Scheduling Jobs Sharing Multiple Resources under Uncertainty: A Stochastic Programming Approach

Brian Keller Güzin Bayraksan*

Systems and Industrial Engineering, University of Arizona

1127 E. James E. Rogers Way, Engineering Building #20

Tucson, AZ 85721-0020

Phone: 520-621-2605, Fax: 520-621-6555

Email: kellerbd@email.arizona.edu, guzinb@sie.arizona.edu

Abstract

We formulate a two-stage stochastic integer program to determine an optimal schedule for jobs requiring multiple classes of resources under uncertain processing times, due dates, resource consumption and availabilities. We allow temporary resource capacity expansion for a penalty. Potential applications of this model include team scheduling problems that arise in service industries such as engineering consulting and operating room scheduling. We develop an exact solution method based on Benders decomposition for problems with a moderate number of scenarios. Then we embed Benders decomposition within a sampling-based solution method for problems with a large number of scenarios. We modify a sequential sampling procedure to allow for approximate solution of integer programs and prove desired properties. We compare the solution methodologies on a set of test problems. Several algorithmic enhancements are added to improve efficiency.

Keywords: stochastic integer programming, stochastic scheduling, sampling

*corresponding author

1 Introduction

We consider the problem of determining a schedule for jobs, each job requiring multiple classes of resources. We refer to this problem as multiple resource constrained scheduling problem (MRCSP). MRCSP arises naturally in many applications. For instance, companies in industries such as consulting and engineering services build teams comprised of people with different skill sets and typically do project based work. In this application, jobs correspond to different projects or products the company is developing, and the multiple resources correspond to people with different skill sets such as electrical and industrial engineers, accountants, managers, or equipment and capital constraints. Another example is operating room scheduling where the jobs correspond to operations to be performed, and the constrained resources correspond to doctors, nurses, anesthesiologists, special equipment, and the operating room. We note that while specific applications can have their own particular side constraints, in this paper, we focus on a general model.

We consider a stochastic version of the MRCSP (SMRCSP). As in real-world scheduling applications, our model allows for uncertain processing times, due dates, resource consumption, and resource availabilities. Each resource class has a capacity, although the capacity can be temporarily expanded for a penalty cost. The temporary expansions are analogous to outsourcing, hiring temporary workers, or renting equipment. Not all resources can be flexible. These hard resource constraints can be enforced by assigning a large penalty cost to the expansion.

Several models and solution approaches have been suggested for stochastic scheduling and typically fit into one of two categories: proactive or reactive scheduling (Aytug et al., 2005). Proactive scheduling constructs a predictive schedule that will perform well under a wide variety of situations. Reactive scheduling, on the other hand, takes place online at the time of job execution incorporating up-to-date information and changing the production schedule as disruptions take place. SMRCSP presented in this paper is proactive.

Closely related to the SMRCSP, stochastic resource constrained *project* scheduling in-

corporates precedence constraints on the activities or jobs that must be scheduled in order to complete a project, such as building a bridge. Precedence constraints ensure that certain activities are completed before other activities begin. The version of SMRCSP studied in this paper does not have precedence constraints since the jobs are assumed to be "projects" independent of each other except for shared resources. For instance, in operating room scheduling most patients only have one surgery at a time. There are typically no precedence constraints on the surgeries but rather a collection of independent surgeries that must be scheduled. The single resource constrained scheduling problem *without precedence constraints* has many applications studied in the literature. Chen and Lee (1999) apply the single resource constrained job scheduling problem to berth allocation where shipping vessels may occupy more than one berth for loading and unloading. Other applications include semiconductor circuit design (Lee and Cai, 1999) and management of multiprocessor computer systems (Drozdowski, 1996).

The stochastic resource constrained project scheduling problem with precedence constraints is typically solved in a two phase procedure consisting of first finding a feasible schedule and then strengthening the schedule to make it robust (van de Vonder et al., 2007; Herroelen, 2007). The initial schedule may be found by solving, either exactly or heuristically, the deterministic equivalent obtained by replacing the uncertain parameters with their average values. Schedule strengthening is accomplished by robust resource allocation (Leus and Herroelen, 2004) or by inserting buffer times into the schedule to discourage propagation of schedule disruptions (van de Vonder, 2006). Buffer insertion is similar in spirit to safe scheduling (Baker and Trietsch, 2007), which explicitly recognizes the need for safety time in order to satisfy service level constraints. Although variations of both robust resource allocation and buffer insertion time approaches could be applied to SMRCSP, it is likely the performance would suffer since the search space could not be reduced via precedence constraints. Exact procedures for stochastic resource constrained project scheduling are typically limited to a single resource class and assume exponentially distributed activity

duration disruption lengths (Leus and Herroelen, 2004). However, heuristic approaches for the multiple resource constrained versions have been developed (Deblaere et al., 2007). We propose an exact procedure for a general model with multiple resource classes and place only two easily satisfied requirements on processing time distributions, which we describe in Section 2. Further, our algorithm requires only one phase whereas project scheduling algorithms require two phases. Generating the optimal input schedule is still an open question. Our methodology can be applied to problems with any combination of uncertainty in processing times, resource consumption, and resource availability. However, existing algorithms for project scheduling are typically tailored for one type of uncertainty (Herroelen, 2007).

We model SMRCSP using a two-stage stochastic program with recourse and build a proactive and, in the terminology of Herroelen and Leus (2005), quality robust schedule. Stochastic programming has been used rarely in the case of stochastic project or job scheduling (Birge and Dempster, 1996; Denton and Gupta, 2003; Morton and Popova, 2004; Zhu et al., 2007). We use a time-indexed formulation that allows us to consider different objectives (e.g., tardiness, net present value) without any change in solution methodology. The resulting model is amenable to solution via Benders decomposition (Benders, 1962). We also add several algorithmic enhancements to further improve computation times. While this approach works well for problems with a moderate number of scenarios, many real-world applications contain a large number of scenarios making the approach intractable. Therefore, we embed Benders decomposition within a sequential sampling procedure to efficiently obtain high-quality approximate solutions.

The remainder of the paper is organized as follows. Section 2 describes the model formulation. Section 3 discusses exact solution methodology for problems with a moderate number of scenarios and provides results from the computational enhancements. Section 4 presents a Monte Carlo sampling-based solution methodology for problems with a large number of scenarios along with computational results. Section 5 concludes the paper.

2 Problem Formulation

We model SMRCSP with a *time-indexed* formulation where time is partitioned into discrete units of time t. We follow the usual convention where period t starts at $t-1$ and ends at t. Hence, many of the objectives considered below require an adjustment of -1. The time-indexed model offers two strong advantages over other formulations. However, the number of variables is usually much larger than other formulations. The first advantage of the time-indexed formulation is its flexibility. It allows modeling many different scheduling costs without changing the structure of the model, hence, *without requiring different solution algorithms*. Let t be the time period, p_j be the processing time of job j, and d_j be the due date of job j. The costs of starting job j in period t, c_{jt}, can accommodate:

- completion time ($c_{jt} = t + p_j - 1$),
- tardiness ($c_{jt} = \max\{0, t + p_j - d_j - 1\}$),
- earliness/tardiness ($c_{jt} = \max\{-t - p_j + d_j + 1, t + p_j - d_j - 1\}$),
- weighted completion, tardiness, and earliness/tardiness,
- complicated nonlinear contract penalties (set c_{jt} equal to the penalty cost paid if job j finishes in period $t + p_j - 1$),
- negative of net present value (NPV) of job completions (to "maximize" NPV).

The solution approach of the time-indexed model is independent of these cost structures – a nice property considering that specialized single machine scheduling and resource constrained project scheduling algorithms typically depend on the modeled cost. For instance, changing the objective from minimizing job tardiness to maximizing net present value for resource constrained project scheduling problems requires changing the solution algorithm (Herroelen et al., 1998). The makespan objective can be modeled by introducing a new variable that is the maximum of the completion times of all jobs, which is then minimized. This slightly alters the formulation given below. However, makespan may not be a desirable objective in applications such as scheduling consulting teams to jobs since revenues and costs typically do not depend on the makespan of all consulting jobs.

The second advantage of the time-indexed formulation is that for several different problem classes, the linear programming (LP) relaxation of the time-indexed formulation provides tight bounds on the optimal integer solution. Dyer and Wolsey (1990) and Queyranne and Schulz (1994) show that the time-indexed formulation is the strongest formulation known for the single machine scheduling problem. Chan et al. (1998) show that the optimal LP objective from a time-indexed formulation equals the optimal integer objective for some parallel machine scheduling problem classes. For time-indexed formulations of the resource constrained project scheduling problem, the LP relaxation tightens as the resource constraints become less tight (Möhring et al., 2001, 2003). We observe a similar effect for SMRCSP that leads to shorter solution times (see Section 3.4). Good LP bounds provide valuable information for solving integer programs and are critical for efficient solution of the SMRCSP. In fact, we can find a very good starting solution by solving the linear programming relaxation of the stochastic program and then applying a rounding heuristic (see Section 3.3).

In real scheduling applications, uncertainty lies mainly in processing times of jobs, resource availabilities, and resource consumptions. The formulation below can model any combination of these uncertainties without changing the implementation of the two-stage stochastic programming algorithm. First stage decisions are when to start the jobs (before processing times are observed) while the second stage decisions measure how much temporary resource expansion should be used in each scenario (recourse decisions). Notation and formulation of the stochastic program are presented below.

