INSTITUTT FOR FORETAKSØKONOMI

DEPARTMENT OF BUSINESS AND MANAGEMENT SCIENCE

FOR 12 2017

ISSN: 1500-4066

October 2017

Discussion paper

Stochastic programs with binary distributions: Structural properties of scenario trees and algorithms

BY

Vit Prochazka AND **Stein W. Wallace**

Norges Handelshøyskole

NORWEGIAN SCHOOL OF ECONOMICS

Stochastic programs with binary distributions: Structural properties of scenario trees and algorithms

Vit Prochazka and Stein W. Wallace
Department of Business and Management Science
NHH Norwegian School of Economics, Bergen, Norway

August 1, 2017

Abstract

Binary random variables often refer to such as customers that are present or not, roads that are open or not, machines that are operable or not. At the same time, stochastic programs often apply to situations where penalties are accumulated when demand is not met, travel times are too long, or profits too low. Typical for these situations is that the penalties imply a partition of the scenarios into two sets: Those that can result in penalties for some decisions, and those that never lead to penalties. We demonstrate how this observation can be used to efficiently calculate out-of-sample values, find good scenario trees and generally simplify calculations. Most of our observations apply to general integer random variables, and not just the 0/1 case.

Keywords: Stochastic programming, scenarios, binary random variables, algorithms.

1 Introduction

Stochastic programming is getting more attention in operations research and has many applications in real processes, where uncertainties are present, see for example Wallace and Ziemba (2005); King and Wallace (2012). Most approaches assume the uncertainty to be represented by a set of scenarios. In this paper we study the case of discrete random variables, with a focus on the multivariate Bernoulli (0/1) distribution. We shall do this using a stochastic version of the knapsack problem. However, the point of the paper is not to study the knapsack problem as such, rather it serves as a simple example suitable for demonstrating our issues that are more general. In this sense we use the same idea as Fortz et al. (2013).

In the case of n random variables, each taking the value 0 or 1, we have 2^n possible combinations. Every combination represents a scenario with a certain probability. Even though the number of possible scenarios is finite, it rises exponentially with the number of random variables and it quickly becomes numerically infeasible to solve most optimization problems using all scenarios. When we are able to solve a problem only for a limited number of scenarios, we need to choose a subset, i.e., construct a scenario tree. The most common approach is to sample the required number of scenarios. However, in this paper we introduce a heuristic based on iterative scenario selection that provides better results than pure sampling for a given number of scenarios. We also show that pure sampling can be strengthened by taking into account that many scenarios cannot lead to penalties caused by the knapsack being too small; the scenarios cannot lead us into the tail of the outcome distribution.

Technically, binary random variables are a subset of the discrete random variables. So in a sense, if we knew how to handle the general discrete case for generating scenarios, we would also have an approach for the binary case. In fact, binary should be simpler than the general case. But when considering what binary random variables represent in a real setting, it becomes clear that having a representative set is critical in a different way than some demand, for example, that appear in integer amounts. Typical applications are the existence of a customer in a routing or facility location problem, the presence of an edge in a network, whether a machine is up in a scheduling problem, and whether the weather allows a ship to sail. And it is often the combinatorial effects in the scenarios that drive the system. Just to give a few simplistic examples: if we by accident find ourselves in a situation where two customers never appear together in a scenario for a stochastic vehicle routing problem, the optimal solution will very likely take that into account and (if meaningful) put the two on the same route, since it knows that it will only need to visit one of them (hence making route length or total demand less varying). If two edges in a network design problem cannot (according to the scenarios) be down at the same time, the optimal solution will take that into account, for example to make sure that the probability of two given nodes being connected is rather high (by placing the two in parallel). Optimal solutions will always "look for" such anomalies and utilize them to create what seems to be very good solutions. So, in a sense, scenario generation for binaries becomes a combinatorial fight against such lack of good representation.

The quality of a decision is given by the true objective function value, that is, the evaluation of the solution over all scenarios. In order to stay coherent with the literature, we call it the out-of-sample value. In the case of the stochastic knapsack problem, and also many other problems, the objective function is given by some value term which is easily computed, and some penalty term related to for example financial risk or lack of capacity, what we might call *tail effects*. For the knapsack problem, the penalty occurs when the items we put into it are larger than the capacity of the knapsack. For such cases we introduce a procedure that computes the penalty in the minimal number of steps.

Despite the wide range of real applications, we are not aware of any paper dealing with the scenario generation process for binary random variables. For a general discusssion of scenario generation, see for example Chapter 4 of King and Wallace (2012).

