulation depend on the initial value of the inventory levels. Since we are interested in the relative performance of the two models, and not in estimating long-term average costs, we do not discard the initial part of each simulation run.

A last remark concerns the exact knowledge of the true parameters of the demand-generating process that we may have in a practical setting. In fact, figure 4 suggests that the same distribution and parameters are used to generate the tree and the out-of-sample scenarios. Actually, in the computational experiments described in the following we have also tried to check the robustness of the approach by introducing some error in the parameters used in building the scenario tree: in this case we use some parameters to generate the scenario tree, and we slightly perturb them to generate the out-of-sample scenarios for the simulation.

5. Computational experiments

5.1 *Problem instance generation*

In order to generate problem instances, we have basically followed the standard approach adopted in the lot-sizing literature.

- We assume that demand within each time bucket has a multivariate normal distribution; in the case where a negative demand is drawn when generating a scenario, we set it to zero. There is no inter-temporal dependence. The average demand for each item is drawn as a uniform number between 10 and 100. In order to set the demand variance, we generate a coefficient of variation for each item (in our experiments it was uniformly distributed between 0.1 and 0.8); from the average demand and the coefficient of variation we obtain the demand variance. To generate the covariance matrix, we select a correlation matrix in order to model the independence, or positive and negative correlation among products; then, given variances and correlations we generate the covariance matrix.
- The inventory cost for each item is generated by drawing a uniform random number between 1 and 10.
- The penalties for lost sales are drawn from the uniform distribution between 200 and 300.
- The initial inventory level for each item is uniformly distributed between 0 and twice the average demand for that item; this is a bit arbitrary, but since our aim is to compare different strategies starting from the same conditions, this should not affect the result too much. This choice is better than simply starting from empty inventories.
- The setup cost is generated as a function of the time between order $T_{bo,i}$ of item i, which, in the case of a deterministic and constant demand D_i, is related to the economic order quantity Q_i by $T_{bo,i} = D_i/Q_i$; from the EOQ formula we see that

$$f_i = \frac{D_i}{2} T_{bo,i}^2 h_i.$$

  In most instances that we consider, setup times are more important than setup costs; in manufacturing, having negligible setup costs is more likely than having negligible setup times. Indeed, setup costs are often just an accounting trick to cope with capacity constraints (Hopp and Spearman 2000). So, we usually set the time between orders to an intermediate value

(2 in our experiments), and compute f_i given average demand and inventory cost for each item. We have also run a few experiments with larger setup costs, setting the time between orders to 4. Then, we add some noise by multiplying the setup cost for each item by a uniform random number between 0.8 and 1.2.

- We consider only one bottleneck resource; the processing time for each item is uniformly distributed between 1 and 5.
- To generate the available capacity, we multiply the average demand for each item by the processing time, we sum over items, and we multiply the result by a capacity factor (e.g., 1.8); for instance, a unit capacity factor means that, if setup times were negligible, during each time bucket we would have enough capacity to produce the average demand just in time; a larger capacity factor may or may not indicate a large capacity, depending on demand variance and the impact of setup times.
- Setup times are generated on the basis of a setup time factor. This factor is the fraction of available capacity which would be lost if, during a time bucket, we did a setup for all items; for instance, if the setup factor is 0.5, this means that 50% of the available capacity is required for setups if all the items are produced during the same time bucket. Given this amount, the setup time for each item is generated by 'allocating' this total setup time to the items using uniform random numbers.

5.2 *Comparing solution strategies*

The first set of experiments was aimed at assessing the efficiency of the time-sweep heuristic with respect to full branch and bound. Experiments were carried out on an IBM RISC 6000 44P workstation (model 270, CPU 375 MHz) using ILOG AMPL and CPLEX 8.0. We considered four problem classes, depending on the capacity factor (set to 1.5 and 2) and the setup factor (set to 0.3 and 0.5). The number of items was 20 and the remaining parameters were set as described in the previous section. Scenario generation was purely random without replication across the tree. The branching structure was (1, 5, 3, 3, 2, 1), resulting in 246 nodes and 90 scenarios. We generated five problem instances per class, resulting in 20 problem instances, and compared solution times and objective values.