Indices and Sets

j	jobs to be scheduled $\quad j = 1, 2, \ldots, J$
t	time periods $\quad t = 1, 2, \ldots, T$
k	resource classes $\quad k = 1, 2, \ldots, K$
$\omega \in \Omega$	random future scenario, Ω is the set of future scenarios
$S^\omega(j,t)$	the interval of time that job j would be processed in if it finished in period t in scenario ω, $S^\omega(j,t) = \{\max\{1, t - p_j^\omega + 1\}, \ \ldots, \min\{t, T - p_j^\omega + 1\}\ \}$

Parameters

c_{jt}^{ω}	cost of starting job j in period t in scenario ω
p_j^{ω}	processing time of job j in scenario ω
p_j^{max}	$\max_{\omega \in \Omega} p_j^{\omega}$
d_j^{ω}	due date of job j in scenario ω
r_{jk}^{ω}	amount of resource from class k consumed by job j during a time period in scenario ω
R_{tk}^{ω}	total amount of resource k available during time period t in scenario ω
U_{tk}	upper limit on temporary expansion of resource k in period t
b_{tk}	per unit penalty for exceeding resource capacity R_{tk}^{ω} in time t that is within upper bound U_{tk}
B_{tk}	per unit penalty for exceeding U_{tk}
π^{ω}	probability of scenario ω
$\bar{c}_{jt}$	expected cost of starting job j in period t, $\bar{c}_{jt} = \sum_{\omega \in \Omega} \pi^{\omega} c_{jt}^{\omega}$

Decision Variables

x_{jt}	1 if job j starts in period t, 0 otherwise
z_{tk}^{ω}	amount of temporary resource expansion in time period t in scenario ω
w_{tk}^{ω}	amount of resource consumed beyond capacity $R_{tk}^{\omega} + U_{tk}$ in scenario ω

Formulation

$$\min \quad \sum_{j=1}^{J} \sum_{t=1}^{T-p_j^{max}+1} \bar{c}_{jt} x_{jt} + \sum_{\omega \in \Omega} \pi^{\omega} Q(x, \omega) \tag{1}$$

$$\text{s.t.} \quad \sum_{t=1}^{T-p_j^{max}+1} x_{jt} = 1, \qquad j = 1, \ldots, J, \tag{2}$$

$$x_{jt} \in \{0, 1\}, \qquad j = 1, \ldots, J, \quad t = 1, \ldots, T - p_j^{max} + 1, \tag{3}$$

$$\text{where } Q(x, \omega) = \min \sum_{t=1}^{T} \sum_{k=1}^{K} (b_{tk} z_{tk}^{\omega} + B_{tk} w_{tk}^{\omega}) \tag{4}$$

$$\text{s.t.} \quad z_{tk}^{\omega} + w_{tk}^{\omega} \geq \sum_{j=1}^{J} \sum_{s \in S^{\omega}(j,t)} r_{jk}^{\omega} x_{js} - R_{tk}^{\omega}, \quad t = 1, \ldots, T, \ k = 1, \ldots, K, \tag{5}$$

$$0 \leq z_{tk}^{\omega} \leq U_{tk}, \qquad t = 1, \ldots, T, \ k = 1, \ldots, K, \tag{6}$$

$$w_{tk}^{\omega} \geq 0, \qquad t = 1, \ldots, T, \ k = 1, \ldots, K. \tag{7}$$

We can write the above model more compactly as

$$\min_{x \in X} E[f(x, \omega)], \tag{8}$$

where E is the expectation operator, taken with respect to the distribution of ω, $f(x,\omega) = \left(\sum_{j=1}^{J} \sum_{t=1}^{T-p_j^{max}+1} c_{jt}^{\omega}\, x_{jt} + Q(x,\omega)\right)$, and X denotes the set of decisions that satisfies (2) and (3). Let c, b, B, $z(\omega)$, $w(\omega)$, $R(\omega)$, U, and x denote the appropriately sized vectors corresponding to $\bar{c}_{jt}$, b_{tk}, B_{tk}, z_{tk}^{ω}, w_{tk}^{ω}, R_{tk}^{ω}, U_{tk}, and x_{jt}, respectively. Then, we can write (8) as $\min_{x \in X} cx + E[Q(x,\omega)]$, where $Q(x,\omega) = \min\{qy(\omega) : \mathcal{W}y(\omega) \geq h(\omega) - \mathcal{T}(\omega)x,\ y(\omega) \geq 0\}$ with $q = [b, B]$, $y(\omega) = \left[\begin{smallmatrix} z(\omega) \\ w(\omega) \end{smallmatrix}\right]$, $\mathcal{W} = \left[\begin{smallmatrix} I & I \\ -I & 0 \end{smallmatrix}\right]$, $h(\omega) = \left[\begin{smallmatrix} -R(\omega) \\ -U \end{smallmatrix}\right]$, and $\mathcal{T}(\omega)$ is a $TK \times J(T-1) - \sum_j p_j^{max}$ matrix with consecutive elements in row tk, $t = 1, \ldots, T$, $k = 1, \ldots, K$, corresponding to the resource demands from resource class k of the jobs. In the stochastic programming literature, $E[Q(x,\omega)]$ is called the *recourse function* and $\mathcal{T}$ and $\mathcal{W}$ are called the *technology* and *recourse* matrices, respectively.

The first term in (1) is the expected cost of starting job j in period t, and the second term is the expected cost of temporary resource expansions. Constraints (2) ensure that each job is completed by time T. This model can be easily changed to a *job selection and scheduling* problem without any change in methodology by changing constraints (2) to "$\leq$" and changing the objective to maximizing some profit measure. Equations (4)-(7) model the second stage, where costs from temporary resource expansions such as outsourcing or renting are minimized. The right hand side of constraints (5) is the amount of resource expansion required in period t for class k. Temporary resource expansion is measured by z_{tk}^{ω} and cannot exceed U_{tk}. We add the auxiliary variables w_{tk}^{ω} to ensure second stage feasibility for the sampling procedure presented in Section 4. By setting B_{tk} large, we ensure that the auxiliary variables will not appear in the optimal solution unless not enough resources are present. In this case, the auxiliary variables provide managerial insight into which resources are bottlenecks.

Since the time-indexed formulation partitions time into discrete units, we impose two mild requirements. First, we require that processing time realizations with non-zero probability

mass should be natural numbers. Note that a discrete approximation may be used in place of a continuous distribution. Second, since jobs must be started such that they finish by the end of planning horizon T in all scenarios, the distribution on processing times must have a bounded support $0 \leq p_j \leq T$ for all $j = 1, \ldots, J$. In real-world applications, jobs are either completed or terminated in a finite amount of time. Therefore, it is reasonable to truncate unbounded distributions to satisfy the bounded support requirement. Note that T can be adjusted and we further investigate this issue in Section 3.4.

Before getting into the details of solution methodology, we first discuss several common scheduling problems as special cases of SMRCSP. Consider a special case of SMRCSP with a single scenario, one resource class with unit consumption and availability, i.e., $r_j = 1$, $\forall j$, $R_t = 1$ and $U_t = 0$, $\forall t$. Set B_t sufficiently large so that no optimal solution will have $w_t > 0$, $\forall t$. The resulting problem is a single machine scheduling problem. Keeping the same changes and setting $R_t = m$, $\forall t$, results in the parallel machine scheduling problem. Complexity results for single and parallel machine scheduling problems are well known (see Brucker, 2001). For instance, minimizing the sum of completion times of jobs on a single machine can be solved in polynomial time and minimizing the weighted sum of tardiness of jobs on a single machine is strongly NP-hard. The multiprocessor scheduling problem with dedicated processors is also a special case of SMRCSP. To see this, consider SMRCSP with a single scenario, $r_{jk} = 0$ or 1, $\forall j, k$, $R_{tk} = 1$, $U_{tk} = 0$, $\forall t, k$, and set B_{tk} sufficiently large so that no optimal solution will have $w_{tk} > 0$, $\forall t, k$. Following standard scheduling notation (Brucker, 2001), the problem can be described as $MPT||f$ with f representing the objective function. Minimizing the sum of completion times, $MPT||\sum C_j$, is a generalization of $MPT2||\sum C_j$ and $MPT|p_j = 1|\sum C_j$, which are known to be strongly NP-hard (Cai et al., 1998; Hoogeveen et al., 1994). Minimizing the sum of tardiness, $MPT||\sum T_j$, is a generalization of minimizing makespan, $MPT||C_{\max}$, which was shown by Kubale (1987) to be strongly NP-hard. The weighted versions of these objectives are generalizations of the non weighted versions. SMRCSP under these conditions is a generalization of $MPT||f$

and is therefore strongly NP-hard for many common objectives. The difficulty of solving the SMRCSP increases when considering more than one scenario due to stochastic data parameters.

3 Exact Solution Methodology

When the number of scenarios, $|\Omega|$, is small-to-moderate, SMRCSP can be solved exactly (i.e., within typical integer programming tolerances) in a reasonable amount of time via Benders decomposition. While the number of scenarios that allow for exact solution is problem dependent, in our computational results, we observed a slowing around 2500 scenarios. Benders decomposition is frequently referred to as the L-shaped method in the stochastic programming literature (van Slyke and Wets, 1969), and from hereon we also refer to it as the L-shaped method. In Section 3.1, we briefly review the L-shaped method for our problem. Next, in Sections 3.2 and 3.3, we describe computational enhancements aimed at improving computation times and report on the results of the enhancements. Finally, in Section 3.4, we investigate the solution characteristics of the SMRCSP under various changes in number of resource classes, planning horizon, and resource capacities.