2 Stochastic knapsack problem

We consider a stochastic version of the knapsack problem, where $\mathcal{I}$ is the set of n items, item i having a value c_i and size w_i, with the knapsack having a capacity of W. All items we want to use must be picked in the first stage and if the total size is too large, we pay a unit penalty of d in the second stage. The aim is to maximize the expected profit.

Let S be the set of scenarios, having $|S|$ elements. A scenario $s \in S$, with elements $s_i = 1$ if the i^{th} item is present, 0 if not, is associated with a probability π_s.

For a decision vector $x \in \{0,1\}^n$, where $x_i = 1$ if item i is picked, 0 otherwise, we formulate the optimization problem

$$\max_{x\in\{0,1\}^n} \sum_s \pi_s \Big(\sum_i s_i c_i x_i - d\big(\sum_i s_i w_i x_i - W\big)^+ \Big), \tag{1}$$

where X^+ takes value X if $X \geq 0$, 0 otherwise. This problem can be rewritten as a standard stochastic integer linear problem by introducing a new variable e_s for exceeded capacity of the knapsack in scenario s. Problem (1) then becomes

$$\max_{x\in\{0,1\}^n} \sum_s \pi_s \Big(\sum_i s_i c_i x_i - d\, e_s \Big) \tag{2}$$

$$\text{s.t} \qquad e_s \geq \sum_i s_i w_i x_i - W \qquad \forall s \in S \tag{3}$$

$$e_s \geq 0 \qquad \forall s \in S \tag{4}$$

If we have marginal probabilities of appearances p_i for each item, either as a starting point or computed from the identity $\sum_s \pi_s s_i = p_i$, we can reorganize terms in the objective function to obtain a simplified form:

$$\sum_s \pi_s \Big(\sum_i s_i c_i x_i - d\, e_s \Big) = \sum_i p_i c_i x_i - d \sum_s \pi_s e_s. \tag{5}$$

In the independent case, we can compute the probability of a scenario by formula

$$\pi_s = \prod_i \Big(s_i p_i + (1 - s_i)(1 - p_i) \Big). \tag{6}$$

In the general case we need $2^n - 1$ parameters to describe a multivariate Bernoulli distribution. This may of course cause practical difficulties, but is not the focus of this paper;

Our goal is to understand scenarios and algorithms. For simplicity, we shall therefore in the following assume that items appear independently. The general case with dependencies is discussed in Section 5.

If not stated otherwise, we use the notation $\prod_i$ for $\prod_{i\in\mathcal{I}}$ (as already used above). The same holds for the summation $\sum$ and indices of scenarios s belonging to S.

3 Out-of-sample evaluation

Out-of-sample (o-o-s) evaluation is a way to provide the true (or approximately true) value of the objective function for a given solution, principally using the whole distribution. Only occasionally can that be done exactly for continuous distributions (for a case when that was possible, see Zhao and Wallace (2014)). Therefore, the o-o-s value is normally obtained using a large number of sampled scenarios which approximates the true distribution extremely well. With discrete distributions, the number of possible scenarios is finite. However, to stay consistent with the literature, we shall call the evaluation o-o-s, whether we use a very large sample or the full discrete distribution when calculating the true value. For the complete scenario set S, and a given a solution x, the o-o-s value $f(x)$ is given by:

$$f(x) = \sum_s \pi_s \Big(\sum_i s_i c_i x_i - d\, e_s \Big) = \sum_i p_i c_i x_i - d \sum_s \pi_s e_s. \tag{7}$$

The evaluation of $f(x)$ in (7) is much faster than solving (2) - (4). That means that there exist many cases where it is impossible to find the solution $x^\star$ for a given S, but it is possible to evaluate $f(x)$ by simple enumeration. However, sometimes a straightforward evaluation over all scenarios is still too time consuming. For those cases, we introduce a method to find the o-o-s value in a recursive manner using a minimal number of calculations.

In (7), the expected profit is split into two terms. The first is simply the expected profit from the selected items. This is given by the sum of expected contributions from the selected items. The second term represents the penalty for exceeding the capacity of the knapsack.

As we can see from the last term, we do not pay a penalty in all scenarios. Let us denote by $Q(x)$ the set of scenarios for which we pay a penalty given a decision x, that is

$$Q(x) = \{s | \sum_i s_i w_i x_i > W\}. \tag{8}$$

For a decision x we define a relation $\geq_x$ for two scenarios s^1 and s^2 and we state the obvious lemma that is used later in this section.

Definition 3.1 *We say that $s^1 \geq_x s^2$ if $s^1_i x_i \geq s^2_i x_i$ for all $i = 1, \ldots, n$.*

Lemma 3.1 *If $s^1 \geq_x s^2$ and $s^1 \notin Q(x)$ then $s^2 \notin Q(x)$.*

This lemma says that if a scenario s^1 does not generate a penalty for a particular decision x, then a scenario s^2, which does not include any item that is not present in the scenario s^1, also does not generate a penalty.

