BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF MATHEMATICS
ÚSTAV MATEMATIKY

ADVANCED DECOMPOSITION METHODS IN STOCHASTIC CONVEX OPTIMIZATION

POKROČILÉ DEKOMPOZIČNÍ METODY VE STOCHASTICKÉ KONVEXNÍ OPTIMALIZACI

DOCTORAL THESIS
DIZERTAČNÍ PRÁCE

AUTHOR — **Ing. Jakub Kůdela**
AUTOR PRÁCE

SUPERVISOR — **RNDr. Pavel Popela, Ph.D.**
ŠKOLITEL

BRNO 2019

ABSTRACT

When working with stochastic programming problems, we frequently encounter optimization problems that are too large to be processed by routine methods of mathematical programming. However, in some cases the problem structure allows for a use of specialized decomposition methods that (when utilizing said structure) can be employed to efficiently solve very large optimization problems. This work focuses on two classes of stochastic programming problems that have an exploitable structure, namely two-stage stochastic programming problems and chance constrained problems, and the advanced decomposition methods that can be used to solve optimization problems in these two classes. We describe a novel warm-start cuts for the Generalized Benders Decomposition, which is used as a methods for the two-stage stochastic programming problems. For the class of chance constraint problems, we introduce an original decomposition method, that we named the Pool & Discard algorithm. The usefulness of the described decomposition methods is demonstrated on several examples and engineering applications.

KEYWORDS

stochastic optimization, stochastic programming, decomposition methods, two-stage stochastic programming problems, chance constrained problems

ABSTRAKT

Při práci s úlohami stochastického programování se často setkáváme s optimalizačními problémy, které jsou příliš rozsáhlé na to, aby byly zpracovány pomocí rutinních metod matematického programování. Nicméně, v některých případech mají tyto problémy vhodnou strukturu, umožňující použití specializovaných dekompozičních metod, které lze použít při řešení rozsáhlých optimalizačních problémů. Tato práce se zabývá dvěma třídami úloh stochastického programování, které mají speciální strukturu, a to dvoustupňovými stochastickými úlohami a úlohami s pravděpodobnostním omezením, a pokročilými dekompozičními metodami, které lze použít k řešení problému v těchto dvou třídách. V práci popisujeme novou metodu pro tvorbu "warm-start" řezů pro metodu zvanou "Generalized Benders Decomposition", která se používá při řešení dvoustupňových stochastických problémů. Pro třídu úloh s pravděpodobnostním omezením zde uvádíme originální dekompoziční metodu, kterou jsme nazvali "Pool & Discard algoritmus". Užitečnost popsaných dekompozičních metod je ukázána na několika příkladech a inženýrských aplikacích.

KLÍČOVÁ SLOVA

stochastická optimalizace, stochastické programování, dekompoziční metody, úlohy dvoustupňového stochastického programování, úlohy s pravděpodobnostním omezením

KŮDELA, Jakub. *Advanced Decomposition Methods in Stochastic Convex Optimization*: doctoral thesis. Brno: Brno University of Technology, Faculty of Mechanical Engineering, Institute of Mathematics, 2019. 92 p. Supervised by RNDr. Pavel Popela, Ph.D.

DECLARATION

I declare that I have written my doctoral thesis on the theme of "Advanced Decomposition Methods in Stochastic Convex Optimization" independently, under the guidance of the doctoral thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the doctoral thesis I furthermore declare that, as regards the creation of this doctoral thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

Jakub Kůdela

PREFACE

This thesis is the summary of the more interesting work that I have done over the 4 years of my doctoral studies. It is partly composed of already published articles where I can claim to be the main (although not the only) author – these are the Chapters 2, 3, and 5. The other two chapters are Chapter 1 (Introduction) and Chapter 4 – this is the largest section, containing new (not published before) work; and, in my opinion, containing the most interesting results.

I would like to use this part of the text for the acknowledgements. First and foremost, I want to thank my parents for their endless support throughout my whole life. Then, I wish to thank my colleagues from the departments of Mathematics and Process Engineering for the collaborations we successfully run. Also, I am extremely grateful that I could (at least in the first few years) regularly attend the seminar of Stochastic Programming and Approximation at the Department of Probability and Mathematical Statistics at Charles University – the presentations and discussions helped me enormously. Likewise, I would like to mention the books that were most inspirational for me – these are Convex Optimization [17] by Boyd and Vandenberghe, Fundamentals of Convex Analysis [47] by Hiriart-Urruty and Lemaréchal, and Introduction to Stochastic Programming [14] by Birge and Louveaux.

I want to single out two others for special acknowledgment. My supervisor, Pavel Popela, that always provided me with a useful advice or a necessary dialectic. And, lastly, Tereza – I could not imagine a better companion for this journey.

Brno ....

Jakub Kůdela

Contents

CHAPTER 1

Introduction to Stochastic Programming

There are numerous introductory texts for stochastic programming. Among the ones we can recommend are the well-known books [50], [92], [14], and [98]. Since the purpose of this text is not to present a new insight into stochastic programming as such, we will use for our introduction some parts and arguments from (in our opinion) an exceptional text [83].

1.1 | Introduction

In our daily life we perpetually make decisions under uncertainty and, moreover, we would certainly like to make these decisions in a reasonably optimal way. We can model the decision making as specifying an objective function $F(x, \xi)$, depending on decision vector $x \in \Re^{n_x}$ and vector $\xi \in \Re^{n_\xi}$ of uncertain parameters, and optimizing (say minimizing) $F(x, \xi)$ over x varying in a permissible (feasible) set $\mathcal{X} \subset \Re^{n_x}$. Needless to say, such an optimization problem is not well defined (i.e. cannot be solved as such) since our objective depends on an unknown and uncertain value of ξ. One possible way of dealing with this issue is to optimize the objective on average. That is, we assume that ξ is a random vector, with known probability distribution $\mathcal{P}$ having support $\Xi \subset \Re^{n_\xi}$, and the following optimization problem is formulated

$$\min_{x \in \mathcal{X}} \quad f(x) = \mathbb{E}_{\mathcal{P}}[F(x, \xi)], \tag{1.1.1}$$

where it is assumed that the considered expectations are well defined, i.e., $F(x, \cdot)$ is measurable and $\mathcal{P}$-integrable (see [98]).

In particular, the formulation (1.1.1) can be applied to the so-called two-stage stochastic programming problem with recourse (investigated in more detail in the following section and in chapters 2 and 3), pioneered by Beale [7] and Dantzig [24]. That is, the optimization problem at hand is divided into two decision stages. At the first stage (sometimes called the planning stage) one has to make a decision

on the basis of some available information. At the second stage (sometimes called the operational stage), after a realization of the uncertain data becomes known, an optimal second-stage decision is made. Such stochastic programming problem can be written in the form (1.1.1) with $F(x, \xi)$ including the optimal value of the second-stage problem.

It is important to note that in the formulation (1.1.1) all uncertainties are concentrated in the objective function while the feasible set $\mathcal{X}$ is supposed to be known (deterministic). In many cases the feasible set itself is delimited by constraints which depend on uncertain parameters. In some cases one can reasonably formulate such problems in the form (1.1.1) by introducing penalties for possible infeasibilities. Another option is to try to optimize the objective subject to satisfying constraints for all values of the unknown parameters in a chosen (uncertain) region. This approach is called robust optimization (see [9]) and has seen a significant increase in interest over the past two decades (following the advances in convex, conic and semidefinite programming, see [10]). Enforcing the satisfaction of the constraints for all possible realizations of random data may result in a too conservative solution and, more reasonably, one may try to satisfy the constraints with a high (close to one) probability instead. This leads to the chance, or probabilistic, constraints formulation which is going back to Charnes and Cooper [23].

There are a few natural questions which arise with respect to formulation (1.1.1) [83]:

(i) How do we know the probability distribution $\mathcal{P}$? In some cases one has historical data which can be used to obtain a reasonably accurate estimate of the corresponding probability distribution. However, this happens in rather specific/rare situations and often the probability distribution either cannot be accurately estimated or changes with time. Even worse, in many cases one deals with scenarios (i.e., possible realizations of the random data) with the associated probabilities assigned by a subjective judgment of a supposed "expert".

(ii) Why, at the first stage, do we optimize the expected value of the second-stage optimization problem? If the optimization procedure is repeated many times, with the same probability distribution of the data, then it could be argued by employing the Law of Large Numbers that this gives an optimal decision on average. However, if in the process, because of the variability of the data one looses all its capital, it does not help that the decisions were optimal on average.

(iii) How difficult is it to solve the stochastic programming problem (1.1.1)? Evaluation of the expected value function $f(x)$ involves calculation of the corresponding multivariate integrals. Only in rather specific cases it can be done

analytically. Therefore, typically, one employs a finite discretization of the random data which allows to write the expectation in a form of summation. Note, however, that if the random vector ξ has n_ξ elements that are independent of each other, each with just 3 possible realizations, then the total number of scenarios is 3^{n_ξ}, i.e., the number of scenarios grows exponentially fast with dimension n_ξ of the data vector.

It turns out that there is a close connection between questions (i) and (ii). As far as question (i) is concerned, one can approach it from the so-called distributionally robust point of view (see [111]). Suppose that a plausible family $\mathcal{B}$ of probability distributions, of the random data vector ξ, can be identified. Consequently, the "worst-case distribution" minimax problem

$$\underset{x\in\mathcal{X}}{\text{minimize}} \quad f(x) = \sup_{\mathcal{P}\in\mathcal{B}} \mathbb{E}_{\mathcal{P}}[F(x,\xi)], \tag{1.1.2}$$

is formulated. This worst-case approach to decision making is not new – it was also discussed extensively in the stochastic programming literature (see, e.g., [114], [29], [102]).