3.1 L-Shaped Method

The L-shaped method works by decomposing the problem into one master problem and $|\Omega|$ subproblems, one for each scenario. At iteration i of the L-shaped method, the master problem is solved to obtain feasible first stage decision variables. Then, the second stage problem for each scenario is evaluated using the x provided by the master problem. Dual information from the second stage problems are used to form a cut that is added to the first stage problem. The cut either improves the piecewise linear lower approximation to the recourse function, resulting in an *optimality cut* or cuts off an infeasible first stage solution, resulting in a *feasibility cut*. Since SMRCSP given in (1)-(7) has relatively complete recourse

(i.e., second stage problems are always feasible for any given x), only optimality cuts must be added.

The master problem can be written as

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T-p_j^{max}+1} \bar{c}_{jt} x_{jt} + \theta \tag{9}$$

$$\text{s.t.} \quad \text{constraints (2), (3),}$$

$$-G_i x + \theta \geq g_i, \qquad i = 1, \ldots, I, \tag{10}$$

where θ is a continuous variable corresponding to the expected value of the second stage objective, I is the number of cuts generated so far, and G_i is the cut gradient and g_i is the cut intercept for cut $i = 1, \ldots, I$. Here, we present a single-cut version of the algorithm where cuts from a given scenario are aggregated: $g_i = \sum_{\omega \in \Omega} \pi^\omega g_i^\omega$, $G_i x = \sum_{\omega \in \Omega} \pi^\omega G_i^\omega x$. In our computational tests, we experimented with the multi-cut version (see, e.g., Birge and Louveaux (1997)) and found it to be too slow for this problem. Hence, discussion and computations are based on the single-cut version. Subproblems for a given first stage x and scenario ω are given in (4)-(7). Let α_{tk}^ω be the optimal dual variables corresponding to constraints (5) and β_{tk}^ω be the dual variables corresponding to constraints (6). The optimal dual solution results from one of the three following cases:

1. If $\sum_{j=1}^{J} \sum_{s \in S^\omega(j,t)} r_{jk}^\omega x_{js} \leq R_{tk}^\omega$, then $\alpha_{tk}^\omega = 0$ and $\beta_{tk}^\omega = 0$.

2. If $R_{tk}^\omega < \sum_{j=1}^{J} \sum_{s \in S^\omega(j,t)} r_{jk}^\omega x_{js} \leq R_{tk}^\omega + U_{tk}$, then $\alpha_{tk}^\omega = -b_{tk}$ and $\beta_{tk}^\omega = 0$.

3. If $\sum_{j=1}^{J} \sum_{s \in S^\omega(j,t)} r_{jk}^\omega x_{js} > R_{tk}^\omega + U_{tk}$, then $\alpha_{tk}^\omega = -B_{tk}$ and $\beta_{tk}^\omega = b_{tk} - B_{tk}$.

Then, the optimality cut is obtained by

$$g_i^\omega = \sum_{t=1}^{T} \sum_{k=1}^{K} \left(\alpha_{tk}^\omega R_{tk}^\omega + \beta_{tk}^\omega U_{tk} \right), \tag{11a}$$

$$G_i^\omega x = \sum_{t=1}^{T} \sum_{k=1}^{K} \sum_{j=1}^{J} \sum_{s \in S^\omega(j,t)} \alpha_{tk}^\omega r_{jk}^\omega x_{js}. \tag{11b}$$

The dual of the SMRCSP second stage problem can be solved directly without requiring an LP solution algorithm. Hence, (11a) and (11b) are calculated very efficiently.

3.2 Computational Enhancements

Even though the L-shaped method enables efficient solution of SMRCSP, several computational enhancements were added to further decrease computation time. These are: (i) trust regions, (ii) GUB branching, (iii) integer cutting planes on the optimality cuts, (iv) warm-starting with the LP solution, and (v) approximate solution of the master problem. Below, we explain these in more detail, and in Section 3.3 we provide the results of our computational tests on the effectiveness of these enhancements.

Trust Regions: In early iterations of the L-shaped method, the solution often oscillates wildly from one iteration to the next thereby slowing convergence. For continuous problems, a penalty can be added to the objective corresponding to the l_2 norm of the new master problem solution from the previous solution. The penalty encourages solutions to remain closer to each other from one iteration to the next. This method is known as regularized decomposition, and has been frequently used (Higle and Sen, 1996; Hiriart-Urruty and Lemaréchal, 1993; Kiwiel, 1990; Ruszczyński, 1986). Implementing regularized decomposition on a stochastic integer program would result in a quadratic integer master problem. We can avoid solving a quadratic integer program while reducing oscillation by instead implementing a trust region. The trust region is a set of constraints that defines a region around the previous solution such that the next solution must lie within this region. Trust regions have been used successfully in stochastic linear programs (Linderoth and Wright, 2003). However, since the SMRCSP decision variables are binary, the l_∞ norm typically used for continuous problems would be meaningless. Instead, we implement two different trust regions and compare their effectiveness.

The first trust region limits the number of first stage variables that can be changed from iteration to iteration, similar to the one implemented by Santoso et al. (2005), called the Hamming distance. Suppose that x^i is the master problem solution at the i^{th} iteration. Let $X^i = \{(j,t) : x^i_{jt} = 1\}$. Then, the Hamming distance trust region is

$$\sum_{(j,t)\in X^i} (1 - x_{jt}) + \sum_{(j,t)\notin X^i} x_{jt} \leq \Delta^i, \tag{12}$$

where Δ^i denotes the limit on the number of variables that can be changed from iteration i to $i+1$. The Hamming distance is a versatile trust region that can be applied to most binary master problems. The second trust region is geared towards our formulation and limits the change in start times for each job from one iteration to the next. Let τ_j be the start time of job j in the previous iteration. Then, the trust region based on job start times is

$$-\Delta^i \leq \sum_{t=1}^{T-p_j+1} t x_{jt} - \tau_j \leq \Delta^i, \quad j = 1, \ldots, J, \tag{13}$$

where Δ^i is the permitted deviation in start times from one iteration to the next.

Intelligent choices for the size of the trust region can make it more effective. In our implementation, we set $\Delta^1 = \lceil \frac{J}{2} \rceil$ for (12), so that initially we allow half of the jobs to change start times. Similarly, we set $\Delta^1 = \lceil \frac{T}{2} \rceil$ for (13), which restricts job start times to half of the horizon. If the objective evaluated at x^i is greater than the upper bound, we shrink the trust region by setting $\Delta^{i+1} = \lceil 0.7\Delta^i \rceil$. If both the objective at x^i is less than or equal to the upper bound and any trust region constraint is binding, then we expand the trust region by setting $\Delta^{i+1} = \lceil 1.3\Delta^i \rceil$. 0.7 and 1.3 were chosen after a little experimentation so that the trust region would not expand or contract too quickly and lose effectiveness. Convergence is not guaranteed when using trust regions with integer master problems. However, since oscillation is most prominent in early iterations, we remove the trust regions once the L-shaped optimality gap has reached a specified threshold.

GUB Branching: Constraints (2) are known as generalized upper bound (GUB) constraints since only one variable in each constraint can be set to 1. Define the GUB set for job

j as $G_j = \{(j,t) : 1 \leq t \leq T - p_j^{max} + 1\}$. When solving the master problem, the standard branching rule is to split the branch and bound tree into one partition with $x_{jt}^{LP} = 0$ and another partition with $x_{jt}^{LP} = 1$ for some fractional x_{jt}^{LP}, $(j,t) \in G_j$. However, this leads to an unbalanced tree since there are roughly $T-1$ possibilities in G_j for nonzero variables in the first branch and only one in the second branch. Let $t_1, t_2, \ldots, t_k$ be some ordering of the fractional variables in G_j. Realizing that variables in G_j must sum to 1, we can create one branch with $x_{jt_i} = 0,\ i = 1, \ldots, r$ and another branch with $x_{jt_i} = 0,\ i = r+1, \ldots, k$, where $r = \min\{l : \sum_{i=1}^{l} x_{jt_i}^{LP} \geq 1/2\}$. This leads to a more balanced tree and can greatly improve solution times (Wolsey, 1998).

Integer Cuts on Optimality Cuts: The optimality cuts for our problem, given in (11), are the same as knapsack constraints with one continuous variable. One way to improve performance is to add integer cuts on the optimality cuts. Marchand and Wolsey (1999) and Miller et al. (2000) have derived families of facet defining cutting planes for continuous knapsack problems. The continuous knapsack problem is characterized by the set

$$Y = \left\{ (y,s) \in \mathcal{B}^n \times \mathcal{R}_+^1 : \sum_{j \in N} a_j y_j \leq b + s \right\}, \tag{14}$$

where $N = \{1, \ldots, n\}$ is the set of elements in the knapsack, $a_j > 0$, $j \in N$, and $b > 0$. For the optimality cuts of SMRCSP, x_{jt} correspond to y_j, the cut gradient G to $[a_1, \ldots, a_j]$, cut intercept g to b, and θ to s in (14). An (i, C, T) cover pair is defined by an index i, set C, and set T such that $C \cap T = i$, $C \cup T = N$, $\lambda = \sum_{j \in C} a_j - b > 0$, and $a_i > \lambda$. The cuts take the form

$$\sum_{j \in C} \min\{\lambda, a_j\} y_j + \sum_{j \in T\ i} \phi_C(a_j) y_j \leq \sum_{j \in C\ i} \min\{\lambda, a_j\} + s, \tag{15}$$

and details of $\phi_C(\cdot)$ can be found in the references above. We implement cuts (15) and provide results in the following section.