Vice versa, we could say that if a scenario s^1 generates a penalty, then a scenario s^2 which includes all selected items from s^1 (that is $s^2 \geq_x s^1$) also generates a penalty. This result has, however, not been used in the paper.

Recursive implementation of o-o-s

A straightforward way to obtain the o-o-s value $f(x)$, for a given x, is to evaluate the function over all scenarios, that is, to perform a full enumeration of the scenarios. However, this might be impossible or very time-consuming if the number of scenarios is large. Therefore we implement a procedure that only visits those scenarios that result in penalties.

We shall utilize the simple observation that to evaluate the penalty term for a solution x, it is sufficient to compute the penalties for scenarios that belong to $Q(x)$. That is,

$$\sum_{s \in S} \pi_s e_s = \sum_{s \in Q(x)} \pi_s e_s$$

.

Let us notice that if an item i is not chosen in the solution x, that is, $x_i = 0$, two scenarios that differ only in the appearance of item i will generate the same exceeded capacity. Thus we aggregate these two scenarios, this practically means that for o-o-s evaluations we ignore the random variables corresponding to items that are not selected. Let us denote the set of selected items $\mathcal{I}_x = \{i \in \mathcal{I} | x_i = 1\}$ and the set of all aggregated scenarios for the selected items S_x. Therefore, a scenario $s \in S_x$ is described as a vector of length $|\mathcal{I}_x|$. Let us further denote by $\overline{1} = (1, 1, \ldots, 1)$ the scenario with all items from $\mathcal{I}_x$ present.

For items that are selected, the relation $\geq_x$ is a partial order over the set of scenarios, i.e., a relation which is reflexive, antisymmetric, and transitive. We shall use Algorithm 1 to generate a tree, based on this partial order, with root node $\overline{1}$. We can search over this tree in a recursive manner to visit all scenarios in $Q(x)$ and thus compute the basis for the penalty, namely $P_x = \sum_{s \in Q(x)} = \pi_s e_s$. Lemma 3.1 gives the condition for stopping the depth first search in a particular branch. By $s = s^{\mathrm{I}}_{i \to 0}$ in Algorithm 1 we mean that scenario s is obtained from scenario s^{I} by changing the i^{th} item in the vector from 1 to 0, while the rest of the items remain unchanged.

The procedure described in Algorithm 1 provides the o-o-s value in the minimal number of steps – we evaluate all scenarios for which the capacity of the knapsack is exceeded, and we stop the search in the branch immediately after discovering that all remaining scenarios in that branch are not penalized for exceeding the capacity.

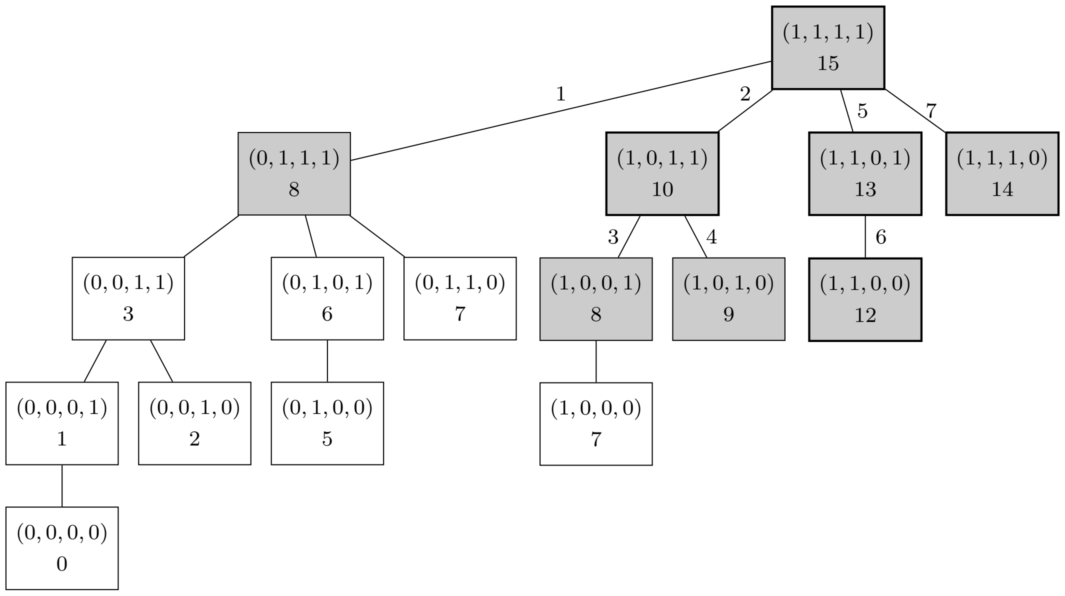

Figure 1: Demonstration of the o-o-s recursive procedure.