Question (ii) has also a long history. One can optimize a weighted sum of the expected value and a term representing variability of the second-stage objective function. For example, we can try to minimize

$$f(x) = \mathbb{E}[F(x,\xi)] + c\mathrm{Var}[F(x,\xi)], \tag{1.1.3}$$

where $c \geq 0$ is a chosen constant – this rather famous approach goes back to Markowitz [76]. The additional (variance) term in (1.1.3) can be viewed as a risk measure of the second-stage (optimal) outcome. It should be noted, however, that adding the variance term may destroy convexity of the function $f(\cdot)$ even if $F(x,\xi)$ is convex for all realizations of ξ (cf., [107]). An axiomatic approach to a mathematical theory of risk measures was suggested by [4]. Even more information about risk modeling and management can be found in the book [89].

1.2 | Complexity of Two-Stage Stochastic Programs

In this section we briefly examine question (iii) mentioned above, that is, how difficult is to solve a stochastic program. The problem (1.1.1) is a problem of minimizing a deterministic objective $f(x)$ that is given implicitly. We should expect that solving this problem is at least as hard as minimizing $f(x), x \in \mathcal{X}$, in the case where $f(x)$ is given explicitly, say by a "closed form analytic expression". Alternatively, we have at our disposal an "oracle" that provides us with the values and the derivatives of $f(x)$ at every queried point. For the problems of minimization of $f(x), x \in \mathcal{X}$, with

explicitly given objective, the known "efficiently solvable case" this is the convex programming case [17]. That is, $\mathcal{X}$ is a closed convex set and $f : \mathcal{X} \longrightarrow \Re$ is a convex function. It is known that generic convex programming problems satisfying mild computability and boundedness assumptions can be solved in polynomial time (see [84]). On the other hand, solving typical nonconvex problems turns out to be an NP-hard task [10]. Consequently, when speaking about conditions under which the stochastic program (1.1.1) is efficiently solvable, it is reasonable to assume that $\mathcal{X}$ is a closed convex set, and $f(\cdot)$ is convex on $\mathcal{X}$. Additionally, we gain from a technical point (and do not lose much from practical viewpoint) by assuming $\mathcal{X}$ to be bounded. These assumptions (plus mild technical conditions) would be sufficient to make (1.1.1) "easy" to solve, if $f(x)$ were given explicitly. However, in stochastic programming it makes no sense to assume that we can compute efficiently the expectation in (1.1.1), i.e., we have no way of obtaining an explicit representation of $f(x)$. If it were so, there would be no need to treat (1.1.1) as a stochastic program.

However, some stochastic programming problems of the form (1.1.1) can be solved reasonably efficiently by using Monte Carlo sampling techniques if the probability distribution of the random data is not "too bad" and if certain general conditions are satisfied (see [101] and [74]). Regarding the above statement, we should clarify what do we mean by "solving" stochastic programming problems. Let us consider, for example, two-stage linear stochastic programming problems with recourse. Such problems can be written in the form (1.1.1) with

$$\mathcal{X} = \{x : Ax = b, x \geq 0\} \quad \text{and} \quad F(x, \xi) = c^T x + Q(x, \xi),$$

resulting in the so called "first-stage" problem:

$$\begin{array}{ll} \underset{x \geq 0}{\text{minimize}} & c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi)] \\ \text{subject to} & Ax = b, \end{array} \tag{1.2.1}$$

where $Q(x, \xi)$ is the optimal value of the so called "second-stage" problem:

$$\begin{array}{ll} \underset{y \geq 0}{\text{minimize}} & q^T y \\ \text{subject to} & Tx + Wy \geq h. \end{array} \tag{1.2.2}$$

Here T and W are matrices of appropriate dimensions and $\xi \in \Re^{n_\xi}$ is a vector whose elements are composed from elements of vectors q and h and matrices T and W which, in the considered problem, are assumed to be random. If we assume that the random data vector has a finite number K of realizations (also called "scenarios") $\xi_k = (q_k, W_k, T_k, h_k)$ with respective probabilities $p_k, k = 1, ..., K$, then the two-stage problem can be written as one large linear programming problem, also

called a "deterministic equivalent" problem:

$$\begin{array}{ll} \underset{x,y_1,\ldots,y_K}{\text{minimize}} & c^T x + \sum_{k=1}^K p_k q_k^T y_k \\ \text{subject to} & Ax = b, \\ & T_k x + W_k y_k \geq h_k, \quad k = 1,\ldots,K, \\ & x \geq 0, y_k \geq 0, \qquad k = 1,\ldots,K. \end{array} \tag{1.2.3}$$

If the number of scenarios K is not "too large", then the above linear programming problem (1.2.3) can be solved to reasonable accuracy in an acceptable time. However, even a coarse discretization of the probability distribution of ξ typically results in an exponential growth of the number of scenarios with increase of the number n_ξ of random parameters (this is the particular manifestation of "the curse of dimensionality" in stochastic programming, see [50]). For example, assume that the components of the random vector ξ are mutually independently distributed each having a small number r of possible realizations. Then the size of the corresponding input data grows linearly in n_ξ (and r) while the number of scenarios $K = r^{n_\xi}$ grows exponentially.

It should noted that from a practical point of view, at least typically, it does not make sense to try to solve a stochastic programming problem with a high precision. Any numerical error resulting from an inaccurate estimation of the involved probability distributions, modeling errors, etc., can have far worse consequences than such an optimization error. The authors of [83] argue that two-stage stochastic problems can be solved efficiently with a reasonable accuracy provided that the following conditions are met:

(a) The feasible set $\mathcal{X}$ is fixed (deterministic).

(b) For all $x \in \mathcal{X}$ and $\xi \in \Xi$ the objective function $F(x, \xi)$ is real valued.

(c) The considered stochastic programming problem can be solved efficiently if the number of scenarios is not "too large".

When applied to two-stage stochastic programming, the above conditions (a) and (b) mean that the recourse is relatively complete (meaning that for every $x \in \mathcal{X}$ and every possible realization of random data, the second-stage problem is feasible) and the second-stage problem is bounded from below. The above condition (c) certainly holds in the case of two-stage linear stochastic programming with recourse.

To proceed further let us consider the following Monte Carlo sampling approach – assume that we can generate an i.i.d (independent and identically distributed) random sample $\xi^1, \ldots, \xi^N$ of N realizations of the considered random vector. Then we can estimate the expected value function $f(x)$ by the sample mean

$$\hat{f}_N(x) = \frac{1}{N} \sum_{j=1}^N F(x, \xi^j). \tag{1.2.4}$$

Consequently, we approximate the true problem (1.1.1) by the problem:

$$\min_{x \in \mathcal{X}} \quad \hat{f}_N(x). \tag{1.2.5}$$

We refer to (1.2.5) as the Sample Average Approximation (SAA) problem. The optimal value $\hat{v}_N$ and the set $\hat{S}_N$ of optimal solutions of the SAA problem (1.2.5) provide estimates of their true counterparts of problem (1.1.1). It should be emphasized that once the sample is generated, $\hat{f}_N(x)$ becomes a deterministic function and problem (1.2.5) becomes a stochastic programming problem with N scenarios $\xi^1, \ldots, \xi^N$ taken with equal probabilities $1/N$. It also should be mentioned that the SAA method is not an algorithm, as we still have to solve the SAA problem (1.2.5) by employing an appropriate algorithm.

By the Law of Large Numbers we have that $\hat{f}_N(x)$ converges (pointwise in x) w.p. 1 (with probability 1) to $f(x)$ as N tends to infinity (see [101]). Consequently, it is natural to expect for $\hat{v}_N$ and $\hat{S}_N$ to converge to their counterparts of the true problem (1.1.1) w.p. 1 as N tends to infinity. Such convergence results can be proved under mild regularity conditions (see [52]). However, for a fixed $x \in \mathcal{X}$, convergence of $\hat{f}_N(x)$ to $f(x)$ is notoriously slow. By the Central Limit Theorem it is of order $O(N^{-1/2})$. This rate of convergence can be improved, sometimes even significantly, by different variance reduction methods. Nonetheless, by using the Monte Carlo (or Quasi-Monte Carlo) techniques one cannot evaluate the expected value $f(x)$ very accurately.

Since, at least generally speaking, nonconvex problems are hard to solve already in the deterministic case, when discussing the question of what is and is not easy in stochastic programming, it makes sense to restrict ourselves with convex problems (1.1.1). Thus, it is assumed by default that $\mathcal{X}$ is a closed and bounded convex set, and $f : \mathcal{X} \longrightarrow \Re$ is convex. As discussed above, these assumptions would be sufficient to make (1.1.1) easy to solve, provided that $f(x)$ were given explicitly, but the latter is not what we assume in stochastic programming. What we usually (and everywhere below) do assume in stochastic programming is that:

(i) The function $F(x, \xi)$ is given explicitly, so that we can compute efficiently its value (and perhaps the derivatives in x) at every given pair $(x, \xi) \in \mathcal{X} \times \Xi$.