LP Warm-Starting: We can find a good starting solution by solving the linear programming relaxation of the stochastic program and then applying a rounding heuristic. We experimented with several heuristics and found the following to be the most effective in our

tests.

Step 1. Solve LP relaxation. Get fractional solution x^{LP}.

Step 2. For each variable, if $x_{jt}^{LP} = 0$, then remove variable from formulation. If $x_{jt}^{LP} = 1$, then start job j in period t. Subtract its effects from the GUB constraints and optimality cuts, and remove variable from formulation.

Step 3. The remaining x_{jt}^{LP} are the fractional variables from x^{LP}. Solve the reduced-size IP to find an integer solution x^{IP}.

Since the second stage problems are continuous, the optimality cuts generated in solving the LP solution are valid for the integer program and provide valuable information on the shape of the recourse function. In our computations, we noticed that the integer master problem took longer to solve at each L-shaped iteration as more constraints were present. We experimented with keeping all cuts from the LP solution and dropping different portions of the cuts and found that dropping the first half of optimality cuts resulted in the best performance. As will be seen below, the LP warm-start facilitates the solution of problems otherwise unsolvable in the given time limit.

Approximate Master Solve: Most of the computational effort is spent solving the master problem. At the early iterations of the L-shaped method, the master problem solutions are often far away from the optimal solutions. Computational effort may be saved if the master problem is solved to within ε-optimality, $\varepsilon \geq 0.1\%$. We resume the exact solution of the master problem when the optimality gap is small enough or if the same master problem solution is generated in consecutive iterations.

Implementation Details: Since the subproblems are efficiently solved, the enhancements discussed above are mainly aimed at improving master problem solution times. L-shaped optimality cuts were added to the master problem after it was solved to integer optimality, except when using the approximate master solve enhancement. In this case, optimality cuts were added once a 1% optimal solution was found. The continuous knapsack cuts were implemented on the master problem within a branch-and-cut framework, generating and adding cuts at each branch-and-bound node. Cut generation was discontinued when

the ratio of cuts added to number of searches fell below 10% to prevent spending time searching for cuts when good cuts were not likely to be found. Continuous knapsack cuts generated in previous L-shaped iterations were retained for future iterations. The GUB branching was implemented using the SOS1 feature in CPLEX 10.1. We used LP warm-starting only in initialization of the L-shaped method to provide a set of optimality cuts and a good starting solution for the MIP solver. The trust regions were used on the MIP master problems and removed once the L-shaped optimality gap fell below 5%.

3.3 Computational Results

The L-shaped method with various computational enhancements was tested on a set of 9 test problems to find the best combination of enhancements. The generated problems are described by a series of numbers A-B-C, where A is the number of jobs (20, 40, or 80), B is the number of resource classes (5), and C denotes the problem instance for an A-B combination. The planning horizon is set to 50 time periods for all problems. Parameters are generated as described in Table 1. Each problem has 10 jobs with two possible processing times. The remaining jobs in each problem have deterministic processing times resulting in a total of $2^{10} = 1024$ scenarios. We explore much larger problems in Section 4.2. For jobs with uncertain processing times, the first processing time is generated from integer uniform(1,50). The second processing time is set to the first processing time plus 5. If this results in a processing time larger than T, we instead set the second processing time equal to the first processing time minus 5. We set $R_{tk}^{\omega} = R_k$ for all $t = 1, \ldots, T$ and $\omega \in \Omega$ and $r_{jk}^{\omega} = r_{jk}$ for all $\omega \in \Omega$, restricting uncertainty to processing times only. R_k, given in Table 1, ensures that R_k is exceeded by a small amount in most time periods. Costs, c_{jt}^{ω}, are set to tardiness. Termination criteria is an optimality gap of less than 1% or a five hour time limit. The algorithms were coded in C++ using the CPLEX 10.1 callable library running on a 900 MHz UltraSPARC-III with 4 GB of memory.

Table 2 compares the results of the enhancements to plain L-shaped method. In Table

parameter	generation scheme	parameter	generation scheme
d_j	U(1,10)	p_j	U(1,50)
r_{jk}	U(1,5)	R_k	between $\bar{p}\bar{r}_k J/T$
b_k	U(1,10)		and $\bar{p}\bar{r}_k J/(0.6T)$
B_k	$\max\{2b_k, 10\}$	U_k	$\lceil 0.1R_k \rceil$

Table 1: Description of how the test problem parameters were generated. $\bar{p}$ is the average processing time for the problem instance, $\bar{r}_k$ is the average resource consumption for resource K.

2, we only provide results from the 20 job problems for brevity and present a summary for all problems in Table 3. All methods enhanced plain L-shaped performance except for the integer cuts on the optimality cuts. Regular knapsack cuts in the presence of GUB constraints are generally not facet defining unless certain conditions are met (Wolsey, 1990). We believe that the GUB constraints present in SMRCP affect the continuous knapsack cuts in a similar manner thereby reducing their effectiveness. The LP warm-start and approximately solving the master problem resulted in the most improvement. These two enhancements are even more effective when combined with the job start trust region and GUB branching. Table 3 shows the results of this combination on problems with up to 80 jobs. The enhancement combination solves 8 of the 9 test problems to the termination criteria of 1% optimality within the five hour time limit. We also tested the enhancements on objectives of minimizing total completion time and total lateness. The results are similar to Tables 2 and 3 and are not presented for the sake of brevity.

3.4 Further Analysis

In this section, we investigate (i) what makes a problem instance difficult to solve and (ii) characteristics of the optimal solutions.

Instance Difficulty: The computational effort required to solve the problems can change dramatically as we vary the resource demands and capacities, planning horizon, and number of jobs. The analysis here provides insight into what makes a problem instance difficult to solve. We have found that the patterns in solution times are best explained by the resource density, ρ. For the test problems, $R_{tk}^{\omega} = R_k$ for all $t = 1, \ldots, T$ and $\omega \in \Omega$ and

	No Enhancements			Hamming			Job Start			GUB		
problem	time	it	gap	time	it	gap	time	it	gap	time	it	gap
20-5-1	18000	203	1.53%	18000	192	1.39%	11525	205	0.98%	7533	219	0.99%
20-5-2	18000	123	6.58%	18000	109	5.80%	18000	151	3.78%	13784	197	0.92%
20-5-3	18000	72	25.59%	18000	71	7.73%	18000	107	6.84%	18000	157	13.03%

	GUB + Hamming			GUB + Job Start			Cuts				
problem	time	it	gap	time	it	gap	time	it	gap	ncuts	ratio
20-5-1	3069	174	0.99%	5313	195	0.99%	18000	208	1.18%	1573	0.09999
20-5-2	9210	158	0.92%	5900	168	0.92%	18000	121	8.87%	1804	0.09999
20-5-3	18000	166	7.41%	18000	183	5.06%	18000	72	25.59%	943	0.09999

	LP					Approx Master		
problem	LP time	IP time	total time	IP it	gap	time	it	gap
20-5-1	382	536	918	47	0.99%	5747	210	0.98%
20-5-2	283	18000	18283	117	2.08%	10362	182	0.92%
20-5-3	813	18000	18813	97	1.84%	18000	212	6.36%

Table 2: Effectiveness of the computational enhancements.

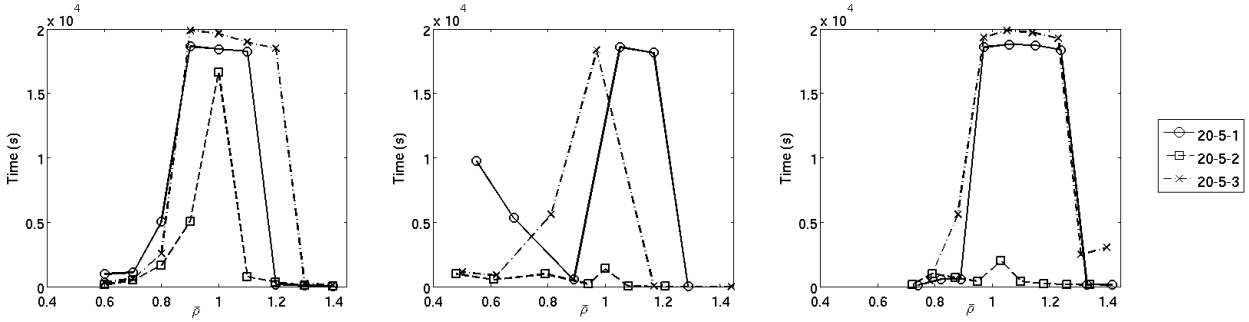
	No Enhancements			GUB + Job Start + LP + Approx Master				
problem	time	it	gap	LP time	IP time	total time	IP it	gap
20-5-1	18000	203	1.53%	386	156	542	27	0.87%
20-5-2	18000	123	6.58%	287	715	1002	71	0.92%
20-5-3	18000	72	25.59%	189	5435	5624	121	0.81%
40-5-1	18000	53	9.68%	1576	18000	19576	56	3.05%
40-5-2	10237	217	0.73%	911	1638	2549	97	0.98%
40-5-3	18000	132	3.24%	950	2332	3282	92	0.99%
80-5-1	18000	21	13.38%	1762	4162	5924	162	0.99%
80-5-2	18000	47	1.27%	973	86	1059	3	0.42%
80-5-3	18000	38	17.18%	3263	1185	4448	14	0.92%
total	**154,237**					**44,006**		

Table 3: Improvements in running time compared to plain L-shaped method due to computational enhancements.