In order to minimize the number of nodes (scenarios) that we need to consider, we sort the items according to size from largest to smallest. Even though the number of scenarios in $Q(x)$ remains the same regardless of the sorting, we reduce the number of cases where a scenario is explored, but turns out not to exceed the capacity. For a proof that sorting is indeed optimal in this procedure, see Appendix B.

Example

Let us demonstrate the procedure on a simple example. We assume we have selected four items with ordered sizes $w = (7, 5, 2, 1)$ and that the capacity of the knapsack is $W = 9$. In Figure 1, we show the tree that is created based on the partial order $\geq_x$.

Each box starts with a scenario, for example $(0, 1, 1, 1)$ (implying that items with sizes 5, 2 and 1 are selected), and their sum of weights in the second row. The scenarios that are explored are those with a gray background, the scenarios from $Q(x)$ are framed by thick line. The order in which scenarios are explored is shown above the edges. In this example, we need to investigate eight scenarios in order to find the set $Q(x)$. Let us note, that if items were sorted in reverse order, the procedure would have to explore 13 scenarios to reveal the set $Q(x)$.

Scenario contribution to the objective value

Another outcome of the procedure that might be useful is the knowledge of the contribution of each scenario to the objective function. This contribution is defined as

Algorithm 1 Computing the out-of-sample value

```
1:  function ExceededCapacity($j, s^{\mathrm{I}}, \pi^{\mathrm{I}}, P_x$)
2:      for $i := j$ to $|\mathcal{I}_x|$ do
3:          $s := s^{\mathrm{I}}_{i \to 0}$
4:          if $e_s > 0$ then                          ▷ $e_s$ is the exceeded capacity at scenario $s$
5:              $\pi_s := \frac{(1-p_i)}{p_i}\pi^{\mathrm{I}}$
6:              $P_x := P_x + \pi_s e_s$
7:              if $i < |\mathcal{I}_x|$ then
8:                  $P_x :=$ ExceededCapacity($i+1, s, \pi_s, P_x$)
9:      return $P_x$

10: Sort the items such that $w_i \geq w_j$ for all $i < j$
11: Set initially: $s := \bar{1}$;  $\pi_s := \prod_{i \in \mathcal{I}_x} p_i$;  $P_x := \pi_s e_s$;  $j := 1$
12: $P_x :=$ ExceededCapacity($j, s, \pi_s, P_x$)

13: Compute the o-o-s value: $f(x) = \sum_{i \in \mathcal{I}_x} p_i c_i - d\, P_x$
```

$$\gamma_s(x) = \sum_{i \in \mathcal{I}_x} s_i c_i p_i + \pi_s e_s \tag{9}$$

for all $s \in Q(x)$. For scenarios that do not belong to the set $Q(x)$, the contribution is simply $\gamma_s(x) = \sum_{i \in \mathcal{I}_x} s_i c_i p_i$. The overall o-o-s value is then $f(x) = \sum_s \gamma_s$.

This can be useful, for example, in a setting where we have solved a problem with a limited set of scenarios, and want to find out what non-included scenarios contribute to the o-o-s value, in order to possibly include them in our scenario set.

We can easily compute the contribution from the scenarios in $Q(x)$ in Algorithm 1, for example between rows 5 and 6.

4 Exact reformulation of the stochastic knapsack problem

Let us define the set Q to be the set of all scenarios that lead to a penalty for some feasible decision x. This set is therefore given as $Q = \cup_x Q(x)$. To get this set, let us use the following:

Definition 4.1 *We say that $x^1 \geq x^2$ if $x^1_i \geq x^2_i$ for all $i = 1, \ldots, n$.*

Lemma 4.1 *If $x^1 \geq x^2$ then $Q(x^2) \subseteq Q(x^1)$.*

Let us take the decision $\overline{x}$, where all items are selected, that is, $\overline{x}_i = 1$ for all $i = 1, 2, \ldots, n$. It holds that $\overline{x} \geq x$ for all $x \in \{0,1\}^n$. Therefore, $Q(x) \subseteq Q(\overline{x})$ for all x, and hence $Q = Q(\overline{x})$. This gives us a way to generate all scenarios that may lead to a penalty in a minimal number of steps - simply by a recursive procedure built just as Algorithm 1, and initiated with the decision $\overline{x}$. See Algorithm 2 for the detailed description.