(ii) We have access to a mechanism which is capable of sampling from the distribution $\mathcal{P}$, that is, we can generate a sample $\xi^1, \xi^2, \ldots$ of independent realizations of ξ.

When applied to two-stage stochastic programming with recourse these assumptions require that the recourse is relatively complete. If it were not the case and for some $x \in \mathcal{X}$ and $\xi \in \Xi$ the second-stage problem is infeasible, we can formally set the value $F(x, \xi)$ of the second-stage problem to be $+\infty$. In order to avoid such infinite penalizations one can introduce a finite penalty for infeasibility. Although, in some

cases, this can reasonably solve the problem, in other situations the infeasibility may result in a catastrophic event. In that case the penalty could be enormous – in a sense, in such situation "nothing works".

It is NP-hard even to check whether a given first-stage decision $x \in \mathcal{X}$ leads to feasible, with probability 1, second-stage problem, and even in the case when the second-stage problem looks as simple as

$$\begin{array}{ll} \underset{y}{\text{minimize}} & q^T y \\ \text{subject to} & Tx + Wy \geq h, \end{array} \tag{1.2.6}$$

with only the second-stage right hand side vector $h = h(\xi)$ being random (see [83] for the explaining example).

Thus, if a two-stage (linear) problem has no relatively complete recourse (which in many applications is a rule rather than an exception), it is, in general, NP-hard just to find a feasible first-stage solution x (one which results in finite $f(x)$), not speaking about minimizing over these x's. As was mentioned above, the standard way to avoid, to some extent, this difficulty is to construct a penalized problem. For example, we can replace the second stage problem (1.2.2) with the penalized version:

$$\begin{array}{ll} \underset{y \geq 0, z \geq 0}{\text{minimize}} & q^T y + Rz \\ \text{subject to} & Tx + Wy \geq h - ze, \end{array} \tag{1.2.7}$$

where e is vector of ones, and the parameter $R >> 1$ plays the role of the penalty coefficient. With this particular penalization, the second stage problem becomes always feasible. At the same time, one can hope that with large enough penalty coefficient r, the first-stage optimal solution will lead to "nearly always nearly feasible" second-stage problems, provided that the original problem is feasible. Unfortunately, in the situation where one cannot tolerate arising, with probability bigger than α, a second-stage infeasibility z bigger than τ (here α and τ are given thresholds), the penalty parameter R should be of order of $(\alpha\tau)^{-1}$ (see [83]).

1.3 | Chance Constraints

A more natural way to handle two-stage stochastic problems without complete recourse is to impose so called chance constraints. The meaning of them is to require that a probability of insolvability of the second-stage problem is at most $\epsilon << 1$ instead of being 0. The reasoning behind this idea is twofold: first, from the practical viewpoint, "highly unlikely" events are not considered "too dangerous": why should we bother about a marginal chance, like 10^{-6}, for the second stage to be infeasible, given that the level of various inaccuracies and errors in our model, especially when

it comes to its probabilistic data, can be by orders of magnitude larger than 10^{-6}? Second, while it might be very difficult (or borderline impossible) to check whether a given first-stage solution results in a feasible (w.p. 1) second-stage problem, it seems to be possible to check whether this probability is at least $1-\epsilon$ by applying Monte Carlo simulation. Also, note that the chance constraints arise naturally not only in the context of two-stage problems without complete recourse, but in a much more general situations of solving a constrained optimization problem with the problem data affected by stochastic uncertainty. Therefore, it is reasonable to ask a question how could one process numerically a chance constraint:

$$\phi(x) = \mathcal{P}\{g(x,\xi) \leq 0\} \geq 1-\epsilon, \tag{1.3.1}$$

where x is the decision vector, ξ is the random disturbance with, say, a known distribution, and $\epsilon << 1$ is a given tolerance.

The idea of chance constraints originates from Charnes and Cooper [23] and is one of the oldest concepts in Operations Research. Unfortunately, more than a half of a century later, this concept still cannot be treated as practical. The first reason being that it is usually extremely difficult to verify exactly whether this constraint is satisfied at a given point. This problem is difficult already in the case of a single linear constraint $g(x,\xi) = (a+\xi)^T x$ with perturbations (uncertainty in the problem data) ξ uniformly distributed in a box [83]. Another serious problem is that usually constraint (1.3.1), even with very simple, say bi-affine in x and in ξ, function $g(x,\xi)$ and a simple-looking distribution of ξ (like uniform in a box) defines a nonconvex feasible set in the space of decision variables, which makes the subsequent optimization over this set of even pretty simple – just linear – objectives very problematic.

There is a generic case when the feasible set given by a chance constraint is convex. This is the case when the constraint can be represented in the form $(x,\xi) \in C$, where C is a closed and convex set, and the distribution $\mathcal{P}$ of the random vector $\xi \in \Re^{n_\xi}$ is logarithmically quasi-concave, meaning that

$$\mathcal{P}(\lambda A + (1-\lambda)B) \geq \max[\mathcal{P}(A), \mathcal{P}(B)]$$

for all closed and convex sets $A, B \subset \Re^{n_\xi}$ (cf., Prekopa [92]). Examples of this case include uniform distributions on closed and bounded convex domains, normal distribution and every distribution on $\Re^{n_\xi}$ with density $p(\xi)$ with respect to the Lebesgue measure such that the function $p^{-1/n_\xi}(\xi)$ is convex. The related result (due to Prekopa [92]) is that in the situation in question, the set $\{x : \mathcal{P}(\{\xi : (x,\xi) \in C) \geq \alpha\}$ is closed and convex for every α.

Aside from few special cases, the chance constraint (1.3.1) "as it is" appears to be "too difficult" for efficient numerical processing. What we aim to do is to replace

it with its "tractable approximation". For the time being, there exist two approaches to building such an approximation: "deterministic" and "scenario".

With the deterministic approach, one replaces (1.3.1) with a properly chosen deterministic constraint

$$\psi_\epsilon(x) \leq 0, \tag{1.3.2}$$

which is a "safe computationally tractable" (see [82]) approximation of (1.3.1), with the latter notion defined as follows:

1. "Safety" means that the validity of (1.3.2) is a sufficient condition for the validity of (1.3.1).
2. "Tractability" means that (1.3.2) is an explicitly given convex constraint.

To give an example, consider a randomly perturbed linear constraint, that is, suppose that

$$g(x, \xi) = (a + M\xi)^T x,$$

where the deterministic vector a is the "nominal data", M is a given deterministic matrix of appropriate dimension and $\xi = (\xi_1, \ldots, \xi_{n_\xi})$ is a tuple of n_ξ independent scalar random variables with zero mean and "of order of 1", i.e.

$$\mathbb{E}[\exp(\xi_i^2)] \leq \exp\{1\}, \quad i = 1, \ldots, n_\xi,$$

e.g., ξ_i can have a distribution supported on the interval $[-1, 1]$, or ξ_i can have normal distribution $\mathcal{N}(0, 2^{-1/2}), i = 1, \ldots, n_\xi$. In such a case, using standard results on probabilities of large deviations for sums of "light tail" independent random variables with zero means, one can verify that when $\epsilon \in (0, 1)$ and $\Omega(\epsilon) = O(1)\sqrt{\log(1/\epsilon)}$ with properly chosen absolute constant $O(1)$, then the validity of the convex constraint

$$a^T x + \Omega(\epsilon)\sqrt{x^T M M^T x} \leq 0 \tag{1.3.3}$$

is a sufficient condition for the validity of (1.3.1) (see [83]).

The rather straightforward result we have just described is very attractive. First, it does not assume a detailed knowledge of the distribution of ξ. Second, the approximation, although being more complicated than a linear constraint we start with, still is pretty simple (it is called a second-order cone constraint [17]). Modern convex optimization techniques can process routinely to high accuracy problems with thousands of decision variables and thousands of constraints of the form (1.3.3). Third, the approximation is "not too conservative", as the "safety" parameter $\Omega(\epsilon)$ grows pretty slowly as $\epsilon \longrightarrow 0$.

In contrast to this "highly specialized and heavily restricted" approach we have just considered, the scenario-based approach is completely universal. All we do is generate a sample $\xi^1, \ldots, \xi^N$ of N "scenarios" – independent realizations of the

random disturbance ξ and approximate the chance constraint (1.3.1) by the random system of inequalities

$$g(x, \xi^j) \leq 0, \quad j = 1, \ldots, N. \tag{1.3.4}$$

Extremely nice features of this approach are its generality and computational tractability – whenever $g(x, \xi)$ is convex in x and efficiently computable, (1.3.4) becomes a system of explicitly given convex constraints and as such can be efficiently processed numerically, provided that the number of scenarios N is not prohibitively large. The question, of course, is how large should be the sample in order to ensure, with reliability close to 1, that every feasible solution to (1.3.4) satisfies the chance constraint (1.3.1). This will be answered in much more detail in Chapter 4, where we describe a novel method of handling the scenario-based approach.

1.4 | Decomposition Methods

In this section we give a brief review of some of the decomposition methods used in stochastic programming, which is based on an computational study [117]. We will mainly focus on solving the "deterministic reformulation" of the problem (1.2.3), with a finite and fixed number of scenarios K. Other approaches, such as the stochastic decomposition by Higle and Sen [46], will be omitted for the sake of brevity. The deterministic equivalent problem can be solved as a linear programming problem by one of the variants of the simplex method or by a interior-point method (see [85]). However, the problem (1.2.3) has a particular structure – for each scenario, a subproblem is included that describes the second stage decision associated with the corresponding scenario realization. The subproblems are linked by the first stage decision variables, resulting in a so called "L-shaped" structure. The authors of [25] noticed that the dual of (1.2.3) fits the structural requirements for the Dantzig-Wolfe decomposition [26].