$r^{\omega}_{jk} = r_{jk}$ for all $\omega \in \Omega$. As such, we define the density of resource class k as

$$\rho_k = \frac{\bar{p}\bar{r}_k J}{TR_k}, \tag{16}$$

where $\bar{p} = \frac{1}{J}\sum_{j=1}^{J} \bar{p}_j$ is the average processing time for all jobs and $\bar{r}_k = \frac{1}{J}\sum_{j=1}^{J} r_{jk}$ is the average resource consumption of class k. The numerator is a rough estimate of the total resource consumption, and the denominator is the total amount of resource k available over the planning horizon T. For $\rho_k < 1$ we expect to have more resources available than consumption over the horizon and for $\rho_k > 1$ we expect to have more consumption than



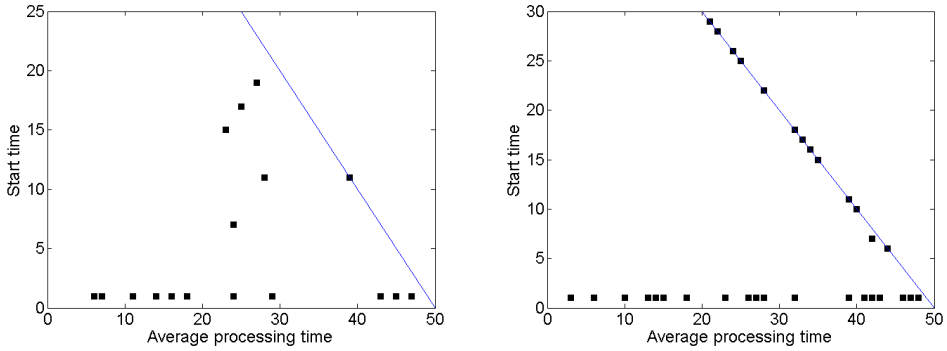
(a) changing $\bar{r}_k/R_k$ (b) changing T (c) changing J

Figure 1: Solution times are strongly influenced by $\bar{\rho}$. The figures show solution times when $\bar{\rho}$ is varied by changing $\bar{r}_k/R_k$, T, and J, respectively.

resource available.

Figure 1 shows the sensitivity of solution times to $\bar{\rho} = \frac{1}{K}\sum_{k=1}^{K}\rho_k$. In Figure 1a, we changed $\bar{\rho}$ by only changing the ratio $\bar{r}_k/R_k$. In Figure 1b, we changed $\bar{\rho}$ by adjusting the planning horizon T by increments of 10, and in Figure 1c, we changed $\bar{\rho}$ by solving versions of the problems with 16 to 34 jobs while fixing all other parameters. For $\bar{\rho} \leq 0.8$, there is enough resource capacity so second stage costs are negligible. The optimization then minimizes the first stage costs while keeping resource consumption below capacity. On the other hand, for $\bar{\rho} \geq 1.3$, resource capacities almost always require expansion. In this case, first stage costs are dominated by second stage costs and the optimal solution is quickly found. For values close to 1 the problems are difficult to solve. In this case, there is a delicate balance between start times of jobs (first stage) and resource capacity expansions (second stage), and the algorithm spends considerable time trying to minimize total cost. Note that in Figure 1b, T increases as $\bar{\rho}$ decreases. Larger T values result in larger problem sizes accounting for the increase in solution times for the smallest $\bar{\rho}$ values.

As $\bar{\rho}$ increases, the computational difficulty of the problem transitions from easy to hard to easy. These *phase transitions* have been observed in several NP-hard problems including resource constrained project scheduling (Herroelen and Reyck, 1999). Existing measures of resource consumption and availability include the resource factor (RF) (Pascoe, 1966), resource strength (RS) (Cooper, 1976), and resource constrainedness (RC) (Patterson,



(a) ample resources, $\bar{\rho} = 0.79$ (b) tight resources, $\bar{\rho} = 0.97$

Figure 2: Problems with small resource expansion display a peak at medium processing times whereas problems with larger resource expansion display an angular schedule.

1976). RF is the average portion of resources requested per activity (job). However, it does not consider processing times of the activities. RS utilizes precedence-based early start schedules to determine a ratio of resource availability to peak consumption. While RS can be calculated for SMRCSP, the measure may lose some meaning since SMRCSP lacks precedence constraints. RC measures the ratio of average resource consumption of activities to resource availability. Our measure, which we refer to as the resource density (RD), is similar to RC but we capture total consumption and availability (rather than average). Hence, we incorporate the planning horizon T into RD as the horizon has an effect on resource availability.

Solution Characteristics: We now examine observed patterns in the optimal solutions. Figure 2 displays optimal job start times versus average processing times for two representative test problems. The lines indicate the latest time in which a job can start $(T - p_j^{max} + 1)$. The solution in Figure 2a corresponds to a problem with $\bar{\rho} = 0.79$, indicating that only small resource expansions are necessary for this schedule. In this case, solutions tend to start jobs with small and large processing times early but medium processing times later, resulting in a peak pattern around the middle. The solution in Figure 2b corresponds to a problem with $\bar{\rho} = 0.97$, indicating tight resources and larger resource expansion for this schedule. Solutions in this case display an angular structure where jobs are either started as early as possible or as late as possible. The optimization tries to minimize the cost of resource expansion and

does this by scheduling jobs as early or late as possible. Examining problems with large values of $\bar{\rho}$ (e.g., $\bar{\rho} = 1.3$), we again observed the angular pattern. Patterns in solutions have been observed in other scheduling problems. For instance, Denton and Gupta (2003) observed that scheduled appointment lengths are initially increasing and then decreasing as the schedule progresses in stochastic appointment scheduling with identically distributed (i.i.d) appointment processing times.

4 Sampling-Based Approximate Solution Methodology

For problems with a moderate number of scenarios, the exact solution method of the L-shaped method can be used. However, modeling even small problems can result in an intractable number of scenarios. For example, a problem with 10 independent jobs where each job has 5 processing times results in $5^{10} = 9,765,625$ scenarios, which is too large to be solved exactly. In this case, we approximate SMRCSP with a sample average.

Let z^* be the optimal objective function value of the SMRCSP, i.e., $z^* = \min_{x \in X} E[f(x, \omega)]$, recall (8). When the number of scenarios is large, this problem becomes intractable and can be approximated by sampling from the distribution of $\omega \in \Omega$ resulting in a sampling approximation

$$z_n^* = \min_{x \in X} \frac{1}{n} \sum_{i=1}^{n} f(x, \omega^i), \tag{17}$$

with an optimal solution x_n^*. While other sampling schemes are possible, we assume $\omega^1, \omega^2, \ldots, \omega^n$ are i.i.d as $\omega \in \Omega$. As $n \to \infty$, z_n^* converges to z^* and similar consistency properties exist for x_n^*; see e.g., the survey by (Shapiro, 2003). For SMRCSP, under i.i.d sampling, since $|X| = JT$ is finite, consistency for z_n^* and x_n^* is achieved when $E[|f(x, \omega)|] < \infty$ for all $x \in X$, see Proposition 2.1 of (Kleywegt et al., 2001). This condition is satisfied for SMRCSP, for instance, when the random parameters have bounded support. However, practical implementations need a finite sample size and a reliable means to stop sampling. Bayraksan and Morton (2008) provide rules to increase sample sizes and to stop a sequential sampling procedure. In this section, we present a modified version of this procedure, show that sim-

ilar desired theoretical properties hold for the modified version and present computational results for SMRCSP.

4.1 Sequential Sampling

The idea behind sequential sampling is simple. First, a candidate solution is generated. This is typically done by solving a sampling problem (17) with increasing sample size. Then, the quality of this candidate solution is evaluated. That is, a statistical estimator of the optimality gap of the candidate solution is formed. If the optimality gap estimate falls below a desired value, then the procedure stops. Otherwise, the procedure continues with a larger sample size. Per iteration, the procedure typically requires solution of at least two sampling problems, one for generating the candidate solution and at least one more for calculating the statistical estimator of the optimality gap. For integer programs, solution of these optimization problems can easily become burdensome. Since through sequential sampling, we aim to find a quality *approximate* solution, we can solve these sampling problems approximately, saving considerable computational effort. Our modifications allow this. In particular, when we solve the sampling problems approximately (e.g., 2.5% optimality), our observed quality for the solution obtained is slightly lower (e.g., within 3% optimality) since the modified statistical gap estimate includes the optimality gap of the sampling problem in addition to sampling error.

First, we introduce some relevant notation. Let X^* denote the set of optimal solutions to SMRCSP. Similarly, let $X^*(\delta)$ denote the set of $100\delta\%$-optimal solutions, i.e., $X^*(\delta) = \{x \in X : E[f(x,\omega)] - z^* \leq \delta|z^*|\}$ with $\delta > 0$. We also use the notation $x_n^*(\delta)$ to denote a $100\delta\%$-optimal solution to (17). Note that $x_n^*(0) = x_n^*$. For any $x \in X$, let $\mu_x = Ef(x,\omega) - z^*$ and $\sigma^2(x) = \text{var}[f(x,\omega) - f(x_{\min}^*(\delta),\omega)]$, where $x_{\min}^*(\delta) \in \arg\min_{y \in X^*(\delta)} \text{var}[f(x,\omega) - f(y,\omega)]$. In words, μ_x denotes the optimality gap of x, and $\sigma^2(x)$ denotes the "minimum" variance of the difference random variables, $f(x,\omega) - f(y,\omega)$ over $y \in X^*(\delta)$.

