

# A Short Note on the Probabilistic Set Covering Problem

Anureet Saxena <sup>\*†‡</sup>

## Abstract

In this paper we address the following probabilistic version (PSC) of the set covering problem:  $\min\{cx \mid \mathbb{P}(Ax \geq \xi) \geq p, x_j \in \{0,1\} \ j \in N\}$  where  $A$  is a 0-1 matrix,  $\xi$  is a random 0-1 vector and  $p \in (0,1]$  is the threshold probability level. In a recent development Saxena, Goyal and Lejeune proposed a MIP reformulation of (PSC) and reported extensive computational results with small and medium sized (PSC) instances. Their reformulation, however, suffers from the *curse* of exponentiality – the number of constraints in their model can grow exponentially rendering the MIP reformulation intractable for all practical purposes. In this paper, we give a polynomial-time algorithm to separate the (possibly exponential sized) constraint set of their MIP reformulation. Our separation routine is independent of the specific nature (concave, convex, linear, non-linear etc) of the distribution function of  $\xi$ , and can be easily embedded within a branch-and-cut framework yielding a distribution-free algorithm to solve (PSC). The resulting algorithm can solve (PSC) instances of arbitrarily large block sizes by generating only a small subset of constraints in the MIP reformulation and verifying the remaining constraints implicitly. Furthermore, the constraints generated by the separation routine are independent of the coefficient matrix  $A$  and cost-vector  $c$  thereby facilitating their application in sensitivity analysis, re-optimization and warm-starting (PSC). We give preliminary computational results to illustrate our findings on a test-bed of 40 (PSC) instances created from the OR-Lib set-covering instance *scp41*.

*Keywords:* Probabilistic Programming, Set Covering, Mixed Integer Programming, Cutting Planes.

## 1 Introduction

In this paper we address the following probabilistic variant of the set-covering problem,

---

<sup>\*</sup>Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Email: anureets@andrew.cmu.edu, Phone No.: 1-412-268-2463.

<sup>†</sup>Research was supported by the National Science Foundation through grant #DMI-0352885 and by the Office of Naval Research through contract N00014-03-1-0133.

<sup>‡</sup>Corresponding Author

$$\begin{array}{ll}
\min & cx \\
\text{s.t} & \\
& \mathbb{P}(Ax \geq \xi) \geq p \\
& x_j \in \{0, 1\} \quad j \in N
\end{array} \tag{PSC}$$

where  $A$  is a 0-1 matrix defined on row-index set  $M$  and column-index set  $N$ ,  $\xi$  is a 0-1 random  $M$ -vector,  $p \in (0, 1]$  is the value of the *threshold probability* (also called the *reliability level*) and  $c \in \mathbb{R}^N$  is the cost vector. Indeed, if we replace the probabilistic constraint  $\mathbb{P}(Ax \geq \xi) \geq p$  in (PSC) by  $Ax \geq 1$  we recover the well-known set covering problem. (PSC) belongs to a class of optimization problems commonly referred to as probabilistic programs. We refer the reader to Prékopa [7] for a review of recent developments in this area. Applications of (PSC) can be found in Beraldi and Ruszczyński [2] while computational aspects of (PSC) are discussed in [2, 8].

Following Beraldi and Ruszczyński [2], we do not make any assumption on the probability distribution of  $\xi$ , except that  $\xi$  can be decomposed into  $L$  blocks say  $\{\xi^1, \dots, \xi^L\}$  such that  $\xi^t$  is a 0-1 random  $M_t$ -vector for  $t \in \{1, \dots, L\}$  (where  $M_1, \dots, M_L$  is a partition of  $M$ ), and  $\xi^i$  and  $\xi^j$  are independent random vectors for distinct  $i, j$ . Henceforth, for  $z \in \mathbb{R}^M$  we denote by  $z^t$  the sub-vector of  $z$  formed by components in  $M_t$  for  $t = 1 \dots L$ . Furthermore, let  $F : \{0, 1\}^M \rightarrow \mathbb{R}$  denote the cumulative distribution function of  $\xi$  and let  $F_t$  denote the restriction of  $F$  to  $M_t$  for  $t = 1 \dots L$ . In other words, for  $z \in \{0, 1\}^M$ ,  $F(z) = \mathbb{P}(\xi \leq z)$  and  $F_t(z^t) = \mathbb{P}(\xi^t \leq z^t)$ . Finally, we assume that  $F_t(v)$  can be computed in polynomial time for  $v \in \{0, 1\}^{M_t}$  and  $t = 1 \dots L$ .

In a recent development Saxena, Goyal and Lejeune [8] proposed a MIP reformulation of (PSC) which encodes the enumerative framework of Beraldi and Ruszczyński [2] as a mixed integer program. Based on their extensive computational experiments they concluded that their reformulation is several orders of magnitude faster than any of the previous approaches to solve (PSC). The MIP reformulation of Saxena et al. [8], however, suffers from the *curse* of exponentiality; in other words, the size of the MIP reformulation proposed in [8] can grow exponentially with the block-size  $m_t$ . Furthermore, the MIP reformulation requires a priori enumeration of the so-called  $p$ -inefficient frontier of the distribution functions  $F_t$  ( $t = 1 \dots L$ ) which raises serious concerns regarding the scalability of their model. Indeed, most of the computational experiments in [8] were restricted to (PSC) instances with block-size of 1, 5 or 10.

In this paper, we give a polynomial time algorithm to separate the (possibly exponential sized) constraint set of the MIP reformulation of (PSC) introduced in [8]. Our separation routine is independent of the specific nature (concave, convex, linear, non-linear etc) of the distribution functions  $F_t$ , and can be easily embedded within a branch-and-cut framework yielding a distribution-free algorithm to solve (PSC). The resulting algorithm can solve (PSC) instances of arbitrarily large block sizes by generating only a small subset of constraints in the MIP reformulation and verifying the remaining constraints implicitly. Furthermore, the constraints generated by the separation routine are independent of the coefficient matrix  $A$

and cost-vector  $c$  thereby facilitating their application in sensitivity analysis, re-optimization and warm-starting (PSC).