Van Slyke and Wets [108] developed a cutting-plane algorithm for the first stage problem (1.2.1) – their so called "L-shaped method" builds respective cutting plane models of the feasible domain (feasibility cuts) and of the expected recourse (optimality cuts). In its original form, the L-shaped method works on the aggregated problem (for more detail on the aggregation/disaggregation, cf. Section 2). A multicut version that works on the disaggregated problem was proposed by Birge and Louveaux [13]. There is a close connection between decomposition and cutting-plane approaches. The following approaches yield methods that are in principle identical:

- Cutting-plane method for either the disaggregated problem or the aggregated problem.

- Dantzig–Wolfe decomposition [26] applied to the dual of the deterministic equivalent problem (1.2.3).
- Benders decomposition [11] applied to the deterministic equivalent problem (1.2.3).

Cutting-plane approaches have the advantage that they provide a nice visual illustration of how the cuts are generated. A classical overview of decomposition methods can be found in [98].

The difference between the aggregated and the disaggregated problem formulations may result in a substantial difference in the efficiency of the solution methods – in the disaggregated version, more information is stored in the master problem (more cuts are added), hence the number of master iterations is reduced, but each iteration takes longer time to compute. Birge and Louveaux [14] conclude that the multicut approach is in general more effective when the number of the scenarios is not significantly larger than the number of the constraints in the first-stage problem. This claim is based on the numerical results [13] and [37].

It was observed that successive iterations of the cutting-plane methods did not generally produce an orderly progression of solutions – while the change in objective value from one iteration to the next may be very small, even zero, a wide difference may exist between corresponding values of the first-stage variables (what was dubbed a "flat objective function" in [53]). This feature of zigzagging in cutting-plane methods is the consequence of using a linear approximation. Improved methods were developed that use quadratic approximation: proximal point method by Rockafellar [93], and bundle methods by Kiwiel [54] and Lemaréchal [63]. These methods construct a sequence of stability centers together with the sequence of the iterates. When computing the next iterate, getting away from the current stability center is penalized.

The Regularized Decomposition method of Ruszczyński [97] is a bundle-type method applied to the minimization of the sum of polyhedral convex functions over a convex polyhedron, hence this method fits the disaggregated problem. The Regularized Decomposition method emphasizes keeping the master problem as small as possible (which is achieved by an effective constraint reduction strategy). A quite recent discussion of the Regularized Decomposition method can be found in [98].

A more recent development in convex programming is the level method of Lemaréchal et al. [64]. This is a special bundle-type method that uses level sets of the model functions for regularization. Fábián [32] developed inexact versions of the level method and the constrained level method. The inexact methods use approximate data to construct models of the objective and constraint functions. At the start of the procedure, a rough approximation is used, and the accuracy is gradually increased as the optimum is approached.

The box-constrained trust-region method of Linderoth and Wright [67] solves the disaggregated problem, and uses a special trust-region approach. Trust-region methods construct a sequence of stability centers together with the sequence of the iterates. Trust regions are constructed around the stability centers, and the next iterate is selected from the current trust region. The constructed trust regions are box-shaped, hence the resulting master problems remain linear. The size of the trust region is continually adapted depending on the quality of the current solution.

Different approaches, such as the Progressive hedging method [95], are a variant of the operator splitting method [27]. A more thorough description of these (and some more) methods can be found in the chapter on decomposition methods written by Ruszczyński in [98].

CHAPTER 2

Warm-Start Cuts for Generalized Benders Decomposition

This chapter is based on the original published article [57]. The only changes were notation/formatting ones to blend in with the rest of the text and a correction of a handful of misprints. It describes a decomposition algorithm suitable for two-stage convex stochastic problems called the Generalized Benders Decomposition and presents a a new reformulation that incorporates a lower bound cut that serves as a warm-start, decreasing the overall computation time.

2.1 | Introduction

In stochastic programming, we usually have to deal with problems that are large-scale but have a special structure [14]. Proper utilization of this special structure is the key part in the construction of any practically usable algorithm. One of the most widely used algorithms for two-stage stochastic linear programs is the L-shaped method developed by Van Slyke and Wets [108]. This method is based on (or, as the authors of the method wrote in the original paper: "is essentially the same as") the algorithm developed by Benders in [11] known as the Benders Decomposition. Over the years, numerous extensions for the L-shaped method have been proposed. A summary of the ones that are currently used can be found in [112] and [117].

A further generalization of the Benders decomposition for nonlinear convex problems ([6], [17]) was proposed by Geoffrion in [38] and was named the Generalized Benders Decomposition (GBD). The method found its main use as a solution technique for mixed-integer nonlinear problems, described in [33] and [34].

In this paper, we describe a formulation of the GBD that suits the particular structure of two-stage stochastic programming problems. After that, we introduce a reformulation that enables us to add a lower bound cut, which acts as a "warm-start" for the algorithm. As the lower bound cut, we decided to use the one that we can compute with the least effort. As there have been several lower bounds

proposed for stochastic programs (for example in [14],[72] and [73]) the question of the appropriate one for our problem will be left open for future research.

2.2 | Main Ideas

In this section, we give a brief insight into the GBD, as it is not our intention to devote several pages to its thorough description. An interested reader can find an in-depth analysis of the method in the original paper [38] and in the works of Floudas in [33] and [34].

The problems GBD aims to solve are of the form:

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) \\ \text{subject to} & G(x,y) \leq 0, \quad x \in X, y \in Y, \end{array} \tag{2.2.1}$$

where $x \in X \subseteq \Re^{n_1}$, $y \in Y \subseteq \Re^{n_2}$, $f : \Re^{n_1} \times \Re^{n_2} \longrightarrow \Re$ is a real-valued objective function and $G : \Re^{n_1} \times \Re^{n_2} \longrightarrow \Re^m$ is an m-vector of constraint functions. The variable x is called a complicating variable in the sense that (2.2.1) is a much easier optimization problem in y when x is temporarily held fixed. The following conditions are required:

C1: Y is a nonempty, convex set and the functions f and G are convex for each fixed $x \in X$.

C2: The set

$$Z_x = \{z \in \Re^m : G(x,y) \leq z \text{ for some } y \in Y\}, \tag{2.2.2}$$

is closed for each fixed $x \in X$.

C3: For each fixed $x \in X \cap V$, where

$$V = \{x : G(x,y) \leq 0, \text{ for some } y \in Y\}, \tag{2.2.3}$$

one of the following conditions holds:

(i) the problem (2.2.1) has a finite solution and has an optimal multiplier vector for the inequalities.

(ii) the problem (2.2.1) is unbounded, that is, its objective function value goes to $-\infty$.

This covers quite a wide range of problems [33]. The particular situation we are interested in is when f and G are linearly separable in x and y, i.e.

$$\begin{array}{rcl} f(x,y) & = & f_1(x) + f_2(y), \\ G(x,y) & = & G_1(x) + G_2(y). \end{array} \tag{2.2.4}$$

The basic idea in GBD is the generation, at each iteration, of an upper bound and a lower bound on the optimal objective function value of (2.2.1). The upper bound

results from a subproblem, while the lower bound results from a master problem. The subproblem corresponds to the problem (2.2.1) with fixed x-variable (i.e., it is in the y-space only), and its solution provides information about the upper bound and the Lagrange multipliers ([6], [17]) associated with the inequality constraints. The master problem is derived via nonlinear duality theory, makes use of the Lagrange multipliers obtained in the subproblem, and its solution provides information about the lower bound, as well as the next set of fixed x-variable to be used subsequently in the subproblem [33].

2.3 | GBD for Two-Stage Stochastic Programming Problems

In stochastic programming linear separability of the objective function and constraints is a very common property. Especially the two-stage stochastic programming problems can be often linearly separated into the functions concerning only the first-stage and the second-stage decision variables – this is the raison d'etre of the following passages, and it is why we believe that the GBD (in its slightly modified form) is a well-suited algorithm for these kinds of problems.

2.3.1 | Problem Formulation

Let us consider the following problem:

$$\begin{array}{ll} \underset{x,y_1,\ldots,y_K}{\text{minimize}} & f_1(x) + \sum\limits_{k=1}^{K} p(\xi_k) f_2(y_k, \xi_k) \\ \text{subject to} & G_{11}(x) \leq 0, \\ & G_{21}(\xi_k)x + G_{22}(y_k, \xi_k) \leq 0, \xi_k \in \Xi, \end{array} \tag{2.3.1}$$

where $f_1 : \Re^{n_1} \longrightarrow \Re$ is a convex function, all m_1 constraint functions $G_{11} : \Re^{n_1} \longrightarrow \Re^{m_1}$ are convex, and for all $\xi_k \in \Xi$ with $|\Xi| = K$ finite, $G_{21}(\xi_k)$ is a $m_2 \times n_1$ matrix, $f_2(\cdot, \xi_k) : \Re^{n_2} \longrightarrow \Re$ is convex, all m_2 constraint functions $G_{22}(\cdot, \xi_k) : \Re^{n_2} \longrightarrow \Re^{m_2}$ are convex, $P(\xi = \xi_k) \equiv p(\xi_k) > 0$, $\sum\limits_{k=1}^{K} p(\xi_k) = 1$.