Scenarios not in Q never lead to a penalty. Therefore, we do not have to evaluate probabilities scenarios outside Q, and we do not need variables e_s for these scenarios since they are never positive. Thus we reduce the number of constraints and input data and solve the following reduced form of the problem:

$$\max_{x \in \{0,1\}^n} \sum_i p_i c_i x_i - d \sum_{s \in Q} \pi_s e_s \tag{10}$$

$$\text{s.t} \qquad e_s \geq \sum_i s_i w_i x_i - W \qquad \forall s \in Q \tag{11}$$

$$e_s \geq 0 \qquad \forall s \in Q, \tag{12}$$

Algorithm 2 Determination of Q

```
 1: function FindSet($j, s^{\mathrm{I}}, \pi^{\mathrm{I}}, Q$)
 2:     for $i := j$ to $n$ do
 3:         $s := s^{\mathrm{I}}_{i \to 0}$
 4:         if $e_s > 0$ then                    ▷ $e_s$ is the exceeded capacity at scenario $s$
 5:             $\pi_s := \frac{(1-p_i)}{p_i} \pi^{\mathrm{I}}$
 6:             $Q := Q \cup s$                  ▷ We also store the corresponding value $\pi_s$.
 7:             if $i < n$ then
 8:                 $Q :=$ FindSet($i+1, s, \pi_s, Q$)
 9:     return $Q$

10: Sort the items such that $w_i \geq w_j$ for all $i < j$
11: Set initially: $s := \overline{1}$;  $\pi_s := \prod_{i \in \mathcal{I}} p_i$;  $Q = \{\overline{1}\}$;  $j := 1$
12: $Q :=$ FindSet($j, s, \pi_s, Q$)
```

The significance of the reduction of constraints and input data depends on the size of Q relative to the size of S. We present a numerical experiment showing the dependency in Figure 2. We have looked at a large number of cases, all with 10 items. As we move right on the horizontal axis, the variation in size increases, but the average item remains at 10. We show the results for knapsack capacities from $W = 10$ (on the top) to $W = 90$ (on the bottom).

The symmetry around the central ($W = 50$) case is striking. First notice that the points for $W = 50$ are perfectly lined up. For every scenario in S, there exists an "inverse scenario", which has the exact opposite combination of 1's and 0's: (0, 0, 1, 0, 1) versus (1, 1, 0, 1, 0). These two scenarios combined have a total size of 100, exactly twice that of the knapsack. Therefore, one of the scenarios must go into Q and the other one must stay outside. The only exception is when both scenarios have the same weight, namely the knapsack capacity. Then both stay outside. However, as we generated weights randomly from a continuous distribution when creating the figure, this has zero probability of happening (in the ideal case).

Similarly, by using these "inverse scenarios", we can explain the symmetry between the $|Q|/|S|$ ratios of 0.9 and 0.1, of 0.7 and 0.3, etc, observed in the figure. If we look at the knapsack capacities 40 and 60, and the total sum of item weights is 100, we observe the following: if a scenario goes into Q for 40, its inverse goes into $S \setminus Q$ for 60 and so on. Again this is distorted only by some special cases when weights of the selected items equals the knapsack capacity. But as we sampled from continuous distributions this never happens.

As we can see from Figure 2, the reduction in the number of scenarios can be substantial if we utilize Q rather than S, especially when the size of the knapsack is close to the total sum of weights of all items. Then, only a small number of scenarios leads to a penalty. This number is further reduced when the variance in sizes is small.

5 Dependent case

For simplicity, we assumed that items appear independently of each other. This is, however, rare in real applications. In this paragraph we discuss the case of a general multivariate Bernoulli distribution where interactions between random variables play a role and the probability of a specific scenario is not given by a simple multiplication of probabilities associated with (non-)appearances of single items. We do not provide a comprehensive description of the multivariate Bernoulli distribution, but we discuss the properties that need to be taken into account when dealing with correlated variables.

The crucial observation is that the partial order from Definition 4.1 and Lemma 4.1 hold regardless of the underlying probability distribution. Since this lemma provides the stopping criterion for Algorithms 1 and 2, the frameworks of both algorithms remain unchanged. However, we need to adjust the computation of probabilities when creating new scenarios, denoted by $s = s^{\mathrm{I}}_{i \to 0}$. One possible description of the multivariate case, with demonstrated modifications in the procedures, is shown in Appendix A.