The rest of the paper is organized as follows. Section 2 revisits the MIP reformulation of (PSC) proposed by Saxena et al. [8]. In section 3 we present our polynomial time algorithm to separate the constraint set of the MIP reformulation. We conclude this paper with some illustratory computational results presented in section 4.

## 2 MIP Reformulation Revisited

In this section we revisit the MIP reformulation of (PSC) introduced by Saxena et al. [8] and discuss some of its shortcomings.

Recall, a point  $v \in \{0, 1\}^m$  is called a *p-inefficient* point of the probability distribution function  $F$  if  $F(v) < p$  and there is no binary point  $w \geq v$ ,  $w \neq v$  such that  $F(w) < p$  [8]. Similarly, a point  $v \in \{0, 1\}^m$  is called a *p-efficient* point of the probability distribution function  $F$  if  $F(v) \geq p$  and there is no binary point  $w \leq v$ ,  $w \neq v$  such that  $F(w) \geq p$  [6]. For  $t \in \{1, \dots, L\}$ , let  $S_t$  denote the set of binary vectors which are either *p-efficient* or dominate a *p-efficient* point of  $F_t$  and let  $I_t$  denotes the set of *p-inefficient* points of  $F_t$ . The theorem that follows gives a MIP reformulation of (PSC).

**Theorem 2.1** (Theorem 2.4 of [8]) (PSC) can be reformulated as the following mixed integer program.

$$\begin{aligned}
 & \min_{(x,z,\eta)} && cx \\
 & \text{s.t} && \\
 & Ax &\geq z \\
 & \sum_{t=1}^L \eta_t &\geq \ln p \\
 & \eta_t &\leq (\ln F_t(v))(1 - \sum_{i \in M_t, v_i=0} z_i) \quad \forall v \in S_t \quad \forall t \in \{1, \dots, L\} \\
 & 1 &\leq \sum_{i \in M_t, v_i=0} z_i \quad \forall v \in I_t \quad \forall t \in \{1, \dots, L\} \\
 & x_j &\in \{0, 1\} \quad \forall j \in N \\
 & z_i &\in \{0, 1\} \quad \forall i \in M
 \end{aligned} \tag{SGL}$$

For the sake of brevity, we refer to the constraints in (SGL) arising from  $v \in I_t$  ( $t = 1 \dots L$ ) and  $v \in S_t$  ( $t = 1 \dots L$ ) as *I-constraints* and *S-constraints*, respectively; both of these constraints are collectively referred to as *IS-constraints*.

Some comments are in order. For a given distribution function  $F_t$  the number of lattice points  $v \in \{0, 1\}^{M_t}$  which are *p-inefficient*, *p-efficient* or dominate a *p-efficient* can be exponential in  $m_t$  ( $t = 1 \dots L$ ). Furthermore, even if the number of such points is polynomial it may be too large rendering the MIP model intractable for all practical purposes. Besides, for large values of block size  $m_t$  it is possible that only a small subset of the *IS-constraints* are required to solve (PSC) while the remaining constraints can be verified implicitly without

ever generating them in an explicit form. This, for instance, is the case with the famous Edmond's algorithm for the matching problem [5]. Even though there are exponentially many odd set inequalities which define facets of the matching polyhedra, only a polynomial sized subset of these inequalities are required to solve the matching problem.

To summarize, the scalability of the (SGL) model i.e its ability to address problems with large block sizes  $m_t$  depends crucially on the existence of a polynomial time separation algorithm for its constraints. We give such a separation algorithm in the following section.

### 3 Poly-Time Separation Algorithm

In this section we present a polynomial time algorithm to separate the constraints of (SGL). Given a point  $(x, z, \eta) \in [0, 1]^N \times \{0, 1\}^M \times \mathbb{R}^L$  which satisfies  $Ax \geq z$  and  $\sum_{t=1}^L \eta_t \geq \ln p$ , Algorithm 3.1 either shows that  $(x, z, \eta)$  is a feasible solution to the LP relaxation of (SGL) or finds an *IS*-constraint which is violated by  $(x, z, \eta)$ .

**Algorithm 3.1** *Inputs to the algorithm are the polynomial time oracles to compute the cumulative distribution functions  $F_t$  ( $t = 1 \dots L$ ) and the point  $(\bar{x}, \bar{z}, \bar{\eta}) \in [0, 1]^N \times \{0, 1\}^M \times \mathbb{R}^L$  to be separated which satisfies  $A\bar{x} \geq \bar{z}$  and  $\sum_{t=1}^L \bar{\eta}_t \geq \ln p$ .*

1. Compute  $F_t(\bar{z}^t)$  for  $t = 1 \dots L$ .
2. Let  $t := 1$ .
3. If  $F_t(\bar{z}^t) \geq p$  then GOTO step (8).
4. Let  $(i_1, \dots, i_q)$  be an arbitrary ordering of elements in the set  $\{i \in M_t \mid \bar{z}_i = 0\}$ . Let  $v := \bar{z}^t$  and  $k := 1$ .
5. If  $F_t(v + e^k) < p$  then set  $v := v + e^k$  ( $e^k$  denotes the unit vector in  $\{0, 1\}^{M_t}$  with 1 in the  $i_k^{\text{th}}$  position).
6. If  $k < q$  then set  $k := k + 1$  and GOTO step (5).
7. **Assert:**  $v \in I_t$  and the inequality  $\sum_{i \in M_t, v_i=0} z_i \geq 1$  is violated by  $(\bar{x}, \bar{z}, \bar{\eta})$ . STOP
8. Set  $t := t + 1$ . If  $t < L$  then GOTO step (3).
9. Let  $t := 1$ .
10. If  $\bar{\eta}_t \leq \ln F_t(\bar{z}^t)$  then GOTO step (13).
11. Let  $v := \bar{z}^t$
12. **Assert:**  $v \in S_t$  and the inequality  $\eta_t \leq (\ln F_t(v))(1 - \sum_{i \in M_t, v_i=0} z_i)$  is violated by  $(\bar{x}, \bar{z}, \bar{\eta})$ . STOP

13. Set  $t := t + 1$ . If  $t < L$  then GOTO (10).

14. **Assert:**  $(\bar{x}, \bar{z}, \bar{\eta})$  is a feasible solution to the LP relaxation of (SGL).