The master problem corresponding to (2.3.1) has the following form:

$$\begin{array}{lll} \underset{x,\theta}{\text{minimize}} & f_1(x) + \theta & \\ \text{subject to} & G_{11}(x) \leq 0, & \\ & D_i x \leq d_i, & i = 1, \ldots, p, \\ & E_j x - \theta \leq e_j, & j = 1, \ldots, r, \end{array} \tag{2.3.2}$$

where $\theta \in \Re$ serves as the lower bound on the second stage objective value. The meaning of matrices D, E and vectors d, e will be fully discussed in the actual solution procedure. These matrices and vectors correspond to the feasibility and optimality cuts derived from the solutions of the subproblem.

Because of the structure of the two-stage stochastic programming problems, the subproblem separates into K independent subproblems (one for each scenario) in the form:

$$\begin{array}{ll} \underset{y_k}{\text{minimize}} & f_2(y_k, \xi_k) \\ \text{subject to} & G_{21}(\xi_k)x + G_{22}(y_k, \xi_k) \leq 0. \end{array} \tag{2.3.3}$$

Remark 2.3.1 *Regarding our notation – one could use k instead of ξ_k in the formulations above (and in the ones that will follow). The use of ξ is standard in the stochastic programming literature.*

2.3.2 | Solution Procedure

The following algorithm is an implementation of the GBD inspired by [38] and [33]. The single difference (apart from the notation) is that the separability of the subproblem into K independent subproblems is taken into account. At the start of the procedure, the matrices D, E and vectors d, e are empty (they store the successive cuts as the iterations progress).

To our best knowledge, this is the first implementation of the GBD for two-stage stochastic convex programming problems of the form (2.3.1).

Step 0. Set $c = 0, r = 0$, and $\epsilon > 0$.

Step 1. Solve (2.3.2) and obtain $(\bar{x}, \bar{\theta})$. The optimal objective value of (2.3.2) gives us a lower bound on optimal objective value of (2.3.1).

Step 2. For fixed $x = \bar{x}$ solve all K subproblems (2.3.3). One of two possibilities can happen.

Step 2A. For some k the subproblem (2.3.3) is infeasible. Solve the following problem:

$$\begin{array}{ll} \underset{y_k, v \geq 0}{\text{minimize}} & ||v||_1 \\ \text{subject to} & G_{21}(\xi_k)\bar{x} + G_{22}(y_k, \xi_k) \leq v, \end{array} \tag{2.3.4}$$

where $v \in \Re^{m_2}$ is a decision vector representing "slacks" in the constraints. Get $(\bar{y}_k, \bar{v})$ and from its dual obtain the optimal Lagrange multipliers λ. Set $c = c + 1$. Add a new row to the matrix D and vector d in (2.3.2):

$$D_c = \lambda^T G_{21}(\xi_k), d_c = \lambda^T(-G_{22}(\bar{y}_k, \xi_k)). \tag{2.3.5}$$

Return to Step 1.

Step 2B. All the subproblems have finite optimal values, we obtained $(\bar{y}_k, u_k)$, where u_k are optimal Lagrange multipliers. The evaluation of the objective of (2.3.1) at $(\bar{x}, \bar{y}_1, \ldots, \bar{y}_K)$ gives us an upper bound on its optimal value. Check for optimality: if

$$\bar{\theta} + \epsilon \geq \sum_{k=1}^{K} p(\xi_k) f_2(\bar{y}_k, \xi_k), \tag{2.3.6}$$

terminate, $(\bar{x}, \bar{y}_1, \ldots, \bar{y}_K)$ are ϵ-optimal [38]. Otherwise, set $r = r + 1$ and add a new row to the matrix E and vector e in (2.3.2):

$$\begin{aligned} E_r &= \sum_{k=1}^{K} p(\xi_k)(u_k^T G_{21}(\xi_k)), \\ e_r &= -\sum_{k=1}^{K} p(\xi_k)(f_2(\bar{y}_k, \xi_k) + u_k^T (G_{22}(\bar{y}_k, \xi_k)). \end{aligned} \tag{2.3.7}$$

Return to Step 1.

Remark 2.3.2 *In Step 1, before any optimality cut is added, $\bar{\theta}$ as well as the optimal objective value of (2.3.2) will be $-\infty$. For computational reasons it is advisable to include a lower bound on θ in the actual implementation of the algorithm.*

Remark 2.3.3 *If $X \subseteq V$ (i.e., in the case of complete or relatively complete recourse [14]), the Step 2A is never needed and for a given $\epsilon > 0$ the GBD terminates in a finite number of iterations. If however, $X \nsubseteq V$, then we may need to solve Step 2A infinitely many successive times. In such a case, to preserve finite ϵ-convergence, we can modify the procedure so as to finitely truncate any excessively long sequence of successive executions of Step 2A and go to Step 2B with $\hat{x}$ equal to the extrapolated limit point which is assumed to belong to $X \cap V$, see [33] or [34].*

2.4 | Reformulation with Bounding Cut

In this section, we introduce a novel reformulation of the master problem (2.3.2) that includes bounds obtained from problems, that can be thought of as predecessors of the two-stage stochastic programming problem (2.3.1). The definitions of these problems, as well as their subsequent relations, are based on [71].

2.4.1 | Bounds

Let us define

$$\begin{aligned} \underset{x_k, y_k}{\text{minimize}} \quad & f_1(x_k) + f_2(y_k, \xi_k) \\ \text{subject to} \quad & G_{11}(x_k) \leq 0, \\ & G_{21} x_k + G_{22}(y_k, \xi_k) \leq 0, \end{aligned} \tag{2.4.1}$$

as the optimization problem for one particular realization $\xi_k \in \Xi$ and denote its optimal objective function value as $z(\xi_k)$. The wait-and-see solution is the solution without nonanticipativity constraints (i.e. all scenarios are treated and optimized separately). We will denote the average of the optimal objective values of (2.4.1) (when treated separately) as:

$$\text{WS} = \sum_{k=1}^{K} p(\xi_k) z(\xi_k). \tag{2.4.2}$$

Now we may compare this wait-and-see solution to the solution of (2.3.1). We will denote the optimal objective value of (2.3.1) as RP (the recourse problem [14]). The following inequality holds for any stochastic program:

$$\text{WS} \leq \text{RP}. \tag{2.4.3}$$

From this, we can see that WS creates a valid lower bound on the harder problem we are aiming to solve. The idea behind the reformulation is to include such a valid lower bound to the algorithmic procedure to "jumpstart" it and by doing so save on iterations, and, as a result, save on the overall computational effort and time.

For practical purposes, many people would believe that finding the wait-and-see solution is still too much work. A natural temptation is to solve a much simpler problem: the one obtained by replacing all random variables by their expected values. This is called the expected value problem, which is simply

$$\text{EV} = z(\bar{\xi}), \tag{2.4.4}$$

where $\bar{\xi} = \sum_{k=1}^{K} p(\xi_k)\xi_k$.

2.4.2 | Reformulation

Although WS is a valid bound, the computational effort for its enumeration is much higher compared to the effort to compute EV (if $|\Xi| = K$, then computing EV is K times faster). However, EV does not necessarily have to play the role of a lower bound on RP; there are instances, where RP $\leq$ EV. For the purpose of deriving the reformulation, we will, for now, suppose that EV is, in fact, a valid lower bound on RP. The discussion on what is going to occur when it is not will follow shortly after. Suppose

$$\text{EV} \leq \text{RP}, \tag{2.4.5}$$

holds, then

$$f_1(x) + \sum_{k=1}^{K} p(\xi_k) f_2(y_k, \xi_k) \geq \text{EV}, \tag{2.4.6}$$

holds for the optimum of (2.3.1). This inequality cannot be added directly to (2.3.1) since it would cease to be a convex program. The reformulation we propose does not directly alter (2.3.1) but is instead aimed at the master problem (2.3.2). A new variable z is introduced to bound the first-stage objective from above (by minimizing this variable we effectively minimize the first-stage objective itself)

$$f_1(x) \leq z, \tag{2.4.7}$$

which is a convexity preserving inequality. Furthermore, this new variable z added to the variable representing the second stage θ form a lower bound on the overall objective function. Finally, the bound

$$z + \theta \geq \text{EV}, \tag{2.4.8}$$

since it is affine, can be added to (2.3.2), and the reformulation of the problem is

$$\begin{aligned} \underset{z,x,\theta}{\text{minimize}} \quad & z + \theta \\ \text{subject to} \quad & f_1(x) \leq z, \\ & G_{11}(x) \leq 0, \\ & z + \theta \geq \text{EV}, \\ & D_i x \leq d_i, \qquad i = 1, \ldots, p, \\ & E_j x - \theta \leq e_j, \quad j = 1, \ldots, r. \end{aligned} \tag{2.4.9}$$

After this reformulation, the algorithm continues as usual, arriving at an ϵ-optimal solution in, preferably, a shorter time than its original counterpart (we will see the results of some numerical examples in later sections).

Now, let us address what happens if (2.4.5) does not hold. One of two possibilities can occur, namely, that optimal objective function value (as determined by the algorithm) will be equal to EV, or that the problem will be infeasible. The price we pay for mistakenly using the cuts (2.4.5) is, in both cases, one iteration of the algorithm – i.e. after one iteration we can assess, if our algorithm will arrive at the desired solution, and, either restart it without (2.4.5) (possibly including WS instead), or continue.