In the time-sweep heuristic, the suboptimality tolerance was 1% for each subproblem. Full branch and bound was still running after 5000 s for all problem instances. So we stopped branch-and-bound runs after 2000 and 5000 s in order to assess how much is gained by extended computational effort. The results are shown in table 1. For each of the four problem classes we give its characteristics (note that class A is the most difficult, as capacity is tighter and setup times are larger), the average CPU time of the time-sweep heuristic, and the average relative percentage deviation of the solution cost of the solution obtained by the time-sweep heuristic (S_{TS}) with respect to the solution obtained by full branch and bound (S_{BB}) after 2000 and 5000 s. This relative deviation is computed as

$$\frac{S_{TS} - S_{BB}}{S_{BB}}.$$

Table 1. Comparison of the time-sweep heuristic and full branch and bound on 20-item instances. Each class is characterized by the capacity and setup factor. We show the CPU time for the heuristic, and the percentage deviation of the heuristic solutions with respect to those provided by branch and bound stopped after 2000 and 5000 s.

Class	Capacity factor	Setup factor	CPU time (s)	BB2000 (%)	BB5000 (%)
A	1.5	0.5	207.93	−0.77	0.73
B	1.5	0.3	98.36	1.16	1.51
C	2.0	0.5	93.54	1.14	1.37
D	2.0	0.3	73.23	1.45	1.49
Total average			118.26	0.75	1.27

Table 2. Assessing sampling uncertainty. The table reports the average mean absolute deviation (MAD) and the maximum mean absolute deviation in objective value over ten randomly sampled scenario trees, for five problem instances in each problem class.

Class	Average MAD (%)	Maximum MAD (%)
A	1.29	1.70
B	1.43	1.64
C	1.21	1.90
D	1.37	1.83

The first four rows of the table are averaged over five problem instances per class; the last row is the general average for the 20 instances.

The first point is that the heuristic requires a non-negligible amount of time and that class A is indeed the most difficult, taking 207.93 CPU seconds on average. However, the solution obtained by the heuristic is, on average, better than that obtained after 2000 s of branch and bound, as we have a negative percentage increase (−0.77%); with a considerable increase in computational effort we are able to do better (the heuristic solution is, on average, 0.73% worse than the solution obtained by branch and bound after 5000 s). For the other problem classes, branch and bound does relatively better. The total average suggests that the heuristic solution is more or less 1% worse than the solution obtained by full branch and bound. One could debate if this is a large difference or not, considering the non-negligible effort of the time-sweep approach. However, this way of tackling the problem is not very sensible. To see this, consider the data of table 2.

These are the results of a second test aimed at assessing the *sampling uncertainty*. Since the stochastic model is based on a randomly sampled scenario tree, the value of the objective function is also random. How much variability in the objective value is induced by the variability in the sampling scenarios? We considered the same problem instances of the first experiment, and, for each instance, we sampled ten scenario trees, comparing the objective values obtained by the time-sweep heuristic. For each instance, variability is measured by the mean absolute deviation of the

objective values for the ten sampled trees. In table 2 we report the average mean absolute deviation (averaged for the five problem instances of each problem class), and the maximum mean absolute deviation (over the set of problem instances). We see that this sampling uncertainty is actually larger than the relative 'error' of the heuristic with respect to branch and bound.

Moreover, we should also consider that the objective function of SCLSP is not the true cost one would incur in implementing the solution in a real setting, with out-of-sample data within a rolling horizon framework. This type of assessment calls for the simulation approach described in section 4.