Before proceeding to the proof of correctness and polynomiality of Algorithm 3.1 we make a technical remark. Note that Algorithm 3.1 requires that the point  $(x, z, \eta)$  to be separated satisfy  $z_i \in \{0, 1\} \forall i \in M$ . This implies that it can only be invoked at those nodes of the branch-and-bound tree whose LP relaxation solution satisfies  $z_i \in \{0, 1\} \forall i \in M$ . The reader should verify that despite this integrality requirement, the separation algorithm (Algorithm 3.1) presented here can still be embedded within a branch-and-cut framework to solve (PSC).

The theorem that follows proves the aforementioned characteristics of Algorithm 3.1.

**Theorem 3.2** For a given  $(\bar{x}, \bar{z}, \bar{\eta}) \in [0, 1]^N \times \{0, 1\}^M \times \mathbb{R}^L$  which satisfies  $A\bar{x} \geq \bar{z}$  and  $\sum_{t=1}^L \bar{\eta}_t \geq \ln p$ , Algorithm 3.1 terminates in polynomial time and either shows that  $(\bar{x}, \bar{z}, \bar{\eta})$  is a feasible solution to the LP relaxation of (SGL) or finds an IS-constraint which is violated by  $(\bar{x}, \bar{z}, \bar{\eta})$ .

**Proof:**

The polynomiality of Algorithm 3.1 follows trivially from the observation that it computes the cumulative distribution functions  $F_t$  ( $t = 1 \dots L$ ) at most polynomial ( $O(m)$  to be precise) number of times. We next prove its correctness.

First, consider the case when  $\exists t \in \{1, \dots, L\}$  such that  $F_t(\bar{z}^t) < p$ ; in other words  $\bar{z}^t$  is either  $p$ -inefficient or is dominated by a  $p$ -inefficient point of  $F_t$ . Steps (4)-(6) of the algorithm give a greedy algorithm to find a  $p$ -inefficient point  $v \in I_t$  which satisfies  $\bar{z}^t \leq v$ ; clearly  $\sum_{i \in M_t, v_i=0} \bar{z}_i = 0$  and the inequality  $\sum_{i \in M_t, v_i=0} z_i \geq 1$  reported by the algorithm in step (7) is violated by  $(\bar{x}, \bar{z}, \bar{\eta})$ . Next consider the case when  $F_t(\bar{z}^t) \geq p \forall t \in \{1, \dots, L\}$  and  $\exists t \in \{1, \dots, L\}$  such that  $\bar{\eta}_t > \ln F_t(\bar{z}^t)$ ; in other words  $\bar{z}^t \in S_t$  and the inequality  $\eta_t \leq (\ln F_t(\bar{z}^t))(1 - \sum_{i \in M_t, \bar{z}_i=0} z_i)$  reported by the algorithm in step (12) is violated by  $(\bar{x}, \bar{z}, \bar{\eta})$ .

Finally, consider the case when  $F_t(\bar{z}^t) \geq p$  and  $\eta_t \leq \ln F_t(\bar{z}^t) \forall t \in \{1, \dots, L\}$ . Since  $\bar{z}^t \in S_t \forall t \in \{1, \dots, L\}$ ,  $(\bar{x}, \bar{z}, \bar{\eta})$  satisfies all the  $I$ -constraints in (SGL). If  $\exists t \in \{1, \dots, L\}$  and  $\exists v \in S_t$  such that  $\bar{\eta}_t > (\ln F_t(v))(1 - \sum_{i \in M_t, v_i=0} \bar{z}_i)$  then  $\sum_{i \in M_t, v_i=0} \bar{z}_i = 0$  (since  $\bar{\eta}_t \leq \ln F_t(\bar{z}^t) \leq 0$ ),  $\bar{z}^t \leq v$ ,  $\ln F_t(\bar{z}^t) \leq \ln F_t(v)$  which implies that  $\bar{\eta}_t \leq \ln F_t(\bar{z}^t) \leq \ln F_t(v) < \bar{\eta}_t$ , a contradiction. Hence, if Algorithm 3.1 reaches step (14) then  $(\bar{x}, \bar{z}, \bar{\eta})$  satisfies all the constraints of the LP relaxation of (SGL). ■

Several comments are in order. First, Algorithm 3.1 shields the inherent complexity of the distribution functions  $F_t$  ( $t = 1 \dots L$ ) thereby allowing us to develop a distribution-free branch-and-cut procedure to solve (PSC); the resulting procedure is independent of the specific nature (concave, convex, linear, non-linear etc) of the distribution functions  $F_t$  as long as there exists a polynomial time oracle to compute  $F_t(v)$  for  $v \in \{0, 1\}^{M_t}$  ( $t = 1 \dots L$ ).

Second, Algorithm 3.1 implicitly prioritizes the  $IS$ -constraints of the (SGL) model; in other words constraints which cut off the incumbent LP relaxation solution are given higher priority and are generated immediately, whereas the generation of remaining constraints is either

postponed or never carried out. This opens up the possibility of solving (PSC) via (SGL) by generating only a subset of the constraint set of (SGL). As our computational results show (section 4), our code was able to solve some of the (PSC) instances to optimality by generating only 25% of the total number of constraints in (SGL).