However, certain situations can happen when we restart the algorithm without (2.4.5) and get the same result again. This occurs if the original problem is infeasible (in which case we have some serious model or data issues) or if EV = RP, in that case we would have to run the entire algorithm only to arrive at the same objective function value (which is a bit unfortunate, but unavoidable).

Another important question is if the cut (2.4.5) is worth having an additional variable. The numerical examples we provide in the later sections should supply us with some, although not definitive, insight into this issue.

Lastly, the question whether or not it is better to use the guaranteed lower bound in WS is also present. As we mentioned earlier, WS is computationally much more expensive than EV. In the examples that will follow we did not carry any examination of the WS bound, nor of any other possible bound. This is one of the areas that require further future investigation.

The solution procedure can be summarized in the following steps:

Step 0. Solve the expected value problem to get EV (2.4.4). Set $p = 0, r = 0$, and $\epsilon > 0$. Solve (2.4.9) and obtain $(\bar{z}, \bar{x}, \bar{\theta})$. If $\bar{z} + \bar{\theta} = \text{EV}$, terminate (and use the original method without the EV cut, or use WS instead). Otherwise, go to Step 2.

Step 1. Solve (2.4.9) and obtain $(\bar{z}, \bar{x}, \bar{\theta})$.

Step 2., Step 2A., Step 2B. The same as in section 2.3.2.

2.5 | Bunching and Multicuts

Just as in the linear case with the L-shaped method, different implementations of the algorithm can be researched for improving its performance ([14],[112]). Two possible adjustments suitable for GBD – bunching and the multicut formulation will be discussed and brought into the numerical examination.

Bunching, as the name suggests, is a technique that instead of the full scenario decomposition uses "bunches" of scenarios and decomposes the original problem alongside these bunches. Having L bunches of scenarios and sets of indices $B_l \neq \emptyset, l = 1, \ldots, L$, such that $B_i \cap B_j = \emptyset$ for $i \neq j$ and $\bigcup_{l=1}^{L} B_l = \{1, \ldots, K\}$. The subproblems (2.3.3) for each bunch l have the form

$$\begin{array}{ll} \underset{y_k, k \in B_l}{\text{minimize}} & \sum\limits_{k \in B_l} p(\xi_k) f_2(y_k, \xi_k) \\ \text{subject to} & G_{21}(\xi_k)x + G_{22}(y_k, \xi_k) \leq 0, k \in B_l. \end{array} \tag{2.5.1}$$

The feasibility and optimality cuts in Step 2A. and Step 2B. of the algorithm are changed accordingly. The feasibility cut in Step 2A. becomes

$$D_p = \sum_{k \in B_l} \lambda_k^T G_{21}(\xi_k), \quad d_p = -\sum_{k \in B_l} \lambda_k^T G_{22}(\bar{y}_k, \xi_k), \tag{2.5.2}$$

and the optimality cut in Step 2B. becomes

$$\begin{aligned} E_r &= \sum_{l=1}^{L} \sum_{k \in B_l} u_k^T G_{21}(\xi_k), \\ e_r &= -\sum_{l=1}^{L} \sum_{k \in B_l} p(\xi_k)(f_2(\bar{y}_k, \xi_k) + u_k^T G_{22}(\bar{y}_k, \xi_k), \end{aligned} \tag{2.5.3}$$

where λ_k and u_k are the Lagrange multipliers corresponding to the inequalities from scenario $k \in B_l$.

In the linear case, bunching comes from the idea that several second-stage problems might have the same optimal basis [14]. In the convex case, the justification is a bit different. Our argumentation is purely in the realm of the actual computation – it is sometimes faster (due to a non-zero initialization time, etc.) to compute a larger instance containing several separable problems than to solve these problems separately. The examples will show, up to a certain point, exactly this kind of behavior.

The multicut formulation comprises of developing one cut for every second-stage problem (i.e. for every scenario) instead of the aggregated cut introduced in (2.3.2). It results in adding a separate θ_k for each scenario and as a consequence in a much greater number of cuts which more accurately describe the recourse function [14]. The master problem for multicut formulation has the following form (without the additional cut developed in the previous section)

$$\begin{array}{lll} \underset{x,\theta_1,\ldots,\theta_K}{\text{minimize}} & f_1(x) + \sum_{k=1}^{K} \theta_k & \\ \text{subject to} & G_{11}(x) \leq 0, & \\ & D_i x \leq d_i, & i = 1, \ldots, c, \\ & E_{j(k)} x - \theta_k \leq e_{j(k)}, & j(k) = 1, \ldots, r(k), \\ & k = 1, \ldots, K, & \end{array} \tag{2.5.4}$$

where $r(k)$ and $j(k)$ indices are related to the k-th subproblem, see the steps below. In this case, the feasibility cuts remain the same, but the remaining steps require the following changes:

Step 0. – Multicut Set $c = 0, r(k) = 0$, for $k = 1, \ldots, K$ and $\epsilon > 0$.

Step 1. – Multicut Solve (2.5.4) and obtain $(\bar{x}, \bar{\theta}_1, \ldots, , \bar{\theta}_K)$.

Step 2. – Multicut For fixed $x = \bar{x}$ solve all K subproblems (2.3.3). One of two possibilities can happen.

Step 2A. – Multicut As before.

Step 2B. – Multicut All the subproblems have finite optimal values, we obtained $(\bar{y}_k, u_k)$, where u_k are optimal Lagrange multipliers. For $k = 1, \ldots, K$ if

$$\bar{\theta}_k + \epsilon \leq p(\xi_k) f_2(\bar{y}_k, \xi_k), \tag{2.5.5}$$

set $r(k) = r(k) + 1$ and add a new row to the matrix E and vector e in (2.5.4):

$$\begin{aligned} E_{r(k)} &= p(\xi_k)(u_k^T G_{21}(\xi_k)), \\ e_{r(k)} &= -p(\xi_k)(f_2(\bar{y}_k, \xi_k) + u_k^T G_{22}(\bar{y}_k, \xi_k)). \end{aligned} \tag{2.5.6}$$

If (2.5.5) does not hold for any k, terminate. Otherwise, return to Step 1.

Even though this formulation provides a more accurate description of the recourse function, its usefulness in the convex case is highly ambiguous. The number

of variables in the master problem is much larger than in the original algorithm and the number of constraints (cuts) added in each iteration is also much higher.

2.6 | Numerical Examples

To test the above mentioned theoretical concepts, we designed two convex two-stage problems. On these problems, we compare the performance of different variations of the GBD as well as a formulation without any decomposition (denoted as full recourse problems).

The implementation was done in `MATLAB` using its embedded `fmincon` solver and the state-of-the-art conic solvers `SeDuMi` and `SDPT3` [78] (which are a part of the `CVX` modeling system, see [39] and [40]). Although the examples are not derived from any applied problems, they provide a valid insight into the advantages and disadvantages of the presented methods.

2.6.1 | Example 1

The first example investigates the following problem

$$\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{y_1},\ldots,\boldsymbol{y_K}}{\text{minimize}} \quad & (x_1-4)^4+(x_2-3)^4+\sum_{k=1}^{K} p_k(q_{k,1}e^{y_{k,1}}+q_{k,2}y_{k,2}^4) \\
\text{subject to} \quad & x_2-\ln(x_1+1)-1\leq 0, \\
& x_2+x_1^3-8\leq 0, \\
& x_1,x_2\geq 0, \\
& x_1+h_{k,1}-y_{k,1}\leq 0, \quad k=1,\ldots,K, \\
& x_2+h_{k,2}-y_{k,2}\leq 0, \quad k=1,\ldots,K,
\end{aligned}$$

where the random parameters q and h are $q\sim|N(0,3)|$, $h\sim 0.7\cdot|N(0,1)|+0.5$. The scenarios are then constructed using the usual Monte Carlo sampling, the number of scenarios will vary to demonstrate the performance of the different approaches. The methods and solvers used for solving the problem were:

- vanilla (original) version of GBD (master and subproblems solved by `fmincon`);
- reformulation with the EV cut (master and subproblems solved by `fmincon`);
- bunching of several scenarios (master and subproblems solved by `fmincon`);
- bunching of several scenarios with the EV cut (master and subproblems solved by `fmincon`);
- full recourse problem (FRP) solved by `fmincon`;
- FRP solved by `SDPT3` (as a part of the `CVX` modeling system);
- FRP solved by `SeDuMi` (as a part of the `CVX` modeling system).

The required precision for all the methods was set to $\epsilon=10^{-5}$. The results are summarized in the tables that follow. The Time[s] value represents the computational

time it took the procedure to terminate, given the same level of accuracy for all methods. The number of scenarios in the first instance is $K = 60$. The first two tables show, how the computational time of the GBD is affected by introducing the EV cut:

Method	Vanilla	EV cut
Time[s]	12.26	9.05

Table 2.1: Computational time [s] for Vanilla version and EV cut, $K = 60$.

and by bunching with different sizes of the bunch:

Bunch size	2	3	5	10	12	15	20	30
Time[s] – Without EV cut	7.04	5.08	3.61	2.76	2.64	2.65	2.75	3.22
Time[s] – With EV cut	5.19	3.79	2.75	2.07	2.02	1.99	2.13	2.44

Table 2.2: Computational time [s] for bunching with different sizes of the bunch, $K = 60$.