The first two experiments suggest that the time-sweep heuristic performs well with respect to branch and bound. As a further test, since 20 items are not too many, we performed another experiment with larger instances: the problem classes and features were the same as before, but we considered 50 items. Given the larger number of items we relaxed the relative suboptimality tolerance for the time-sweep heuristic from 1 to 2%. The results are reported in table 3 and should be read just like table 1. We can see that increasing the number of items results in a corresponding increase in the CPU time for the time-sweep heuristic, which is certainly not computationally cheap. However, we can see that, particularly for class A, the heuristic performs rather well. It should also be noted that, for one specific instance within class A, the incumbent solution (i.e. best integer so far) found by branch and bound after 5000 s was 24.75% worse than that provided by the heuristic.

We see that the relative assessment of the two solution methods depends on the problem class, since, for the last class, branch and bound seems competitive, but the heuristic is somewhat more robust with respect to the problem features. Furthermore, the heuristic approach offers several degrees of freedom in optimizing the performance, since the suboptimality tolerance need not be the same for all the time layers. Actually, one could even consider giving up integrality of the setup variables for the later time periods, since their role is just to obtain a good set of first-stage decision variables.

As a final test, we have run the time-sweep heuristic on a set of instances in order to assess the relative impact on CPU times of the number of items and the size of the scenario tree. We have considered instances with 5, 20, and 40 items, with a setup factor of 0.5, a capacity factor of 1.5, and two branching structures.

Table 3. Comparison of the time-sweep heuristic and full branch and bound on 50-item instances. We list the CPU time for the heuristic and the average percentage deviation from the best incumbent when branch and bound is stopped after 2000 and 5000 s.

Class	Capacity factor	Setup factor	CPU time (s)	BB2000 (%)	BB5000 (%)
A	1.5	0.5	425.32	−12.18	−4.93
B	1.5	0.3	255.95	−6.82	2.03
C	2.0	0.5	199.37	1.31	3.03
D	2.0	0.3	181.70	2.23	3.23
Total average			265.59	−3.86	0.84

The first tree structure is [1, 5, 3, 3, 2, 1] and the second is [1, 8, 5, 4, 3, 2]. Note that this enlarged scenario tree involves, in the case of 40 items:

- 960 scenarios and 1649 nodes;
- about 300 000 decision variables (the number of binary setup variables is $40 \times 1649 = 65\,960$);
- about 170 000 constraints.

Given this size, we had to set the relative suboptimality tolerance within each subproblem to 5%. Five instances were generated for each problem/tree class. The average CPU times are shown in table 4. We see that while increasing the number of items has an obvious impact on computational effort, a much larger effect is due to the increased tree size. This shows that we must keep the branching structure of the tree quite limited with respect to the uncertainty it should represent. However, the real aim of a scenario tree is not really to represent uncertainty to an arbitrary accuracy, but to obtain robust solutions. Robustness can only be evaluated relative to a benchmark, based on a simulation of out-of-sample scenarios. This evaluation is carried out in the next section.

5.3 *Rolling horizon simulation of small size instances*

The following experiments required MATLAB, as well as ILOG AMPL/CPLEX, and were carried out on a HP Vectra PC, with a Pentium III 1 GHz processor. The aim was not to assess speed, but to evaluate the practical performance of the stochastic planning model with respect to the simpler deterministic model ignoring uncertainty. The basic setting of parameters to generate problem instances is the same as in the previous section.

Each experiment consists of simulating the application of the first-stage planning decisions over 30 time periods, and comparing the relative increase of the total cost we pay for ignoring uncertainty (refer to figure 4). This percentage cost increase is computed as $(C_D - C_S)/C_S$, where C_D and C_S are the total costs accumulated over each simulation run for the deterministic and the stochastic model, respectively. Note that the cost we consider here is not the objective function of the optimization model, but the sum over time of the true costs we incur during the first time bucket of the rolling horizon, in which we apply the first-stage planning decision.

We varied the setup and capacity factors and some demand characteristics, as illustrated in table 5. For each type of problem we generated ten instances.