Third, Algorithm 3.1 can be easily modified to generate more than one violated inequality for the case when  $(\bar{x}, \bar{z}, \bar{\eta})$  is not a feasible solution to the LP relaxation of (SGL). Such a modification, for instance, may involve repeating steps (4)-(7) of the algorithm and using a randomly generated ordering of elements in each iteration. The modified separation procedure will generate a collection of cuts instead of a single violated inequality. It is well-known (and confirmed by our experience) that the time spent in generating these additional cuts is more than compensated by the strengthening of the relaxation which results due to the collaborative action of the cuts (also see Fischetti and Lodi [4]).

Fourth, consider the branch-and-cut procedure to solve (PSC) which uses Algorithm 3.1 to separate the constraint set of (SGL). Let (SGL') denote the relaxation of (SGL) obtained by retaining only those *IS*-constraints which were generated by Algorithm 3.1 during the branch-and-cut procedure. Note that the optimal values of (SGL') and (SGL) are identical, and there exists an optimal solution to (SGL') which is also a feasible solution to (SGL). As our computational results show (section 4), (SGL') provides a very good approximation of (SGL), and it can be used for sensitivity analysis, re-optimization and warm-starting (PSC). Furthermore, the constraints generated by Algorithm 3.1 depend only on the distribution of the random variable  $\xi$  (see PSC) and are independent of the coefficient matrix  $A$  and cost-vector  $c$ . Consequently, these cuts can be used to warm-start (PSC) even after one or more entries of the coefficient matrix  $A$  or cost-vector  $c$  have been altered.

Fifth, the branch-and-cut procedure described above and the cut-and-branch algorithm proposed in [8] represent two extreme view-points to solve (PSC). To see this, let us compare these two approaches on the quantum of probabilistic information computed by each one of them. While our approach is based on delayed constraint generation which postpones the generation of *IS*-constraints unless absolutely necessary, the cut-and-branch approach of [8] a priori computes all the *IS*-constraints and uses them to generate strong cutting planes (called polarity cuts [8]) for (PSC). Consequently, our algorithm can handle arbitrarily large block sizes at the price of working with a weaker relaxation of (PSC) whereas the algorithm presented in [8] works with a much stronger relaxation of (PSC) but lacks the scalability property.

## 4 Computational Results

In this section we discuss preliminary computational results obtained by embedding the separation Algorithm 3.1 within a branch-and-cut framework. We will like to stress that the computational results presented here are for the sake of illustration only, and are not meant to evaluate the practical usefulness of our algorithm.

We implemented our algorithm using COIN-OR [3] and CPLEX (version 10.1). The separation algorithm was implemented using COIN-OR modules, and was subsequently embedded within the branch-and-cut framework of CPLEX using callbacks<sup>1</sup>. We applied the following two modifications to Algorithm 3.1 for the sake of efficiency.

1. Algorithm 3.1 terminates after it has generated a single violated inequality arising from  $v \in S_t \cup I_t$  for some  $t \in \{1, \dots, L\}$ . In our experiments, we found it useful to execute steps (3)-(7) and (11)-(12) of the algorithm for all values of  $t$  in the set  $\{1, \dots, L\}$  and generate violated inequalities from as many of them as possible.
2. For  $t \in \{1, \dots, L\}$  such that  $F_t(\bar{z}^t) < p$ , we repeated steps (4)-(7) of the algorithm  $\min\{100, 2^{n_t}\}$  number of times where  $n_t = |\{i \in M_t \mid \bar{z}_i = 0\}|$  denotes the number of zero components of  $\bar{z}^t$ , and used a random ordering of elements in the set  $\{i \in M_t \mid \bar{z}_i = 0\}$  in each iteration.

Next we describe the test-bed of probabilistic instances we used in our experiment. For each block  $t \in \{1, \dots, L\}$  we generated the distribution function  $F_t$  in the following manner. We chose a number  $l_t \geq 1$  and generated a set  $\Sigma_t = \{b^{jt} \in \{0, 1\}^{M_t} \mid j = 1 \dots l_t\}$  of  $l_t$  (not necessarily distinct) random 0-1 vectors to serve as the *support* of the distribution of  $\xi^t$ ; in other words for  $v \in \{0, 1\}^{M_t}$ ,  $\mathbb{P}(\xi^t = v) \neq 0$  if and only if  $v \in \Sigma_t$ . With each element  $b^{jt}$  of  $\Sigma_t$  we associated the probability value  $\mathbb{P}(\xi^t = b^{jt}) = \frac{1}{l_t}$  to completely define the distribution function  $F_t$ . Thus for  $v \in \{0, 1\}^{M_t}$ ,  $F_t(v) = \sum_{j=1 \dots l_t, b^{jt} \leq v} \mathbb{P}(\xi^t = b^{jt})$ . We chose  $l_t = 10m_t$  in our experiments. Note that  $F_t(v)$  can be computed in polynomial time for  $v \in \{0, 1\}^{M_t}$  and  $t \in \{1, \dots, L\}$ .

We used the OR-Lib [1] set-covering instance scp41 (1000 columns and 200 rows) to create a test-bed of 40 probabilistic instances in the following manner. We considered ten different block sizes, namely 5,10,15,...,50. For each one of the ten block sizes we generated four probabilistic instances differing only in the values of the threshold probability  $p$  which were chosen from  $\{0.80, 0.85, 0.90, 0.95\}$ . For each instance, we ran our code with a time-limit of 1hr. All experiments were carried out on a linux workstation with a 2.6GHz AMD Opteron(tm) 852 Processor and 8GB RAM.

Tables 1-3 summarize our key findings. Table 1 gives statistics on the performance of our algorithm while tables 2 and 3 give detailed information about  $I$ -constraints and  $S$ -constraints which were generated by our code, respectively. The results are categorized by block size and threshold probability  $p$  which are given in the first and second columns of the tables, respectively.