An identical structure is utilized in the case of $K = 240$ scenarios:

Method	Vanilla	EV cut
Time[s]	43.84	35.42

Bunch size	2	3	5	6	8	10	12	15
Time[s] – Without EV cut	24.8	17.7	12.4	11.4	9.9	9.2	8.8	8.7
Time[s] – With EV cut	19.9	14.3	10.2	9.2	8.0	7.4	7.2	7.0
Bunch size	16	20	24	30	40	60	80	120
Time[s] – Without EV cut	8.7	8.9	9.4	10.5	12.2	16.8	21.5	25.3
Time[s] – With EV cut	7.0	7.2	7.7	9.5	9.8	13.6	17.3	20.2

Table 2.3: Computational time [s] for Vanilla, EV cut and bunching, $K = 240$.

From these results, we see that the EV cut, as well as efficient bunching, can have a strong effect on the overall computation time. The experiments suggest that there exists an "optimal" bunch size that is independent of the number of scenarios. For this particular problem, it seemed that a bunch size between 12 and 16 was the one. For the subsequent computations, the bunch size 15 was chosen.

In the following table, we compare the computation times for a growing number of scenarios using the methods and solvers mentioned above:

Number of scenarios	60	240	1,500	2,400	4,800	6,000
Vanilla	12.3	43.8	258.6	410.1	–	–
EV cut	9.1	35.4	208.9	332.7	–	–
Bunch 15	2.7	8.7	52.5	78.0	154.75	194.9
Bunch 15 with EV	1.9	7.0	40.1	62.6	124.6	156.4
FRP – `SDPT3`	6.1	22.3	139.9	241.7	–	–
FRP – `SeDuMi`	1.4	6.2	32.8	65.2	170.3	242.5
FRP – `fmincon`	0.5	12.2	3,000*	3,000*	–	–

Table 2.4: Computational time [s] for different methods, increasing number of scenarios.

The asterisk(*) denotes that the algorithm did not arrive at the desired precision (i.e. even after 3,000s the `fmincon` did not arrive sufficiently near the optimum). The dash(–) means that we did not pursue the analysis in this direction since we anticipated results incomparable with the more efficient methods.

These results show that for big enough problems, the efficient implementation of GBD, even with simpler solvers, can outperform the state-of-the-art solvers. For smaller instances, however, these solvers are more efficient (as will be presented in the results of the second example).

2.6.2 | Example 2

The second example included in our investigation, compared to the first one, adds some more first and second-stage variables and non-differentiable functions. These are the reason why, in the implementation, the more efficient solvers had to be utilized for the solution of the master problem (`fmincon` performed very poorly in this case). The problem in question is the following

$$\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{y_1},\ldots,\boldsymbol{y_k}}{\text{minimize}} \quad & (x_1-4)^4+(x_2-3)^4+(x_3-2.5)^2+3|x_1+x_4+4x_5-15| \\
& +\sum_{k=1}^{K} p_k(q_{k,1}e^{y_{k,1}}+q_{k,2}y_{k,2}^4+q_{k,3}(y_{k,3}-2)^2 \\
& +q_{k,4}|y_{k,4}+q_{k,5}y_{k,5}|) \\
\text{subject to} \quad & x_2-\ln(x_1+1)-\sqrt{x_3}+x_4+x_5^2-10\leq 0, \\
& x_2+x_1^3+x_3^3-10\leq 0, \\
& -x_4-\sqrt{x_5}+5<=0, \\
& x_i\geq 0, i=1,\ldots,5 \\
& T_kx+W_ky_k\leq h_k, \quad k=1,\ldots,K.
\end{aligned}$$

The random parameters q, h, W and T are (using some `MATLAB` syntax), $q\sim|N(0,1)|$, $h\sim -0.7\cdot|N(0,1)|-1$, $W=-\mathrm{I}_5$, $M=$ 5x5 matrix with 1 to 3 zeros in each column, the rest are 1, $T=\mathrm{abs}(0.2.*\mathrm{randn}(5)).*M$. The scenarios are, again, constructed

using the Monte Carlo sampling. As before, we used several methods and solvers for solving the problem:

- vanilla version of GBD (master solved by `SeDuMi`, subproblems by `fmincon`);
- EV cut version (master solved by `SeDuMi`, subproblems by `fmincon`);
- bunching + EV cut (master solved by `SeDuMi`, subproblems by `by fmincon`);
- bunching + EV cut + multicut
  (master solved by `SeDuMi`, subproblems by `fmincon`);
- FRP solved by `SDPT3`;
- FRP solved by `SeDuMi`.

The required precision for all the methods was set to $\epsilon = 10^{-5}$. By computations similar to that of the first example, we found the appropriate bunching size to be 5. The comparison of the different methods for varying number of scenarios is summarized in the following table:

Number of scenarios	125	250	500	1,000	2,000	3,000	5,000	7,500
Vanilla	19	45	77	164	313	–	–	–
EV cut	15	34	61	123	320	–	–	–
Multicut	19	36	134	150	309	–	–	–
Bunch 5	12	32	44	84	170	255	452	650
Bunch 5 + EV	9	17	34	71	175	210	364	578
Bunch 5 + Multicut	11	20	32	74	250	452	–	–
Bunch 5 + Multicut + EV	9	15	34	99	256	463	–	–
FRP – `SDPT3`	13	30	64	130	334	–	–	–
FRP – `SeDuMi`	1	2	6	15	40	102	355	622

Table 2.5: Computational time [s] for different methods, increasing number of scenarios.

The results demonstrate the pros and cons of using the GBD algorithm. For smaller instances, it is much more efficient to use the appropriate state-of-the-art and free solver (`SeDuMi`) to attack the full recourse formulation. However, for larger problems, the bunching variation of the GBD was able to outperform all the rest. The multicut variation suffered from a growing size of the master problem and, in this setting, cannot be considered as an improvement (a similar behavior for linear problems was shown in [112]).

2.7 | Conclusion

In this paper, we introduced a novel utilization and reformulation of the traditional Generalized Benders Decomposition. To support the utility of our reformulation (as well as the utility of the GBD itself), we presented our computational experience.

From the result of the numerical examples, it is apparent that the GBD and our modifications definitely have a place as solid techniques for solving medium-sized convex two-stage stochastic problems and that especially the bunching ideas and modifications produce fruitful results.

It must be acknowledged that further investigation (i.e. a wider variety of numerical tests, preferably from applications) is needed to make the arguments more conclusive. Also, further research in terms of usable lower bound as the "warm-start" cuts is anticipated.

Acknowledgement

This work was supported by NETME CENTRE PLUS (LO1202) created with financial support from the Ministry of Education, Youth and Sports under National Sustainability Programme I, by the research project "Moderní matematické metody pro modelování problémů technických a přírodních věd" (FSI-S-17-4464), and by the research project "Vývoj a aplikace moderních dekompozičních metod v oblasti stochastického programování" (FSI-FV-17-04).

CHAPTER 3

Waste Transfer Station Planning by Stochastic Programming

This is a transcript of another original article [60]. The only changes were, again, only notation/formatting/misprint-correcting ones. It describes a waste-management application of the two-stage stochastic programming problem and utilizes the warm-start cut described in Chapter 2.

3.1 | Introduction

Since the situation in waste management is unknown due to the undecided support to the particular technology system and treatment from the government or the EU, the planning of future infrastructure is not secured from the investment point of view. The state-of-the-art in the field of location and network flow problem is extensive. The paper by [55] is worth mentioning, because they summarized the progress in the sustainability applications from the recent years. Another important result was published by [110], where the network was utilized for the organic and dry fractions of municipal waste through the p-graph approach. The authors of [104] analyzed the current state of the waste handling, which is an important input for the simulations of future development. However, the all the previous planning is performed globally and for all subsystems at the same time. Some papers deal with sequential development and construction as in [31]. The individual decisions are not robust enough to comprise the unknown future development (the problems were not handled as multi-stage as in [48]).

This paper proposes a novel approach in the planning of transport infrastructure for efficient treatment of residual waste which is in line with all the possible cases of future development of waste management system. The future uncertainty (legislative development and support for different systems) in the treatment grid design is projected through the processing cost for different facilities at various locations. The computational approach was designed to handle real-life tasks in reasonable