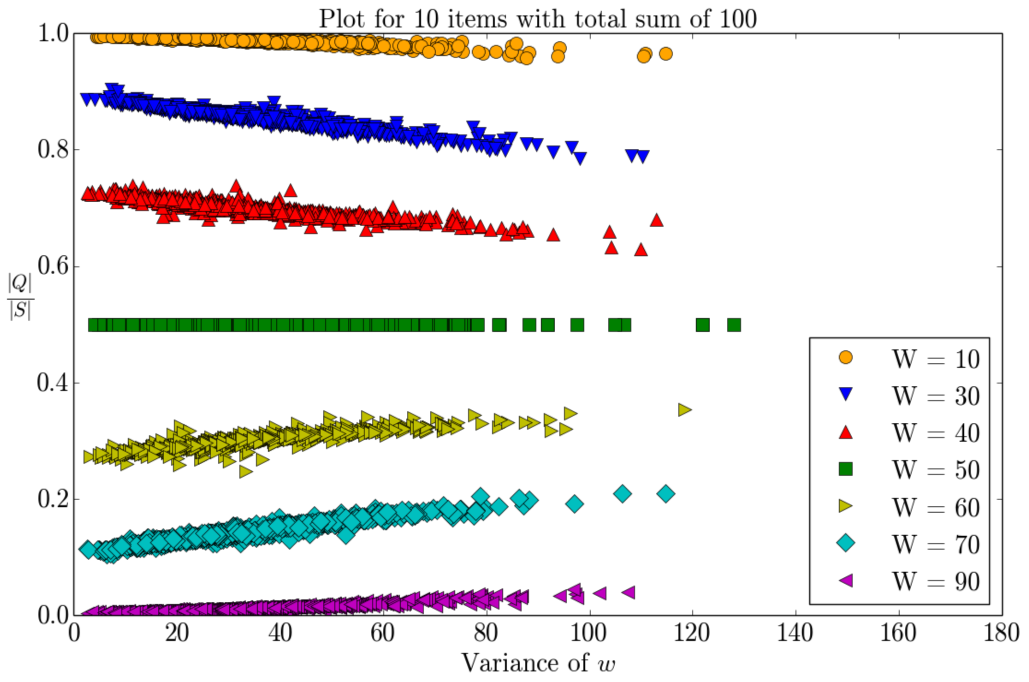

Figure 2: The number of scenarios in Q, relative to the number of scenarios in S for different knapsack capacities as the variation in item sizes increases.

Further derivations and properties of the multivariate Bernoulli distribution can be found in Dai et al. (2013) or Whittaker (2009).

6 Other applications

The structures just observed for the simple stochastic knapsack problem also appear in other settings. Consider a network design problem with stochastic existence (breakdown) of edges, and where the task is to satisfy demand at some nodes. If demand is not satisfied, we pay a penalty proportional to the unsatisfied demand. The first-stage decision is to invest in new edges. Assume for the moment that there are no flow costs in the second stage, just penalties for unsatisfied demand. The scenario used to initiate Algorithm 1 is “All edges down”, while the decision that we denoted as $\overline{1}$ in Algorithm 2, that is, the decision that will always lead to a penalty, is ”not to buy any additional edges”. Then if we have a scenario that leads to a penalty with this decision, every scenario where no further edges are present, automatically must lead to a penalty as well. Vice versa, if some scenario does not lead to a penalty, every other scenario that includes the same edges (and possible some additional) are also penalty free. With this in mind, we end up with a partial order of the scenarios and a subset Q just as for the knapsack case. Calculating the penalty is somewhat more complicated, but can still be done efficiently.

For a given feasible solution, the root node in the tree cooresponding to Figure 1 is the case when no edges are working. Then as we proceed through the evaluation, edges are added, one by one. In the root node of the tree we shall have to solve a min cost network flow problem (for a given first-stage decision and no edges working). Then one additional edge is opened. The previous optimal flow is now feasible in this new looser problem, and a warm start can take place. So it will never be necessary to start from scratch when calculating the penalties; warm starts for min cost network flow problems will suffice.

If there are flow costs in addition to penalty costs in the second stage, a scenario set Q will still exist if the penalty is so high that ordinary flow is always preferrred over penalties. If not, there is no clear-cut set Q (as there is no clear-cut tail), but the partial order remains, and so does the recursive way of solving the second-stage problem using warm starts.

Just as for the knapsack problem, the difference between S and Q (when it exists) can be both negligible or substantial based on the input data. For stochastic network design, Q is small when the network provides high flexibility. That is, the situation where even if some, but not all, edges break, it is still possible to satisfy demand at nodes by using other edges. For instance, highly connected graphs, like city streets, provide such flexibility. On the other hand, poorly connected networks, where one single break might lead to a penalty, no matter what happens to the other edges, provide little flexibility. Therefore, the number of scenarios in Q is high (all combinations of 1's and 0's of remaining edges). So when dealing with such networks, it is not particularly beneficial to use the presented methodology – the ratio $|Q|/|S|$ is high.