The third and fourth columns of table 1 give the total solution time and the number of branch-and-bound nodes enumerated by CPLEX, respectively. Of the 40 probabilistic instances on which we had run our code, we were able to solve 37 instances to optimality within 1hr.

---

<sup>1</sup>Callbacks are software provisions which allow the user to intermittently take control of the branch-and-cut algorithm and guide the behaviour of the MIP solver.

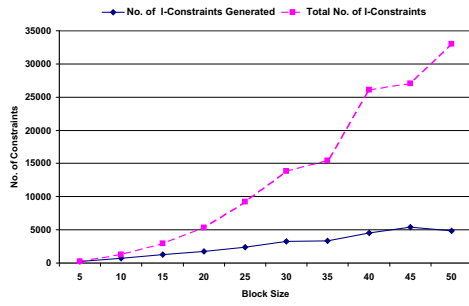
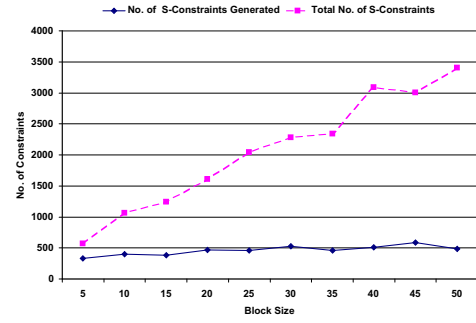
(a) *I*-Constraints(b) *S*-Constraints

Figure 1: Number of Constraints Generated

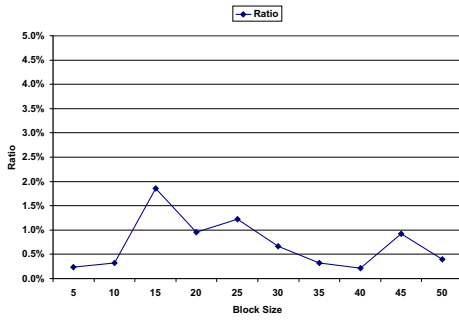
In order to assess the performance of our algorithm over the unsolved instances we give the percentage relative gap<sup>2</sup> which remained at the end of 1hr in the fifth column of the table.

Table 2 gives detailed information about the *I*-constraints which were generated by our code. The third column of the table reports the number of *I*-constraints which were generated whereas the fourth column reports the total number of *I*-constraints in the (SGL) model. Figure 1(a) represents the same information graphically for instances in our test-bed with threshold probability  $p = 0.8$ . Note that, unlike the total number of *I*-constraints in the (SGL) model, the number of *I*-constraints generated by our code is a slowly growing function of the block size.

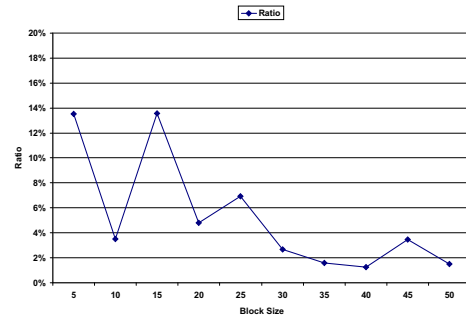
The fifth and sixth columns of the table give statistical information about nodes of the branch-and-bound tree where the *I*-constraints were generated. Suppose the code generated  $q$  *I*-constraints and the  $i^{\text{th}}$  *I*-constraint was generated at a node located at depth  $d_i$  of the branch-and-bound tree, and after CPLEX had enumerated  $n_i$  branch-and-bound nodes ( $i = 1, \dots, q$ ). The fifth column of the table reports the average depth  $\frac{\sum_{i=1}^q d_i}{q}$  of the branch-and-bound tree at which the constraints were generated, whereas the sixth column reports the average amount of enumeration  $\frac{\sum_{i=1}^q n_i}{q}$  which was required to generate them. For the sake of comparison, we also give the total number of branch-and-bound nodes enumerated by CPLEX in the last column of the table. Note that most of the constraints were generated at nodes close to the root node. Furthermore, the numbers in the sixth column are substantially smaller than the ones in the seventh column. This suggests that even though CPLEX enumerated thousands of nodes to solve some of the instances to optimality, most of the constraints were generated in the early stages ( $< 5\%$  of the total number of nodes) of the enumeration process. As we will see later, this phenomenon can be exploited to solve instances which are otherwise very difficult to solve. Figure 2(a) represents this phenomenon graphically for instances with threshold probability  $p = 0.8$ ; the vertical axis plots the ratio of columns six and seven of

---

<sup>2</sup> $RG = 100 \times \frac{ip - bb}{bb}$  where RG is the percentage relative gap,  $ip$  is the value of the best solution and  $bb$  is the value of the best bound available at the end of 1hr.



(a)  $I$ -Constraints (Table 2)



(b)  $S$ -Constraints (Table 3)

Figure 2: Ratio of columns six and seven of Tables 2 and 3

Table 2 whereas the horizontal axis gives the block size of the instances.

Table 3 and figures 1(b) and 2(b) give the same information for  $S$ -constraints as provided by Table 2 and figures 1(a) and 2(a) for  $I$ -constraints.

Next we describe two experiments to highlight the warm-starting capabilities of the constraints generated by Algorithm 3.1. Both of these experiments are based on the probabilistic instance, referred to as the *prototype instance* in the sequel, derived from scp41 using block size of 30 and threshold probability  $p = 0.8$ ; this instance can be solved to optimality by our algorithm in 948 sec by generating 3253  $I$ -constraints and 531  $S$ -constraints (see Tables 1, 2 and 3).