Table 4. Average CPU times (seconds) for the time-sweep heuristic as a function of the number of items and scenario tree structure. S1 refers to branching structure [1, 5, 3, 3, 2, 1], and S2 to [1, 8, 5, 4, 3, 2]. Five instances were generated per problem type.

	Items		
	5	20	40
S1	32.23	86.45	243.41
S2	1362.22	4142.51	7616.51

Table 5. Assessing the percentage increase in cost from using the deterministic model ignoring uncertainty. For each problem class we give the average increase (Δ) and its mean absolute deviation (MAD) over ten instances.

Exp.	Description of conditions	Δ (%)	MAD (%)
1	Independent demand; setup factor 0.3; capacity factor 1.8	2.31	2.34
2	Independent demand; setup factor 0.5; capacity factor 1.8	139.80	77.21
3	Independent demand; setup factor 0.3; capacity factor 1.8; true expected demand is 10% higher than assumed	7.34	7.11
4	Independent demand; setup factor 0.3; capacity factor 1.8; true expected demand is 90% of assumed expected value	−1.28	5.29
5	Independent demand; setup factor 0.5; capacity factor 1.8; true expected demand is 90% of assumed expected value	23.92	27.80
6	Demand is positively correlated ($\rho = 0.7$); setup factor 0.3; capacity factor 1.8	38.45	49.44
7	20 items; independent demand; setup factor 0.5; capacity factor 1.8	0.43	1.23
8	20 items; independent demand; setup factor 0.5; capacity factor 1.5	62.53	28.48
9	Same as experiment 1 (five items, capacity factor 1.8, setup factor 0.3) but time between orders is 4	0.25	4.17
10	Same as experiment 9, but capacity factor 1.5	14.52	18.85
11	Same as experiment 9, but capacity factor 1.5 and setup factor 0.5	75.99	55.30

Since running the simulation is quite costly (solving SCLSP requires half a minute, but repeated simulation on many instances calls for several CPU hours), we mostly considered small instances with only five items in order to gain some basic insights.

The branching structure is $[1, 5, 3, 3, 2, 1]$. Since for these small instances the deterministic model is easy to solve, we use full branch and bound, with no sub-optimality tolerance, for the deterministic model. The stochastic model is solved by the time-sweep heuristic with a 5% suboptimality tolerance. This is needed to speed up the simulation and makes the comparison a bit fairer by counter-balancing the increased CPU effort required by SCLSP. In a practical setting, we would not be able to solve SCLSP to optimality.

We carried out a first set of 11 experiments. For each experiment, table 5 reports the average relative increase in cost, as defined before, and its mean absolute deviation. We see that this mean absolute deviation is rather large. This is partly due to the large variability of our problem instance generation framework: we may have quite different problem instances within the same group, also because of the small number of items involved. Furthermore, the intrinsic variability of demand, over only 30 periods, may contribute. Just one 'odd' instance may have a large impact on our statistics and we must be extremely careful in drawing conclusions.

In experiment 1, we have five items, independent demand, a relatively low impact of setup times (setup factor 0.3) and a relatively high capacity (capacity factor 1.8). We see that, on average, we lose 2.31% by ignoring uncertainty, which may not be so impressive. Furthermore, the mean absolute deviation is 2.34%, suggesting that the stochastic model is not reliably better than the deterministic model. This result is arguably due to the fact that, with this problem setting, there is sufficient reactive capacity to obtain a good solution even with a deterministic model. The stochastic model is able to build safety stocks, but this is not needed in this case. When capacity is tighter, or the impact of setup times is larger, it is important to build safety stocks. In experiment 2, we raise the setup factor to 0.5, and this results in an average increase in cost of 139.80% by neglecting uncertainty. Of course, this result should be taken with great care, given the large mean absolute deviation, but it definitely suggests that, provided lost sales have a large penalty, the stochastic model does a better job.