Optimality Gap Estimators: Given a candidate solution $x \in X$, we denote the point estimator of the optimality gap, μ_x, as $G_n(x)$ and the point estimator of the variance, $\sigma^2(x)$, as $s_n^2(x)$. To obtain these estimators, we modify the single replication procedure (SRP) developed in (Bayraksan and Morton, 2006) such that the sampling problem is solved to $100\delta\%$ optimality. Let $\bar{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} f(x,\omega^i)$. The resulting estimators are:

$$G_n(x) = \bar{f}_n(x) - \bar{f}_n(x_n^*(\delta)) \tag{18a}$$

$$s_n^2(x) = \frac{1}{n-1}\sum_{i=1}^{n}\left[(f(x,\omega^i) - f(x_n^*(\delta),\omega^i)) - G_n(x)\right]^2. \tag{18b}$$

Note that in (18) a sampling problem (17) with sample size n is solved approximately (e.g., via L-shaped) to obtain $x_n^*(\delta)$, and the same set of observations is used in all terms in (18) above. Since the sampling problem is solved to $100\delta\%$ optimality, we do not know z_n^*, rather, we know upper and lower bounds on it, i.e., $\underline{z}_n^*(\delta) \leq z_n^* \leq \overline{z}_n^*(\delta)$. Note that with the notation above, we can equivalently write $z_n^* = \bar{f}_n(x_n^*)$ and $\overline{z}_n^*(\delta) = \bar{f}_n(x_n^*(\delta))$. We point out that estimators in (18) are different than ε-optimal SRP discussed in (Bayraksan and Morton, 2006). In ε-optimal SRP, lower bounds, $\underline{z}_n^*(\delta)$, are used instead of upper bounds, $\overline{z}_n^*(\delta)$, in the second term of (18a). ε-optimal SRP is designed for very small ε , while δ is typically larger to allow for efficient solution of stochastic integer programs. With large δ, when lower bounds are used, the estimators become overly conservative and the sequential sampling procedure does not stop in a reasonable amount of time. With the use of upper bounds, (18a) now underestimates μ_x and we correct this by appropriately inflating the confidence interval on the candidate solution found.

Outline of the Procedure: At iteration $l \geq 1$, we select sample sizes, m_l and n_l, and use m_l to generate a candidate solution, denoted $\hat{x}_l$, and use n_l to evaluate the quality of this solution. To generate $\hat{x}_l$, we solve a sampling problem with m_l observations to $100\delta_1^l\%$ optimality with $0 \leq \delta_1^l < 1$ such that $\delta_1^l \downarrow 0$ as $l \to \infty$ and set $\hat{x}_l = x_{m_l}^*(\delta_1^l)$. To evaluate $\hat{x}_l$'s quality, we solve a sampling problem with n_l observations to $100\delta_2\%$ optimality, $0 \leq \delta_2 < 1$, to obtain the estimators given in (18). In our implementation, we set $\delta_1^1 = \delta_2$ and divide δ_1^l

Input:	Values for $h > h' > 0$, $\epsilon > \epsilon' > 0$, $0 < \alpha < 1$, $p > 0$ and $0 \leq \delta_1^l < 1$ such that $\delta_1^l \downarrow 0$ as $l \to \infty$ and $0 \leq \delta_2 < 1$.
Output:	A candidate solution, $\hat{x}_L$, and a $(1-\alpha)$-level confidence interval on μ_L.
Step 1.	(Initialize) Set $l = 1$, calculate n_l as given in (21) and set $m_l = 2 \cdot n_l$. Sample i.i.d observations $\omega_1^1, \omega_1^2, \ldots, \omega_1^{m_l}$ and independently sample i.i.d observations $\omega_2^1, \omega_2^2, \ldots, \omega_2^{n_l}$ from the distribution of ω.
Step 2.	(Generate Candidate Solution) Use $\omega_1^1, \omega_1^2, \ldots, \omega_1^{m_l}$ to solve a sampling problem to $100\delta_1^l\%$ optimality to obtain $x^*_{m_l}(\delta_1^l)$. Set $\hat{x}_l = x^*_{m_l}(\delta_1^l)$.
Step 3.	(Assess Solution Quality) Given $\hat{x}_l$, use $\omega_2^1, \omega_2^2, \ldots, \omega_2^{n_l}$ to solve a sampling problem to $100\delta_2\%$ optimality to form G_l and s_l^2, given in (18).
Step 4.	(Check Stopping Criterion) If $\{G_l \leq h's_l + \epsilon'\}$, then set $L = l$, and go to 6.
Step 5.	(Increase Sample Size) Set $l = l + 1$ and calculate n_l according to (21). Set $m_l = 2 \cdot n_l$. Sample $m_l - m_{l-1}$ i.i.d observations, $\omega_1^{m_{l-1}+1}, \ldots, \omega_1^{m_l}$, and independently sample $n_l - n_{l-1}$ i.i.d observations $\omega_2^{n_{l-1}+1}, \ldots, \omega_2^{n_l}$ from the distribution of ω. Update δ_1^l. Go to 2.
Step 6.	(Output Approximate Solution) Output candidate solution $\hat{x}_L$ and a one-sided confidence interval on μ_L, $\left[0,\ hs_T + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right]$.

Figure 3: Sequential sampling procedure to obtain quality approximate solutions as implemented in our computational results.

by 2 if the procedure does not stop for the next 25 iterations. We augment the observations from the previous iteration by generating $m_l - m_{l-1}$ and $n_k - n_{k-1}$ additional independent observations. To simplify notation, from now on, we drop the dependence on the candidate solution, $\hat{x}_l$, and the sample size n_l, and simply denote $\mu_l = \mu_{\hat{x}_l}$, $\sigma_l^2 = \sigma^2(\hat{x}_l)$, $G_l = G_{n_l}(\hat{x}_l)$ and $s_l = s_{n_l}(\hat{x}_l)$.

A brief algorithmic statement of the sequential sampling procedure, as implemented in our computational results, is given in Figure 3. The procedure terminates the first time G_l falls below $h's_l > 0$ plus a small positive number ϵ', i.e.,

$$L = \inf_{l \geq 1} \{l : G_l \leq h's_l + \epsilon'\}. \tag{19}$$

Let $h > h' > 0$ and $\epsilon > \epsilon' > 0$ (typically, epsilon terms are small, e.g., around 10^{-7}). When the algorithm terminates with approximate solution $\hat{x}_L$, a confidence interval on its optimality gap, μ_L, is given by

$$[0,\ hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)], \tag{20}$$

where $\overline{z}^*_{n_L}(\delta_2) = \bar{f}_{n_L}(x^*_{n_L}(\delta_2))$ is obtained from G_L (see (18a)). We show below this confidence interval (CI) is asymptotically valid when the sample sizes n_l satisfy

$$n_l \geq \left(\frac{1}{h-h'}\right)^2 \left(c_p + 2p\ln^2 l\right), \tag{21}$$

where $c_p = \max\{2\ln\left(\sum_{j=1}^{\infty} j^{-p\ln j}/\sqrt{2\pi}\alpha\right), 1\}$. Here, $p > 0$ is a parameter that affects the number of samples we generate. See (Bayraksan and Morton, 2008) for more details on the parameters.

Theoretical Properties: Two desired properties are: (i) the procedure stops in a finite number of iterations with probability one (w.p.1) and (ii) terminates with a quality solution with a desired probability. The theorem below summarizes conditions under which these properties hold.

Theorem 1. *Suppose $X \neq \emptyset$, $|X| < \infty$, $E\sup_{x\in X}|f(x,\omega)| < \infty$ and that the distribution of ω has a bounded support. Let $\omega_i^1, \omega_i^2, \ldots$, $i = 1, 2$, be i.i.d as ω. Let $\epsilon > \epsilon' > 0$, $p > 0$, $0 < \alpha < 1$ and $0 \leq \delta_2 < 1$ be fixed and let $0 \leq \delta_1^l < 1$ be such that $\delta_1^l \downarrow 0$ as $l \to \infty$. Consider the sequential sampling procedure where n_l is increased according to (21), $m_l \to \infty$ as $l \to \infty$ and the optimality gap estimators are calculated according to (18). If the procedure stops at iteration L according to (19) then, (i) $P(L < \infty) = 1$, and (ii) $\liminf_{h\downarrow h'} P\left(\mu_L \leq hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right) \geq 1 - \alpha$.*