In the first experiment we generated ten probabilistic instances by perturbing the cost coefficients  $c_j$  of the prototype instance. If  $c$  denotes the cost vector of the prototype instance, the cost vector  $\bar{c}$  of the  $i^{th}$  instance ( $i = 1 \dots 10$ ) was defined as  $\bar{c}_j := c_j(1 + \mu_{ij}) \forall j \in N$ , where  $\mu_{ij}$  was chosen uniformly randomly from the interval  $[-\frac{i}{100}, \frac{i}{100}]$ . Since the constraints generated by Algorithm 3.1 are independent of the cost-vector, they can be used to provide warm-starts while solving the perturbed instances. We ran our code on these ten perturbed instances in two setups. In the first setup we strengthened the initial formulation by adding the constraints generated for the prototype instance. The second setup was identical to the first one except that the constraints generated for the prototype instance were not provided.

Table 4 summarizes our key findings. The first column of the table gives the extent of perturbation. The next four columns describe the solution characteristics of the first setup while the last four columns give the same for the second setup. Figure 3 plots the ratio of the time-taken and number of nodes enumerated by CPLEX in the second and first setups, respectively. As is evident from the figure, warm-starting (PSC) using constraints generated for the prototype instance reduced the solution characteristics (time and number of nodes) by at least two orders of magnitude.

In the second experiment we generated nine (PSC) instances whose only common characteristic with the prototype instance was the distribution of the random variable  $\xi$ ; in other words

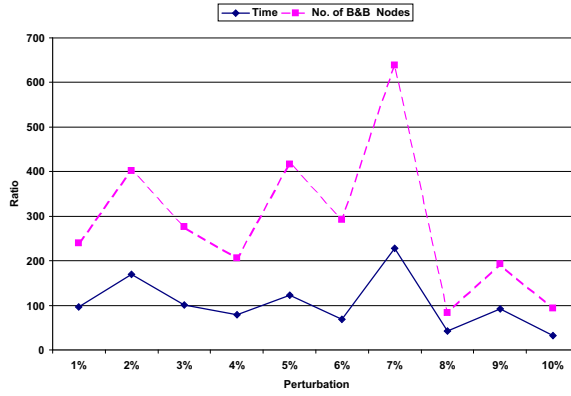


Figure 3: Ratio of Solution Characteristics: First Warm-Start Experiment

each one of these nine instances had 200 rows (similar to scp41), block size of 30 and  $p = 0.8$  (similar to the prototype instance) and the distribution function  $F$  of the random variable  $\xi$  was identical to the one used to define the prototype instance; besides these similarities each one of them had a distinct coefficient matrix  $A$  and cost vector  $c$ . We used the OR-Lib [1] set-covering instances scp42-scp49 and scp410 to construct these nine instances. For the sake of comparison, we also included the prototype instance (derived from scp41) in our test-bed yielding ten instances in all.

As in the case of the first experiment, we ran our code on these ten instances in two setups wherein only the first setup was warm-started. Table 5 describes our key findings. The first column reports the source of the probabilistic instance. The remaining columns have the same interpretation as the columns of table 4. Figure 4 plots the ratio of the time taken and the number of nodes enumerated by CPLEX in the second and first setup, respectively. While the first setup, aided by constraints generated for the prototype instance, solved each one of the ten instances in less than six minutes, the second setup was unable to solve eight of the ten instances within the prescribed time-limit of 1hr.

We conclude this section by briefly discussing a 2-phase algorithm to solve (PSC) and demonstrating its application in solving the three previously unsolved problems in our test-bed (see table 1). For the sake of motivation, consider the prototype instance described earlier; our code solved this instance to optimality by enumerating  $q = 180878$  branch-and-bound nodes. For  $i = 0, \dots, q$ , let  $(SGL_i)$  denote the relaxation of (SGL) obtained by retaining only those  $IS$ -constraints which were generated by Algorithm 3.1 until CPLEX enumerated  $i$  branch-and-bound nodes; let  $n_i$  denote the number of  $IS$ -constraints in  $(SGL_i)$ . Figure 5 plots the fraction  $\frac{n_i}{n_q}$  of the total number of constraints generated by Algorithm 3.1 and the duality gap<sup>3</sup> closed by the LP relaxation of  $(SGL_i)$ , with respect to the number of branch-and-bound

<sup>3</sup> $DG = 100 \times \frac{LP(SGL_i) - LP(SGL_0)}{ip - LP(SGL_0)}$  where DG denotes the duality gap closed,  $ip$  denotes the optimal value of the prototype instance and  $LP(SGL_i)$  ( $LP(SGL_0)$ ) denotes the optimal value of the LP relaxation of

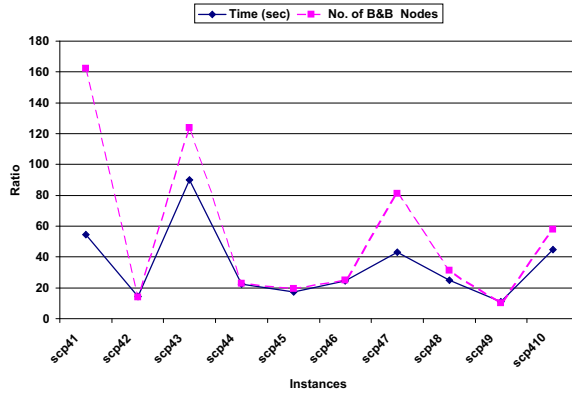


Figure 4: Ratio of Solution Characteristics: Second Warm-Start Experiment

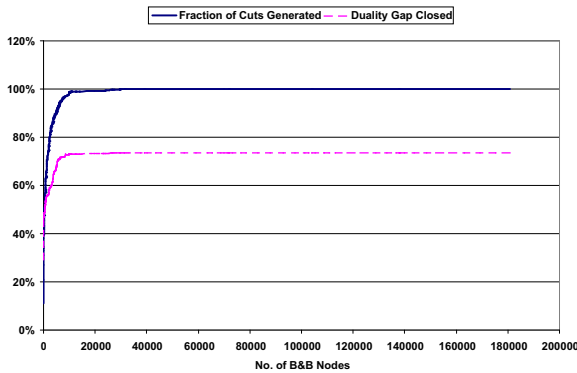


Figure 5: Detailed Results: Prototype Instance

nodes enumerated by CPLEX.