An important issue in practice is the accuracy of our model of uncertainty. For instance, the expected demand of the model will probably be different from the expected demand of the 'true' demand process. In experiment 3, which has the same parameter settings as experiment 1, the expected value of the simulated demand for all items is 10% larger than the expected value assumed in the two mathematical models; the demand variances are the same. In this case we have a larger increase in cost, 7.34% instead of 2.31%. This could suggest that a stochastic model may be more robust to parameter uncertainty. Actually, apart from the caution suggested by the large MAD, one could postulate what happens if the true expected demand is smaller than assumed. In experiment 4 we simulate a demand process with an expected demand that is 90% of that assumed in the models. In this case, the stochastic model performs worse, on average, because it builds unnecessary safety stocks. But if reactive capacity is low, as in experiment 5, we see that neglecting uncertainty has a very large cost, even if the type of parametric error is the same as in experiment 4.

Of course, errors in expected values are not the only ones we may see in practice. We may over- or underestimate correlations, the demand distribution may be asymmetric, and so on. Experiment 6 shows that if demands are positively correlated ($\rho = 0.7$ for all pairs of items), neglecting uncertainty may also have a remarkable cost. Of course, a negative correlation tends to mitigate the problem by leaving more capacity to react just in time.

In experiments 7 and 8 we examine what happens with 20 items, in a setting similar to experiment 2. In fact, one may question if the poor performance of the deterministic model in that case is due to the large impact that a single setup has. If the setup factor is 0.5 and there are only five items, carrying out a setup for an item has a large impact on the reactive capacity. This may happen, for example, if there are a few product families with large setup times and demand for items within the same family is positively correlated so that there are no 'risk pooling' effects. However, if there are more items, one may question whether the advantage of the stochastic model is so remarkable, since the impact of a single setup is smaller. In fact, experiment 7 shows that, with the capacity factor set to 1.8, there is no gain from the stochastic model. But if the capacity factor is 1.5, as in experiment 8, we see again the benefit of modeling uncertainty explicitly.

Finally, in experiments 9–11 we consider the case of more significant setup costs, obtained by setting the time between orders to 4. Experiment 9 is similar to experiment 1, and we see that using stochastic programming is even less attractive with larger setup costs. A reasonable explanation for this is that, due to the increased setup costs, the deterministic solver increases the inventory level by building a cycle stock, which is also used as a safety stock to hedge against uncertainty. However, experiments 10 and 11 show that, if available capacity is lower and setup times are larger, the stochastic model performs relatively better, repeating the pattern of previous experiments.

A second set of experiments was carried out in order to compare different scenario generation strategies. Since the problem instances considered in experiment 2 are those where the benefit of stochastic programming is larger, we have simulated ten instances for this class, using the following scenario generation strategies (refer to column 1 of table 6):

- latin1 and latin2 refer to Latin hypercube sampling; in the second case, when the branching factor is 1, the expected value of demand is used (this is a form of moment matching);
- random and antithetic refer to pure random and antithetic sampling, respectively;
- 'yes' and 'no' refer to the use, or not, of scenario replication across the tree;
- 'opt' refers to optimized scenario generation, with different settings of the four weights in the objective function (7): for instance, in the first case we consider only the expected value and covariance matrix; in the second and third cases we also consider skewness and kurtosis; in the remaining cases we experiment with different weight settings.

In table 6 we give the average increase in cost from using the deterministic model over the ten instances, its mean absolute deviation, and the mean deviation with

Table 6. Comparison among different scenario generation approaches in terms of the percentage cost increase of the deterministic solution with respect to the stochastic solution. Data include the average cost increase, the mean absolute deviation (MAD), and the mean deviation from the best (MDB).