6.1 Sampling

In case we wish to sample scenarios we can either first generate Q and then sample in a controlled way from Q or $S \setminus Q$ in a controlled way. Or we can sample and then check where the sample belongs (which is equivalent to checking if a sample generates a penalty). This way we can achive different densities of scenarios in the tail and outside the tail.

6.2 Integer random variables

Most of what we have said apply also to integer random variables if we have a model where there are tail effects; random integer demand that ought to be satisfied, and where there is a penalty for not doing so; network flow problems where edge capacities are random and integer. The scenarios will again be split into two sets, one that cannot lead to penalties and one that can. The algorithms have obvious extensions in this case.

7 Conclusion

The main point of this paper has been to show that many two-stage stochastic programs with special tail costs on the output distribution (which could correspond to tail risk measures) possess a specific structure that allows the scenario set to be split into two sets: those scenarios that may lead to a penalty (depeneding on the optimal solution) and those that may not, whatever happens. We demonstrate the situation on a stochastic knapsack problem, but indicate how many related problems, such as network design, are similar.

The structure and the split of the scenario set allows for an efficient way to calculate the out-of-sample value for the problam at hand, and can also be used in a setting of sampling if we wish more scenarios in the tail of the output distribution, rather than evently spread all over as normal sampling would do.

Our procedures are more powerful when the number of scenarios that cannot go to the tail is large. We show by numerical testing as well as verbal arguments when that will be the case, and hence, when our approach is particularly useful.

So a practical way of proceeding whenever our approach potentially applies would be

- Try to solve the problem by using full enumeration of scenarios. If this is possible, there is no need to use anything more advanced.
- If not, try to use our overall procedure, thus still solving the problem exactly. If this is numerically feasible, all is well.
- If not, use some heuristic to to find feasible solutions, possibly using ideas from this paper in terms of sampling and scenario contributions. Check the quality of these solutions by full enumeration. If this is possible, there is no need to be more advanced.
- If not, use the out-of-sample evaluation procedure from this paper in combination with the heuristic.
- If not, use some out-of-sample estimation approach.

References

Dai, B., Ding, S., and Wahba, G. (2013). Multivariate Bernoulli distribution. *Bernoulli*, 19(4):1465–1483.

Fortz, B., Labbe, M., Louveaux, F., and Poss, M. (2013). Stochastic binary problems with simple penalties for capacity constraints violations. *Mathematical Programming*, 138:199–221.

King, A. J. and Wallace, S. W. (2012). *Modeling with Stochastic Programming.* Springer Series in Operations Research and Financial Engineering. Springer.

Wallace, S. W. and Ziemba, W. T., editors (2005). *Applications of Stochastic Programming.* MPS-SIAM Series on Optimization. SIAM, Philadelphia.

Whittaker, J. (2009). *Graphical Models in Applied Multivariate Statistics.* Wiley Publishing.

Zhao, Y. and Wallace, S. W. (2014). Integrated facility layout design and flow assignment problem under uncertainty. *INFORMS Journal on Computing*, 26(4):798–808.

Appendix A Multivariate Bernoulli distribution

Let $s = (s_1, s_2, \ldots, s_n)$ be a scenario, that is a realization of an n-dimensional random vector from the Cartesian product space $\{0,1\}^n$. There are several ways how to uniquely describe the desired distribution. The most common way is by providing the joint probability mass function. For the case of $n = 2$ and a random vector $Y = (Y_1, Y_2)$ with realizations $y = (y_1, y_2)$ we have:

$$P(Y = y) = p(y_1, y_2) = p_{00}^{(1-y_1)(1-y2)} p_{10}^{y_1(1-y_2)} p_{01}^{(1-y_1)y_2} p_{11}^{y_1 y_2}, \tag{13}$$

where naturally $p_{ij} = P(Y_1 = i, Y_2 = j)$.

It is possible to write the logarithm of the density function.

$$\begin{aligned}\log p(y_1, y_2) &= \log p_{00} + y_1 \log \frac{p_{10}}{p_{00}} + y_2 \log \frac{p_{01}}{p_{00}} + y_1 y_2 \log \frac{p_{11} p_{00}}{p_{01} p_{10}} \\ &= u_\phi + y_1 u_1 + y_2 u_2 + y_1 y_2 u_{12}. \end{aligned} \tag{14}$$

The coefficients u_ϕ, u_1, u_2, u_{12} are called *u-terms* and are widely used for studying interactions among random variables in the multivariate Bernoulli distribution. Therefore, they can be considered as an alternative way to define the distribution.