Two observations are in order. First, figure 5 shows that 95% of the constraints were generated within the first 5% of the total number of nodes enumerated by CPLEX. Second, the last 5% of the constraints do not improve the strength of the LP relaxation of  $(SGL_i)$  significantly. In other words, most of the useful constraints were generated in the early stages of the enumeration process. These observations suggest the following 2-phase algorithm to solve hard (PSC) instances. The first phase involves a branch-and-cut algorithm to solve (PSC) which uses Algorithm 3.1 to separate the constraints of (SGL) and terminates after a pre-determined time-limit of  $T$  seconds. After the first phase has ended, the second phase restarts the branch-and-cut algorithm working with a relaxation of (SGL) strengthened by  $SGL_i (SGL_0)$ .

constraints generated in the first phase. Alternatively, the first phase serves as an exploratory phase aimed at generating a good subset of  $IS$ -constraints, while the second phase consolidates the outcome of the first phase by starting with a strengthened relaxation.

Table 6 summarize the results of applying this 2-phase algorithm to the three unsolved instances in our test-bed of 40 instances; we chose a time-limit of  $T = 120$  sec for the first phase of this experiment. As is evident from the table, the 2-phase algorithm solved each one of the three instances in less than five minutes.

## References

- [1] Beasley, J.E.: OR-Library, [people.brunel.ac.uk/mastjjb/jeb/info.html](http://people.brunel.ac.uk/mastjjb/jeb/info.html)
- [2] Beraldi, P., Ruszczyński, A.: The probabilistic set covering problem. *Operations Research* 50, 956-967 (2002)
- [3] COIN-OR: Computational infrastructure for operations research, <http://www.coin-or.org>
- [4] Fischetti, M., Lodi, A.: Optimizing over the first Chvtal closure. *Mathematical Programming*. To appear
- [5] Nemhauser, G.L. and Wolsey, L.A., *Integer and Combinatorial Optimization*, J. Wiley Publication, 1988.
- [6] Prékopa, A.: Dual method for a one-stage stochastic programming with random rhs obeying a discrete probability distribution. *Zeitschrift of Operations Research* 34, 441-461 (1990)
- [7] Prékopa, A.: Probabilistic programming models. Chapter 5 in: *Stochastic Programming: Handbook in Operations Research and Management Science* 10. Edited by Ruszczyński, A., Shapiro, A. Elsevier Science Ltd, 267-351 (2003)
- [8] Saxena, A., Goyal, V. and Lejeune, M., MIP Reformulations of the Probabilistic Set Covering Problem, *Optimization Online* (e-print). [http://www.optimization-online.org/DB\\_HTML/2007/02/1579.html](http://www.optimization-online.org/DB_HTML/2007/02/1579.html)

# Appendix

Block Size	p	Time (sec)	Number of B&B Nodes	% RG
5	0.80	7.43	2731	-
5	0.85	3.28	1575	-
5	0.90	0.26	12	-
5	0.95	0.14	9	-
10	0.80	50.31	30724	-
10	0.85	11.13	5313	-
10	0.90	2.42	1235	-
10	0.95	0.08	0	-
15	0.80	36.69	8850	-
15	0.85	12.61	5441	-
15	0.90	1.73	730	-
15	0.95	0.09	0	-
20	0.80	155.15	40929	-
20	0.85	32.69	12192	-
20	0.90	1.20	399	-
20	0.95	0.15	0	-
25	0.80	193.20	38859	-
25	0.85	40.36	19851	-
25	0.90	2.39	1100	-
25	0.95	0.23	0	-
30	0.80	948.33	180878	-
30	0.85	47.99	17171	-
30	0.90	2.30	1065	-
30	0.95	0.27	0	-
35	0.80	>3600	509100	2.57
35	0.85	17.52	4980	-
35	0.90	1.95	688	-
35	0.95	0.35	0	-
40	0.80	>3600	492073	2.34
40	0.85	88.23	27683	-
40	0.90	1.59	364	-
40	0.95	0.55	0	-
45	0.80	2862.29	263754	-
45	0.85	1177.10	464795	-
45	0.90	2.01	355	-
45	0.95	0.67	0	-
50	0.80	>3600	440701	6.11
50	0.85	321.62	98981	-
50	0.90	2.13	537	-
50	0.95	0.85	0	-

Table 1: Summary Results: scp41

Block Size	p	No. of <i>I</i> -Constraints Generated	Total No. of <i>I</i> -Constraints	Avg Depth of B&B Nodes	Avg No. of B&B Nodes	Total No. of B&B Nodes
5	0.80	208	240	2.11	6.59	2731
5	0.85	224	253	2.22	8.44	1575
5	0.90	188	198	0.06	0.06	12
5	0.95	176	176	0.00	0.00	9
10	0.80	695	1251	15.67	98.77	30724
10	0.85	556	729	8.82	46.30	5313
10	0.90	333	420	3.51	10.77	1235
10	0.95	188	188	0.00	0.00	0
15	0.80	1238	2918	19.58	164.56	8850
15	0.85	836	1306	16.01	80.19	5441
15	0.90	337	509	1.93	7.32	730
15	0.95	198	198	0.00	0.00	0
20	0.80	1770	5300	28.53	389.03	40929
20	0.85	1077	1831	24.28	246.28	12192
20	0.90	414	683	1.85	7.72	399
20	0.95	197	197	0.00	0.00	0
25	0.80	2403	9213	27.42	474.49	38859
25	0.85	1203	2361	18.05	169.68	19851
25	0.90	478	838	4.50	18.52	1100
25	0.95	199	199	0.00	0.00	0
30	0.80	3253	13829	37.39	1205.14	180878
30	0.85	1334	2641	24.84	324.41	17171
30	0.90	564	987	4.48	26.78	1065
30	0.95	199	199	0.00	0.00	0
35	0.80	3313	15439	38.84	1655.80	509100
35	0.85	1439	3048	21.28	256.28	4980
35	0.90	470	847	3.17	18.80	688
35	0.95	199	199	0.00	0.00	0
40	0.80	4505	26127	38.30	1047.18	492073
40	0.85	1740	3867	26.93	292.39	27683
40	0.90	552	1292	2.86	12.77	364
40	0.95	200	200	0.00	0.00	0
45	0.80	5439	27053	37.62	2439.47	263754
45	0.85	1876	3999	27.49	745.87	464795
45	0.90	593	1127	4.58	24.03	355
45	0.95	198	198	0.00	0.00	0
50	0.80	4891	33039	38.36	1747.99	440701
50	0.85	1980	4889	30.22	620.79	98981
50	0.90	497	951	3.89	18.81	537
50	0.95	200	200	0.00	0.00	0