Scenario generation	Mean (%)	MAD (%)	MDB (%)
Latin1/yes	60.93	29.69	10.11
Latin2/yes	58.07	29.71	12.96
Random/yes	56.42	30.81	14.62
Antithetic/yes	50.65	38.52	20.38
Latin1/no	52.37	32.54	18.67
Latin2/no	64.17	25.61	6.87
Random/no	49.92	24.14	21.11
Antithetic/no	60.63	31.00	10.40
Opt/[1, 1, 0, 0]	41.73	34.80	29.30
Opt/[1, 1, 1, 0]	51.72	34.87	19.31
Opt/[1, 1, 1, 1]	51.09	30.91	19.94
Opt/[1, 1, 1, 0.5]	46.54	25.89	24.49
Opt/[1, 10, 1, 1]	60.50	26.73	10.53
Opt/[1, 1, 10, 1]	60.28	28.18	10.75

respect to the best approach. The last quantity is a measure of robustness: for each instance, we take the best result over the different scenario generation strategies, and we compute the deviation of each strategy with respect to this target. Then we take the average deviation over the ten instances.

The first striking observation is the difference between the results we obtain here and those obtained in experiment 2, in terms of relative cost increase. This is no surprise given the large MAD values, but it again stresses that great care must be taken when interpreting the results.

Keeping this in mind, the results depict something like a bad news/good news situation. The bad news is that it is difficult to see a clear winner. The good news is that no method yields a consistently poor performance. However, this is also probably due to the small number of items; with more items, representing uncertainty with a handful of scenarios is certainly more difficult. Apparently, the second version of Latin hypercube sampling, without scenario replication (latin2/no), performs best, and is remarkably robust (the average deviation from the best is 6.87%, which is the smallest value). Optimized scenario generation may also work well, but we see that setting the weights in the objective function is important.

6. Conclusions, critical remarks, and further research directions

In this paper we have considered a plant-location-based formulation of a stochastic version of the well-known CLSP model. As customary in multi-stage stochastic programming with recourse, uncertainty is represented by a scenario tree. This modeling framework results in a large-scale MILP problem. For its solution we have proposed a time-sweep-based heuristic solution strategy which relies on commercially available branch-and-bound methods to solve a sequence of restricted MILP subproblems. The heuristic strategy is not computationally cheap, but it can be applied in a practical setting, since this is not a real-time problem, and it works well with respect to full branch and bound, particularly for difficult problem instances. We have not considered special purpose methods for stochastic mixed-integer programming, which are an active field of current research (see, e.g., Sen (2003) for a recent survey). In fact, while there are efficient methods for specific two-stage stochastic MILP models, much less is known about general multi-stage stochastic MILP models. It should also be noted that off-the-shelf software such as CPLEX also includes rounding heuristics to obtain good upper bounds and cut generation mechanisms to improve the lower bounds. To solve our model instances, much work is done in the root problem of the search tree by the rounding heuristic and a good number of cuts are generated, including both Gomory cuts and flow cuts (Wolsey 1998). Actually, since we use a suboptimality tolerance, only a few branch-and-bound nodes are explored. In order to obtain a better solution approach, one should be able to incorporate novel algorithms within this architecture, which is not trivial. However, solving the first LP relaxation by specialized methods (e.g., nested Benders decomposition) could be beneficial for large-scale problems, mainly in terms of the ability of dealing with larger scenario trees.

Actually, the main aim of the paper was to assess if there is some practical advantage in using a stochastic model formulation rather than a deterministic model based on expected demand values. Indeed, it should *not* be taken for granted that a more sophisticated approach works better. It is relatively easy to show that, provided the scenario tree is an exact representation of uncertainty, the expected cost of applying the solution of the stochastic model cannot be larger than the expected cost of applying the deterministic solution (in other words, the value of the stochastic solution is non-negative; see Birge and Louveaux (1997)). However, this does not imply that, in practice, this will be the case, as a limited size scenario tree is only able to model uncertainty quite roughly, and that this representation may also be affected by errors in the parameters of the assumed probability distribution, or in the distribution itself. So the real issue may only be assessed through a costly rolling horizon simulation process.