Proof. (i)
$$\begin{aligned} P(L=\infty) &\leq \liminf_{l\to\infty} P(G_l > h's_l + \epsilon') \\ &\leq \liminf_{l\to\infty} P(\bar{f}_{n_l}(\hat{x}_l) - \bar{f}_{n_l}(x^*_{n_l}) > \epsilon') \qquad (22) \\ &\leq \liminf_{l\to\infty} P(|\bar{f}_{n_l}(\hat{x}_l) - \bar{f}_{n_l}(x^*_{n_l}) - \mu_l| > \epsilon' - \mu_l), \qquad (23) \end{aligned}$$

where (22) follows from $h's_l > 0$, and $G_l \leq \bar{f}_{n_l}(\hat{x}_l) - \bar{f}_{n_l}(x^*_{n_l})$. The right hand side of (23) is zero, since $\bar{f}_{n_l}(x)$ converges to $E[f(x,\omega)]$ uniformly in X; $\bar{f}_{n_l}(x^*_{n_l}) = z^*_{n_l}$ converges to z^*; and $\mu_l \downarrow 0$ w.p.1, for $|X|$ finite, as $l \to \infty$, see Proposition 2.1 of (Kleywegt et al., 2001). (ii) Let $\Delta h = h - h'$, $\Delta\epsilon = \epsilon - \epsilon'$ and define $D_l = \bar{f}_{n_l}(\hat{x}_l) - \bar{f}_{n_l}(x^*_{\min}(\delta_2))$, where

$x^*_{\min}(\delta_2) \in \arg\min_{y \in X^*(\delta_2)} \text{var}[f(\hat{x}_l, \omega) - f(y, \omega)]$. Note that

$$\begin{aligned} P\left(\mu_L > hs_L + \epsilon + \delta_2 \overline{z}^*_{n_L}(\delta_2)\right) &\leq P\left(\mu_L \geq G_L + \delta_2 \overline{z}^*_{n_L}(\delta_2) + \Delta h s_L + \Delta\epsilon\right) \\ &\leq \sum_{l=1}^{\infty} P\left(G_l + \delta_2 \overline{z}^*_{n_l}(\delta_2) - \mu_l \leq -\Delta h s_l - \Delta\epsilon\right) \\ &\leq \sum_{l=1}^{\infty} P\left(D_l - \mu_l \leq -\Delta h s_l - \Delta\epsilon\right), \end{aligned} \tag{24}$$

where (24) follows from the fact that $G_l + \delta_2 \overline{z}^*_{n_l}(\delta_2) \geq D_l$, w.p.1 for all l. The rest of the proof is same as in proof of Theorem 3 in (Bayraksan and Morton, 2008). We note that the required assumptions of the existence of moment generating functions, the central limit theorem for the difference random variables D_l and $\liminf_{n\to\infty} s_n(x) \geq \sigma^2(x)$, w.p.1 for all $x \in X$ are satisfied since ω has a bounded support, sampling is done in an i.i.d fashion and $|X|$ is finite. □

Part (ii) of Theorem 1 implies that the CI on the optimality gap of the candidate solution found by the procedure, given in (20), is a valid CI for values of h close to h'. We note that while this is an asymptotic result, CIs formed in much simpler settings such as for the mean of a random variable are often only asymptotically valid. Also note that the hypotheses of Theorem 1 are satisfied by SMRCSP, that is, $X \neq \emptyset$, $|X| = JT < \infty$ and SMRCSP has relatively complete recourse, and ω has bounded support, thus, $E \sup_{x \in X} |f(x, \omega)| < \infty$, provided the cost terms are finite. As a final remark, we note that when $\delta_2 = 0$, the above is same as the sequential procedure in (Bayraksan and Morton, 2008) with estimators SRP. However, for $\delta_2 > 0$, the estimators (18) and CI (20) are different. These changes, along with approximate solution of the sampling problems for generating the candidate solutions (i.e., $\delta_1^l > 0$) are essential to enable efficient solution of stochastic integer programs via a sampling method.

4.2 Computational Results

We first run the sequential sampling procedure on the eight test problems (all but 40-5-1) from Section 3.3 that were solved to within 1% optimality. We use these small problems to *verify* the sampling procedure by comparing against the solutions found by the exact procedure. We then test the performance of the sampling procedure on larger problems with up to 10^{24} scenarios. We set $h = 0.312$ and $h' = 0.025$ resulting in $n_1 = 50$ and $m_1 = 100$ and set $\delta_1^1 = \delta_2 = \delta$. Following Bayraksan and Morton (2008), the other parameters are set to $\alpha = 0.10$, $\epsilon' = 1 \times 10^{-7}$, $\epsilon = 2 \times 10^{-7}$, and $p = 1.91 \times 10^{-1}$.

Table 4 presents the results of the sampling procedure performed on the test problems from Section 3.3. Column "L" reports the number of iterations at termination, column "m_L" lists the sample size used to generate the solution found by the procedure ($m_L = 2n_L$), and column "CI%" reports the output confidence interval's width given in (20) divided by the exact solution objective from the solutions on the right-hand-side of Table 3. The "gap%" reports the percentage difference between the actual objective function values of the sampling solution and the exact solution. In some cases, the sampling procedure found a better solution than the exact solution as indicated by negative gap% values. All values are reported with a 90% CI around the means. While the output confidence interval on the gap is slightly inflated, the optimality gap compared to the exact objective remains small. The procedure is run with $1 - \alpha = 0.90$ (see part (ii) of Theorem 1), resulting in an empirical

$\delta = 2.5\%$	Average of 10 runs for each problem			
problem	L	m_L	CI%	gap%
20-5-1	2.0 ± 0.7	102.8 ± 1.9	3.78% ± 0.62%	0.37% ± 0.09%
20-5-2	1.6 ± 0.4	101.4 ± 1.0	3.27% ± 0.59%	0.19% ± 0.16%
20-5-3	1.5 ± 0.4	101.2 ± 1.0	5.32% ± 1.26%	1.99% ± 0.23%
40-5-2	2.2 ± 1.1	103.2 ± 3.2	3.55% ± 0.48%	0.24% ± 0.19%
40-5-3	1.6 ± 0.6	101.6 ± 1.5	4.25% ± 0.68%	1.41% ± 0.05%
80-5-1	2.3 ± 0.7	103.4 ± 1.9	3.04% ± 0.14%	0.28% ± 0.19%
80-5-2	1.4 ± 0.4	101.0 ± 1.0	2.86% ± 0.10%	-0.08% ± 0.13%
80-5-3	3.1 ± 1.1	106.0 ± 3.4	2.63% ± 0.51%	0.35% ± 0.27%

Table 4: Results from the sampling procedure when the sampling problems were solved to 2.5% optimality. 79/80 = 99% of runs resulted in actual optimality gaps within the gap estimate.

coverage probability of 0.99 ± 0.01 (79 out of 80 runs). Note that this is an empirical coverage probability ($\hat{p}$) along with a 90% CI half-width from 80 runs ($\pm 1.645\sqrt{\hat{p}(1-\hat{p})/80}$).

The main advantage of the sampling procedure is to allow (approximate) solution of problems of practical scale. Therefore, we now examine the algorithm's performance on problems with a large number of scenarios. We modified the test problems from Section 3.3 giving each job two processing time scenarios for a total of 2^J scenarios, for $J = 20, 40$, and 80. We label the modified test problems with a large number of scenarios by adding * to the instance number. For instance, problem 20-5-1 has $2^{10} = 1024$ scenarios and problem 20-5-1* has 2^{20} scenarios. Initially, all modified problems reached the 10 hour time limit on the sampling procedure. However, we found that this was due to $\bar{\rho}$ values close to 1. By extending the planning horizon from $T = 50$ to $T = 60$, we were able to lower the densities to around 0.8 - 0.9 (see Section 3.4). The slight change in density resulted in a dramatic difference in solution times that are reported in Tables 5-7. We report on selected problems for brevity. It is impossible to solve even the 20 job problem exactly since with 2^{20} scenarios the computer runs out of memory at the first iteration. Therefore, we cannot report the actual quality of the solutions but we can provide an upper bound on the quality by comparing the sampling solution CI to the solution of the expected value problem, which provides a valid lower bound on the stochastic problem. The expected value processing times were calculated as $\lfloor \bar{p}_j \rfloor$ for $j = 1, \ldots, J$. The "$\overline{\text{CI}}$%" column is CI divided by the objective from the expected value problem.

Tables 5-7 indicate that larger values of δ can considerably reduce solution times while still obtaining high-quality solutions. In particular $\delta = 2.5\%$ problems are solved very quickly. We note that $\overline{\text{CI}}$% provides an upper bound on CI% and in light of Table 4, we believe the actual solutions are much closer to optimality.

$\delta = 1\%$	Average of 10 runs for each problem					
problem	$\lvert\Omega\rvert$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20-5-1*	1.05×10^6	24,426 ± 5657	1.7 ± 0.3	101.4 ± 0.5	10.3 ± 2.6	2.20% ± 0.56%
40-5-2*	1.10×10^{12}	2556 ± 2908	1.4 ± 0.4	101.0 ± 1.0	23.2 ± 4.7	1.83% ± 0.37%
80-5-1*	1.21×10^{24}	25,964 ± 7757	2.2 ± 0.6	103.4 ± 1.9	44.5 ± 3.9	1.90% ± 0.17%

Table 5: Solving problems to 1% optimality results in solutions with optimality gaps of around 2%.

$\delta = 2\%$	Average of 10 runs for each problem					
problem	$\lvert\Omega\rvert$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20-5-1*	1.05×10^6	2692 ± 2411	3.6 ± 1.5	107.4 ± 4.3	12.8 ± 3.1	2.73% ± 0.66%
40-5-2*	1.10×10^{12}	437 ± 125	3.3 ± 1.5	106.4 ± 4.2	26.1 ± 4.2	2.05% ± 0.33%
80-5-1*	1.21×10^{24}	938 ± 223	3.8 ± 1.6	107.8 ± 4.5	46.8 ± 5.1	1.99% ± 0.22%

Table 6: Lowering δ to 2% reduces computation time by an order of magnitude and still provides quality solutions.