Table 2: *I*-Constraints

Block Size	p	No. of <i>S</i> -Constraints Generated	Total No. of <i>S</i> -Constraints	Avg Depth of B&B Nodes	Avg No. of B&B Nodes	Total No. of B&B Nodes
5	0.80	332	566	63.56	369.18	2731
5	0.85	212	327	45.43	220.12	1575
5	0.90	113	191	1.87	2.00	12
5	0.95	24	66	0.29	0.29	9
10	0.80	399	1060	71.51	1077.36	30724
10	0.85	219	418	55.38	649.86	5313
10	0.90	106	156	29.38	208.77	1235
10	0.95	15	35	0.00	0.00	0
15	0.80	381	1245	65.70	1202.40	8850
15	0.85	174	325	66.05	651.98	5441
15	0.90	63	128	19.60	131.08	730
15	0.95	2	16	0.00	0.00	0
20	0.80	469	1611	71.20	1972.56	40929
20	0.85	198	374	75.04	1635.93	12192
20	0.90	60	124	21.23	112.05	399
20	0.95	3	13	0.00	0.00	0
25	0.80	463	2045	66.23	2701.20	38859
25	0.85	169	342	60.68	1215.98	19851
25	0.90	69	120	28.30	257.16	1100
25	0.95	1	9	0.00	0.00	0
30	0.80	531	2280	69.35	4843.91	180878
30	0.85	161	319	68.89	2333.81	17171
30	0.90	79	127	19.67	200.48	1065
30	0.95	1	9	0.00	0.00	0
35	0.80	462	2339	72.65	8101.78	509100
35	0.85	143	262	53.99	1238.89	4980
35	0.90	58	109	20.00	185.02	688
35	0.95	1	9	0.00	0.00	0
40	0.80	514	3088	75.46	6127.18	492073
40	0.85	130	272	65.27	1863.58	27683
40	0.90	50	116	16.54	103.14	364
40	0.95	0	5	0.00	0.00	0
45	0.80	589	3001	61.81	9178.94	263754
45	0.85	133	266	66.55	9739.74	464795
45	0.90	52	107	14.83	132.31	355
45	0.95	2	8	0.00	0.00	0
50	0.80	482	3401	66.22	6604.52	440701
50	0.85	112	216	58.68	2707.67	98981
50	0.90	47	87	14.70	128.04	537
50	0.95	0	4	0.00	0.00	0

Table 3: *S*-Constraints

Perturbation (%)	With Warm Start				Without Warm Start			
	Time (sec)	No. of B&B Nodes	No. of Constraints Generated		Time (sec)	No. of B&B Nodes	No. of Constraints Generated	
			<i>I</i>	<i>S</i>			<i>I</i>	<i>S</i>
1	18.46	1076	40	13	1768.47	258200	3453	467
2	21.31	890	33	12	>3600	358200	3604	492
3	15.64	970	41	13	1574.90	268616	3489	486
4	19.60	1179	36	17	1553.76	242372	3611	519
5	19.32	1097	29	21	2366.64	456451	3219	510
6	15.96	747	41	11	1100.08	218851	3336	484
7	15.80	923	33	11	>3600	589901	3129	484
8	14.86	937	34	10	620.45	77790	3616	525
9	13.00	817	28	28	1191.33	158131	3639	519
10	13.68	808	27	16	449.26	75199	3427	492

Table 4: Summary Results: First Warm-Start Experiment

Source Instance	With Warm Start				Without Warm Start			
	Time (sec)	No. of B&B Nodes	No. of Constraints Generated		Time (sec)	No. of B&B Nodes	No. of Constraints Generated	
			<i>I</i>	<i>S</i>			<i>I</i>	<i>S</i>
scp41	17.41	1116	29	14	948.33	180878	3253	531
scp42	251.32	28299	337	126	>3600	396199	3170	530
scp43	40.01	3023	160	82	>3600	374501	3257	448
scp44	159.45	15201	352	136	>3600	344601	3896	603
scp45	143.87	18595	342	114	2482.94	358395	3337	526
scp46	146.16	15560	243	116	>3600	386301	3455	591
scp47	83.24	6262	319	104	>3600	507222	2976	474
scp48	144.59	13024	385	120	>3600	405128	3302	459
scp49	323.77	28615	357	124	>3600	289107	3821	617
scp410	80.23	10879	235	97	>3600	631424	3298	550

Table 5: Summary Results: Second Warm-Start Experiment

Source Instance	Phase 1				Phase 2			
	Time (sec)	No. of B&B Nodes	No. of Constraints Generated		Time (sec)	No. of B&B Nodes	No. of Constraints Generated	
			<i>I</i>	<i>S</i>			<i>I</i>	<i>S</i>
scp35	120.00	11051	3225	378	48.51	3544	99	59
scp45	120.00	9501	5164	423	165.83	6216	88	108
scp50	120.00	9914	4738	385	167.60	7155	792	166

Table 6: 2-Phase Algorithm to solve (PSC)