Our experiments *do not* show that modeling demand uncertainty explicitly is always worth the increase in complexity. What our experiments suggest is that stochastic programming models *may* give a significant advantage in some cases. In fact, we have built a model in the simplest setting enabling us to assess the interplay between two managerial levers when facing demand uncertainty: reactive capacity and safety stocks. In a practical setting there may be others (e.g., product modularity and late differentiation within an Assembly-to-Order environment). The stochastic programming model essentially builds safety stocks, and when reactive capacity is a sufficient buffer to deal with uncertainty the deterministic model is satisfactory. But for tight capacity constraints and large setup times, the advantage from using the stochastic programming model is significant, even though our experimental data are affected by large variability, partially due to our random mechanism for generating problem instances.

This observation possibly raises a strong objection. Our experiments are somewhat unfair since the deterministic model we have used has no way of building stock buffers, but it would be easy to introduce a constraint such as

$$I_{it} \geq I_{it}^{\min}, \tag{8}$$

requiring that the inventory level does not fall below a possibly time-varying safety stock level $I_{it}^{\min}$ for each item. However, there is no easy way to compute optimal safety stocks for a capacity constrained problem. Furthermore, there is a still subtler issue. A constraint like equation (8) is a hard constraint: this could result in a paradoxical plan in which we fail to meet demand because we keep items in inventory. To better reflect the real behaviour of a decision maker, we should relax the safety stock constraint through some penalty, but finding the right penalties may be difficult.

We have also seen that the computational effort increases significantly with the size of the scenario tree. In order to keep its size limited, good scenario generation strategies are needed. We have considered sampling approaches inspired by variance reduction strategies and an optimization approach motivated by moment matching. No clear winner emerges from our limited experiments, but Latin hypercube sampling and optimized scenario generation seem promising. The last approach is sensitive to weight setting, and may suffer from the non-convexity of the objective function. Nevertheless, it is flexible and intuitively appealing.

Again, the true answer can only be given by an extensive analysis in a specific real case, but we should close by mentioning other ideas for dealing with the scenario generation issue. One is the scenario reduction approach proposed by Heitsch and Roemisch (2003) based on the concept of the distance between probability distributions as expressed by a suitable metric. The idea is to start from a large tree and reduce it to a manageable size by keeping the approximating probability distribution close to the original one.

The shape of the scenario tree is another factor: one should consider the potential benefit of trees with high initial branching factors and a short time horizon. To this aim, we must do more work to come up with suitable terminal cost functions in order to overcome the end-of-horizon effect. It seems that a stochastic model is *per se* better suited to the task than a deterministic model, but we believe that setting the terminal inventory conditions is the most important missing piece of the puzzle in order to make mathematical programming models a really practical tool.

Acknowledgements

We gratefully acknowledge the financial support of both the Fondazione CRT (research grant for Supply Chain Management Under Uncertainty) and the Italian Ministry of University and Research (PRIN 2001 project Stochastic Mixed-integer Programming: An Application to Master Production Scheduling With Uncertain Demand, prot. 2001011353). The comments of an anonymous referee lead to an improved presentation.

References

Ahmed, S., King, A.J. and Parija, G., A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. *Journal of Global Optimization*, 2003, **26**, 3–24.

Alfieri, A., Brandimarte, P. and d'Orazio, S., LP-based heuristics for the capacitated lot-sizing problem: the interaction of model formulation and solution algorithm. *International Journal of Production Research*, 2002, **40**, 441–458.

Bakir, M.A. and Byrne, M.D., Stochastic linear optimisation of an MPMP production planning model. *International Journal of Production Economics*, 1998, **55**, 87–96.

Birge, J.R. and Louveaux, F., *Introduction to Stochastic Programming*, 1997 (Springer: Berlin).

Brandimarte, P., *Numerical Methods in Finance: A MATLAB-based Introduction*, 2001 (Wiley: New York).