5 Conclusions

SMRCSP addresses the problem of determining an optimal schedule of multiple resource consuming jobs under uncertainty of processing times, resource consumptions, and resource capacities while allowing temporary resource capacity expansions for a cost. SMRCSP arises in operating room scheduling and many manufacturing and consulting applications. The single resource version arises in shipping berth allocation, semiconductor circuit design, and multiprocessor computer systems. The model presented creates schedules that minimize expected cost under uncertainty, and the formulation is generic allowing application of proposed methods to a variety of different objectives. The algorithm does not require input schedules to generate the proactive schedules. We solve the model with the L-shaped method, and we significantly improve the performance of the L-shaped method with GUB branching, trust regions, LP warm starting, and approximately solving the master problem. The combination of enhancements resulted in more than a 70% reduction in total solution time over 9 test problems. As our model handles uncertainties in processing times, resource consumption, and resource availabilities, problems of practical scale can quickly become intractable due to the prohibitively large number of scenarios. To alleviate this difficulty, we present and prove desired properties of a sequential sampling procedure that obtains high quality solutions without having to evaluate every scenario. Our computational results with up to 10^{24}

$\delta = 2.5\%$	Average of 10 runs for each problem					
problem	$\|\Omega\|$	time	L	m_L	CI	$\overline{\text{CI}}\%$
20-5-1*	1.05×10^6	668 ± 348	1.4 ± 0.4	101.0 ± 1.0	21.3 ± 1.9	$4.54\% \pm 0.40\%$
40-5-2*	1.10×10^{12}	343 ± 24	2.9 ± 0.9	105.2 ± 2.4	44.5 ± 1.8	$3.50\% \pm 0.14\%$
80-5-1*	1.21×10^{24}	694 ± 46	2.5 ± 0.8	104.4 ± 2.2	85.4 ± 2.8	$3.64\% \pm 0.12\%$

Table 7: Quality solutions are quickly found for problems with up to 10^{24} scenarios.

scenarios indicate that quality solutions can be found quickly ($\leq$ 1000 seconds) with this methodology. We conclude the paper by summarizing insights gained:

- Problems are difficult to solve when the resource density, a measure of the ratio of total resource consumption to availability, is close to 1. When the density is ≤ 0.8 or ≥ 1.3, problems are typically computationally easier to solve.
- A small increase in T can lower resource densities to a more desirable level. This also indicates that the jobs can be more efficiently scheduled within a longer time horizon.
- Alternatively, the number of jobs can be reduced by changing the problem to a job selection and scheduling problem. This can be achieved by changing "=" to "$\leq$" in constraints (2) and changing the objective to maximize some profit measure. This change does not affect the model structure, and the solution method developed in this paper remains valid.
- Schedules resulting in $w_{tk}^{\omega} > 0$ for a sufficiently large number of $\omega \in \Omega$ indicate resource bottlenecks in resource k in period t. Resources k with $z_{tk}^{\omega} > 0$ for a sufficiently large number of $\omega \in \Omega$ indicate resources that may benefit from permanent expansion.
- When the resources are tight ($\bar{\rho} \geq 1$), jobs are optimally started either as early as possible or as late as possible. As the resources become more available, jobs with medium processing times tend to start later while jobs with small and large processing times tend to start early.
- The sequential sampling procedure may be quite slow to converge when resource densities are close to 1. By a small increase in T, the procedure stops in a reasonable time. Preliminary computational results indicate 110 scenarios can be enough to find a quality approximate solution for this problem class.

Acknowledgments

We thank the referees for their insightful comments and Ken Baker for valuable input on an earlier version of this paper. This research is partially supported by the National Science Foundation under Grant EFRI-0835930.

References

Aytug, H., M. Lawley, K. McKay, S. Mohan, and R. Uzsoy (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research 161*, 86–110.

Baker, K. and D. Trietsch (2007). Safe scheduling. In *Tutorials in Operations Research*, pp. 79–101. INFORMS.

Bayraksan, G. and D. Morton (2006). Assessing solution quality in stochastic programs. *Mathematical Programming 108*, 495–514.

Bayraksan, G. and D. Morton (2008). A sequential sampling procedure for stochastic programming. Technical report, Department of Systems and Industrial Engineering, the University of Arizona. Available at: http://www.sie.arizona.edu/faculty/guzinb/publications.htm.

Benders, J. (1962). Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik 4*, 238–252.

Birge, J. and M. Dempster (1996). Stochastic programming approaches to stochastic scheduling. *Journal of Global Optimization 9*(3-4), 417–451.

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

Brucker, P. (2001). *Scheduling Algorithms* (3 ed.). Springer.

Cai, X., C. Lee, and C. Li (1998). Minimizing total completion time in two-processor task systems with prespecified processor allocations. *Naval Research Logistics 45*, 231–242.

Chan, L., A. Muriel, and D. Simchi-Levi (1998). Parallel machine scheduling, linear programming, and parameter list scheduling heuristics. *Operations Research 46*, 729–741.

Chen, J. and C. Lee (1999). General multiprocessor task scheduling. *Naval Research Logistics 46*, 57–74.

Cooper, D. (1976). Heuristics for scheduling resource constrained projects: An experimental comparison. *Management Science 22*, 1186–1194.

Deblaere, F., E. Demeulemeester, W. Herroelen, and S. V. de Vonder (2007). Robust resource allocation decisions in resource-constrained projects. *Decision Sciences 38*, 5–37.

Denton, B. and D. Gupta (2003). A sequential bounding approach for optimal appointment scheduling. *IIE Transactions 35*, 1003–1016.

Drozdowski, M. (1996). Scheduling multiprocessor tasks - An overview. *European Journal of Operational Research 94*, 215–230.

Dyer, M. and L. Wolsey (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics 26*, 255–270.

Herroelen, W. (2007). Generating robust project baseline schedules. In *Tutorials in Operations Research*, pp. 124–144. INFORMS.

Herroelen, W. and R. Leus (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research 165*, 289–306.

Herroelen, W. and B. D. Reyck (1999). Phase transitions in project scheduling. *The Journal of the Operational Research Society 50*, 148–156.

Herroelen, W., B. D. Reyck, and E. Demeulemeester (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers and Operations Research 25*(4), 279–302.

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

Hiriart-Urruty, J. and C. Lemaréchal (1993). *Convex Analysis and Minimization Algorithms II. Comprehensive Studies in Mathematics*. Springer-Verlag.

Hoogeveen, J., S. van de Velde, and B. Veltman (1994). Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics 55*, 259–272.

Kiwiel, K. (1990). Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming 46*, 105–122.

Kleywegt, A., A. Shapiro, and T. Homem-De-Mello (2001). The sample average approximation method for stochastic discrete optimization. *SIAM Journal of Optimization 12*, 479–502.

Kubale, M. (1987). The complexity of scheduling independent two-processor tasks on dedicated processors. *Informations Processing Letters 24*, 141–147.

Lee, C. and X. Cai (1999). Scheduling one and two-processor tasks on two parallel processors. *IIE Transactions 31*, 445–455.

Leus, R. and W. Herroelen (2004). Stability and resource allocation in project planning. *IIE Transactions 36*, 667–682.

Linderoth, J. and S. Wright (2003). Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications 24*, 207–250.

Marchand, H. and L. Wolsey (1999). The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming 85*, 15–33.

Miller, A., G. Nemhauser, and M. Savelsbergh (2000). On the capacitated lot-sizing and continuous 0-1 knapsack polyhedra. *European Journal of Operational Research 125*, 298–315.

Möhring, R., A. Schultz, F. Stork, and M. Uetz (2001). On project scheduling with irregular starting time costs. *Operations Research Letters 28*, 149–154.

Möhring, R., A. Schultz, F. Stork, and M. Uetz (2003). Solving project scheduling problems by minimum cut computations. *Management Science 49*, 330–350.

Morton, D. and E. Popova (2004). A bayesian stochastic programming approach to an employee scheduling problem. *IIE Transactions 36*, 155–167.

Pascoe, T. (1966). Allocation of resources - CPM. *Rev fr Rech opér 38*, 31–38.

Patterson, J. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics 23*, 95–123.

Queyranne, M. and A. Schulz (1994). Polyhedral approaches to machine scheduling. Preprint 408/1994, Dept. of Mathematics, Technical University of Berlin.

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

Santoso, T., S. Ahmed, M. Goetschalckx, and A. Shapiro (2005). A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research 167*, 96–115.

Shapiro, A. (2003). Monte Carlo sampling methods. In A. Ruszczyński and A. Shapiro (Eds.), *Handbooks in Operations Research and Management Science, Volume 10: Stochastic Programming*, pp. 353–425. Elsevier.

van de Vonder, S. (2006). *Proactive-reactive procedures for robust project scheduling*. Ph. D. thesis, Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Leuven, Belgium.

van de Vonder, S., E. Demeulemeester, and W. Herroelen (2007). A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling 10*, 195–207.

van Slyke, R. and R. Wets (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal of Applied Mathematics 17*, 638–663.

Wolsey, L. (1990). Valid inequalities for 0-1 knapsacks and MIPs with generalised upper bound constraints. *Discrete Applied Mathematics 29*, 251–261.

Wolsey, L. (1998). *Integer Programming*. John Wiley & Sons.

Zhu, G., J. Bard, and G. Yu (2007). A two-stage stochastic programming approach for project planning with uncertain activity durations. *Journal of Scheduling 10*, 167–180.