From the u-terms we can compute the logarithms of probabilities:

$$\log p_{00} = u_\phi \tag{15}$$

$$\log p_{10} = u_\phi + u_1 \tag{16}$$

$$\log p_{01} = u_\phi \qquad + u_2 \tag{17}$$

$$\log p_{11} = u_\phi + u_1 + u_2 + u_{12} \tag{18}$$

In the case of $n = 3$, the logarithm of the density function by using u-terms notation is

$$\log p(y_1, y_2, y_3) = u_\phi + y_1 u_1 + y_2 u_2 + y_3 u_3 + y_1 y_2 u_{12} + y_1 y_3 u_{13} + y_2 y_3 u_{23} + y_1 y_2 y_3 u_{123}, \tag{19}$$

so we can derive expressions for selected u-terms (similarly for u_2, u_3, u_{13} and u_{23}):

$$u_\phi = \log p_{000} \tag{20}$$

$$u_1 = \log \frac{p_{100}}{p_{000}} \tag{21}$$

$$u_{12} = \log \frac{p_{000}p_{110}}{p_{100}p_{010}} \tag{22}$$

$$u_{123} = \log \frac{p_{100}p_{010}p_{001}p_{111}}{p_{000}p_{011}p_{101}p_{110}} \tag{23}$$

We omit the general notation of the probability density function for a case with n random variables. We hope that the idea of how the density would look is evident from the given examples. It can be also observed that for the general distribution, there are 2^n parameters (either probabilities or u-terms) with the additional condition that the sum of the probabilities must equal one. So we need $2^n - 1$ parameters to fully describe the distribution.

When the probabilities are known, there is no need to recompute the probabilities of scenarios when we deal with the transition $s = s^{\mathrm{I}}_{i\to 0}$. A different situation is in the case of describing the distribution by u-terms. Let us assume we know all u-terms, each u-term is associated with one element of the power set of $\mathcal{I}$, denoted $\mathcal{P}(\mathcal{I})$. Further we denote $\mathcal{F}_s \subseteq \mathcal{P}(\mathcal{I})$, which includes all subsets of appearing items. $\mathcal{F}_s$ plays a role as a collection of indices of items that appear in the scenario s and are used for computing the scenario probability as is shown in Algorithm 3 in the function ComputeProbability. As initial values for the scenario $s = \overline{1}$ in algorithms 1 and 2 we set $\mathcal{F}_s = \mathcal{P}(\mathcal{I})$ and compute the corresponding probability as it is described in the function InitialValues.

Algorithm 3 Modification of computing probabilities and initial values

```
1:  function ComputeProbability($i, s^{\mathrm{I}}, \mathcal{F}_{s^{\mathrm{I}}},$)
2:      $s := s^{\mathrm{I}}_{i\to 0}$
3:      $\mathcal{F}_s = \{F \in \mathcal{F}_{s^{\mathrm{I}}} \mid i \notin F\}$
4:      $\pi_s = \exp\Big(\sum_{F\in\mathcal{F}_s} u_F\Big)$
5:      return $s, \pi_s$

6:  function InitialValues($\mathcal{I}$, u-terms)
7:      $s = \overline{1}$
8:      $\mathcal{F}_s = \mathcal{P}(\mathcal{I})$
9:      $\pi_s = \exp\Big(\sum_{F\in\mathcal{F}_s} u_F\Big)$
10:     return $s, \pi_s$
```

Appendix B

We prove that sorting of items in Algorithm 1 is optimal.

Let us assume two orderings of the same set of items, represented by vectors of weights w^A and w^B with swapped items at positions k and l, that is $w_k^A = w_l^B$ and $w_l^A = w_k^B$. Without loss of generality, let us assume $k < l$ and $w_k^A \geq w_l^A$. Then, the recursive procedure ExceededCapacity, described in Algorithm 1, generates for both vectors w^A and w^B and for $i < m$ or $i \geq n$ fundamentally equal scenarios (the same items are present). For $i \in \{k, k+1, \ldots, l-1\}$, we get $\sum_{m=i}^{n} w_m^A s_m \leq \sum_{m=i}^{n} w_m^B s_m$ and $s_k = 0$ for every scenario s.

Therefore, $s \in Q$ for w^A implies $s \in Q$ for w^B at every branch. However, it may happen that $s \in Q$ for w^B, but $s \notin Q$ for w^A. In such case, the recursive procedure is called again for w^B in the particular branch, but not for w^A. Thus the total number of times the procedure is called for w^B is higher or equal to the number of times it is called for w^A.

Let us take some arbitrary ordering and apply, for instance, the bubble sort method where two items are swapped whenever $w_i < w_j$ for $i < j$. Such a method results in a descending order of items, and since for every swap we can only improve (decrease) the number of times the procedure ExceededCapacity is called, working with an ordered vector of items in Algorithm 1 is optimal in this respect.