Brandimarte, P., The value of the stochastic solution in a two-stage assembly-to-order problem. Submitted for publication, 2004. Available from the author upon request.

Dillenberger, C., Escudero, L.F., Wollensak, A. and Zhang, W., On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research*, 1994, **75**, 275–286.

Fisher, M., Ramdas, K. and Zheng, Y.-S., Ending inventory valuation in multiperiod production scheduling. *Management Science*, 2001, **45**, 679–692.

Fourer, R., Gay, D.M. and Kernighan, B.W., *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed., 2002 (Boyd and Fraser Publishing Company: Danvers, MA).

Guan, Y., Ahmed, S. and Nemhauser, G.L., A branch-and-cut algorithm for the stochastic uncapacitated lot-sizing problem. Technical Report, School of Industrial

& Systems Engineering, Georgia Institute of Technology, 2004. Available online at: http://www.optimization-online.org

Haugen, K.K., Løkketangen, A. and Woodruff, D.L., Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 2001, **132**, 116–122.

Heitsch, H. and Roemisch, W., Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications*, 2003, **24**, 187–206.

Hochreiter, R. and Pflug, G.Ch., Scenario tree generation as a multidimensional facility location problem. Aurora Technical Report, University of Wien, 2002. Available online at: http://www.vcpc.univie.ac.at/aurora/publications/

Hopp, W. and Spearman, M., *Factory Physics*, 2nd ed., 2000 (McGraw-Hill: New York).

Hoyland, K. and Wallace, S.W., Generating scenario trees for multistage decision problems. *Management Science*, 2001, **47**, 296–307.

Kall, P. and Wallace, S.W., *Stochastic Programming*, 1994 (Wiley: Chichester).

Kazan, O., Nagi, R. and Rump, C.M., New lot-sizing formulations for less nervous production schedules. *Computers and Operations Research*, 2000, **27**, 1325–1345.

Law, A.M. and Kelton, D.W., *Simulation Modeling and Analysis*, 3rd ed., 1999 (McGraw-Hill: New York).

Lulli, G. and Sen, S., Stochastic batch-sizing: models and algorithms. In *Network Interdiction and Stochastic Integer Programming*, edited by D.L. Woodruff, 2002 (Kluwer Academic: Boston).

Maes, J., McClain, J.O. and Van Wassenhove, L.N., Multilevel capacitated lotsizing complexity and LP-based heuristics. *European Journal of Operational Research*, 1991, **53**, 131–148.

Ponce-Ortega, J.M., Rico-Ramirez, V., Hernandez-Castro, S. and Diwekar, U.M., Improving convergence of the stochastic decomposition algorithm by using an efficient sampling technique. *Computers and Chemical Engineering*, 2004, **28**, 767–773.

Saliby, E., Descriptive sampling: a better approach to Monte Carlo simulation. *Journal of the Operational Research Society*, 1990, **41**, 1133–1142.

Sen, S., Algorithms for stochastic mixed-integer programming models. Preprint, University of Arizona, Tucson, 2003. Available online at: http://tucson.sie.arizona.edu/MORE/papers.html

Sox, C.R., Jackson, P.L., Bowman, A. and Muckstadt, J.A., A review of the stochastic lot scheduling problem. *International Journal of Production Economics*, 1999, **62**, 181–200.

Stadtler, H., Improved rolling schedules for the dynamic single-level lot-sizing problem. *Management Science*, 2000, **46**, 318–326.

Stadtler, H., Multilevel lot sizing with setup times and multiple constrained resources: internally rolling schedules with lot sizing windows. *Operations Research*, 2003, **51**, 487–502.

Stadtler, H. and Kilger, C., editors, *Supply Chain Management and Advanced Planning: Concepts, Models, Software and Case Studies*, 2nd ed., 2002 (Springer: Berlin).

Wolsey, L.A., *Integer Programming*, 1998 (Wiley: Chichester).