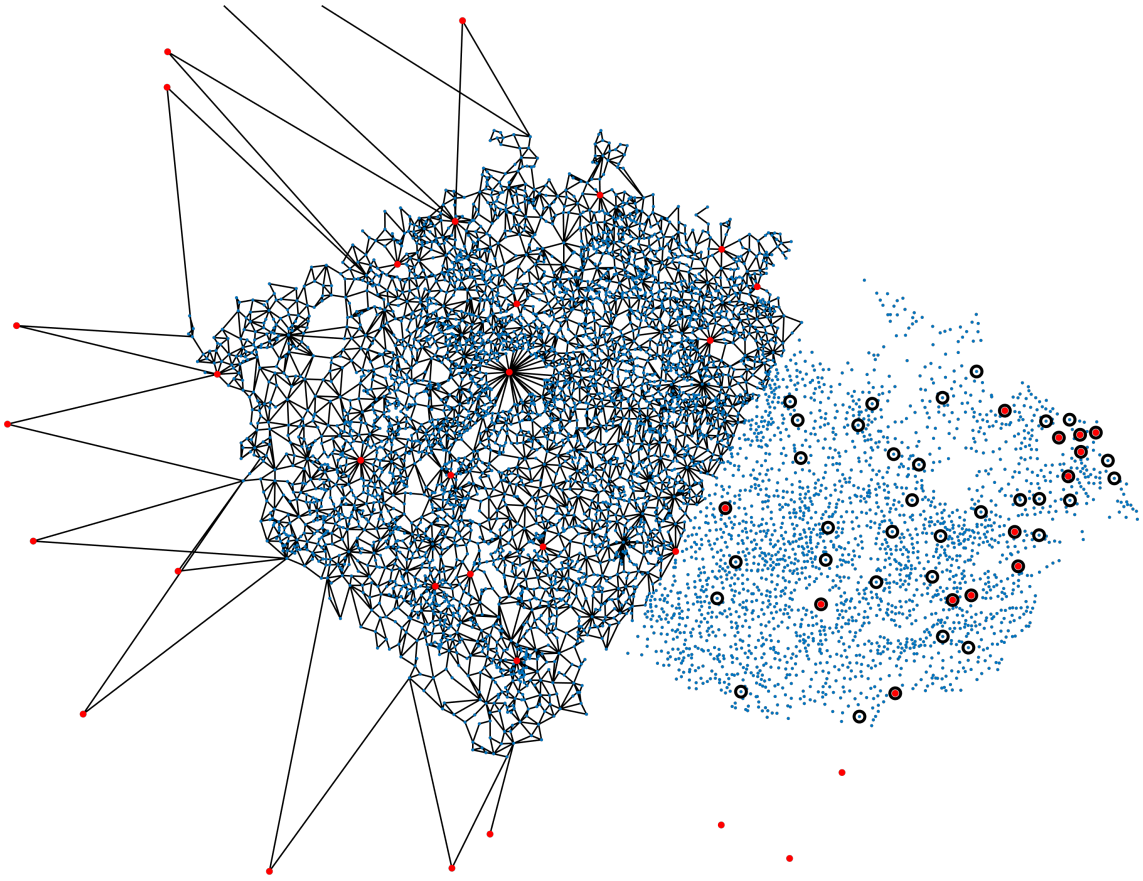


Fig. 3.1: A map showing the producers of waste (blue dots), the places processing waste (red dots). The road network (black lines) and the possible transfer stations (black rings) are shown on two separate parts.

time. In section 3, the case study is presented with the use of data from the Czech Republic.

3.2 | Problem Description

The problem consists of deciding where to construct the transfer stations, what should be their respective capacities and from which producer of waste to which waste-processing plant should the cargo be send, provided that some of the data are uncertain. This problem can be categorized as a two-stage stochastic facility location problem [14], where the so-called first-stage decision must be made prior to the realization of the uncertainty (this corresponds to the construction of the transfer stations and their capacities). The second-stage decision then depends on the realization of the uncertainty (in this case, all the other decisions about transport and processing are second-stage). The uncertainty is modeled using a large number of possible realizations called scenarios. The more scenarios are considered, then

generally speaking the better the model is, but the more difficult it is to solve.

Type	Symbol	Description
Sets	$s \in S$	Set of scenarios
	$j \in J$	Set of nodes (cities)
	$i \in I \subset J$	Set of possible transfer stations
	$t \in T$	Set of possible options for transfer station capacities
Parameters	A_1	First incidence matrix (Fig. 3.1)
	A_2	Second incidence matrix (from pre-processing)
	c_1	Transfer costs, without the transfer stations (on A_1)
	c_2	Transfer costs, using transfer stations (on A_2)
	p_s	Probability of a scenario s
	$e_{i,t}$	Cost of a construction of a transfer station at location i, with capacity option t
	$k_{i,t}$	Capacity of a transfer station at location i with capacity option t
	$f_{j,s}$	Cost of processing waste at node j, scenario s
	r_j	Production of waste at node j
	q_j	Waste processing capacity of node j
Variables	$d_{i,t}$	Decision on building the transfer station at location i, with capacity option t; binary, first-stage
	$x_{1,s}$	Flows on A_1 in scenario s; continuous, second-stage
	$x_{2,s}$	Flows on A_2 in scenario s; continuous, second-stage
	$y_{j,s}$	Amount of processed waste in node j, scenario s; continuous, second-stage

Table 3.1: The notation.

Possibly the most important data regarding this problem is the road network partly depicted in Fig 3.1 (and described by an incidence matrix A_1 in the mathematical model). This network had 24,770 arcs (roads) connecting the 6,258 nodes (waste producers and waste-processing plants). The second important piece of data are the locations of the waste producers, the waste-processing plants and the possible locations for transfer stations – some of these are depicted in Fig 3.1. In the problem there were 6,258 places producing waste, 44 waste-processing plants (where 15 correspond to foreign facilities – potential export of waste abroad) and 116 possible places for the transfer stations (these sets were not mutually exclusive).

To be able to differentiate between the transportation of waste that does or does not use the transfer stations, a separate road network was computed – for each possible transfer station was found the shortest path to each waste-processing

plant. In this pre-processing step, 5,075 shortest path optimization problems were solved, resulting in the additional network with 5,075 arcs (omitting the ones that started and ended at the same place). The transfer of waste when using the transfer stations is assumed 3 times cheaper than the regular one. Each of the possible transfer stations can be constructed with 6 different capacities (higher capacities have higher construction costs, but the unit cost decreases). Combining this with the 116 locations results in 696 binary first-stage decisions. The second-stage decisions are the flows on the arcs of the two networks and the amounts of waste processes at the plants, in total 29,889.

The uncertain parameters that are considered in the model are the costs for processing the waste at the 44 different plants, which correspond with the legislation development and local conditions (such as the demand for heat, etc.). The number of scenarios for this model was set to 1,000 and so the model has almost 30M variables. The notation that is used to develop the mathematical model is described in Table 3.1.

To simplify the notation, some subscripts were hidden, meaning that the appropriate parameters/variables were stacked to form a vector of a fitting size (and the associated equalities/inequalities are meant for each element in the vector). The mathematical model has the following form:

$$\text{minimize} \quad \sum_{i\in I, t\in T} e_{i,t}d_{i,t} + \sum_{s\in S} p_s(c_1^T x_{1,s} + c_2^T x_{2,s} + f_s^T y_s) \tag{3.2.1}$$

$$\text{subject to} \quad A_1 x_{1,s} + A_2 x_{2,s} + y_s = r, \quad \forall s \in S, \tag{3.2.2}$$

$$y_s \leq q, \quad \forall s \in S, \tag{3.2.3}$$

$$\sum_{\text{flows from } i\in I} x_{2,s} \leq \sum_{t\in T} k_{i,t}d_{i,t}, \quad \forall s \in S, \forall i \in I, \tag{3.2.4}$$

$$\sum_{t\in T} d_{i,t} \leq 1, \quad \forall i \in I, \tag{3.2.5}$$

$$x_{1,s}, x_{2,s}, y_s \geq 0, \quad \forall s \in S, \tag{3.2.6}$$

$$d_{i,t} \in \{0,1\}, \quad \forall i \in I, \forall t \in T. \tag{3.2.7}$$

The objective function given by (3.2.1) is the expected waste transportation and processing costs and the building cost for building the transfer plants. The constraint (3.2.2) is the conservation of waste – at each node and for each scenario, the amount produced must be equal to the amount transported (by one of the two possibilities) plus the amount processed. The constraint (3.2.3) is an upper bound on the amount of waste that can be processed at a given node. The constraint (3.2.4) guarantees that the amount transferred using the transfer station i is less than the installed capacity of that transfer station. The constraint (3.2.5) ensures that at most one of the possible capacities is installed at location i. The last two constraints (3.2.6) and (3.2.7) are the nonnegativity and integrality constraint, respectively. The only

constraints that do not depend on the scenarios are (3.2.5) and (3.2.7). The total number of constraints that depend on scenarios is 36,307, meaning that the model has over 36M constraints.

3.3 | Implementation and Results

3.3.1 | Algorithms and Software

The model was solved using the Benders decomposition scheme described in [59] enhanced by the warm-start cuts developed in [57]. It was programmed in the high-performance dynamic language `JULIA` [12] with the `JuMP` package for mathematical optimization [28]. In this scheme, the first stage problem was solved using the branch-and-cut method for mixed-integer problems, calling the `CPLEX 12.6.3` solver. The MIP gap parameter was set at 1.5%. The individual subproblems in the second stage were solved by the primal-dual simplex method, calling the `GUROBI 7.5` solver. This combination of solvers and algorithms achieved the best overall performance – this scheme reached the 1.5% optimality gap for the problem formulation with 1,000 scenarios within 24 h. These computations were carried out on an ordinary computer (3.2 GHz i5-4460 CPU, 16 GB RAM). Another suitable solution strategy could be a heuristic based on genetic algorithms as in [56] or differential evolution as in [109]. All of these strategies can utilize parallel computing to accelerate the execution.

3.3.2 | Summary of the Results

The results of the computation are best summarized in Fig 3.2 and Fig 3.3. Of the 116 possible locations, 71 were chosen as optimal places for the transfer stations. One scenario of optimal flows and the optimal places for the transfer stations is depicted in Fig 3.2 (the optimal places are the same for all scenarios, the flows are different).

The optimal expected cost was 260.14M EUR and the expected total distance traveled by all vehicles was 8.23M km, assuming that the regular flows are serviced by vehicles with capacity 10 t and the flows from transfer stations are serviced by vehicles with capacity 24 t (all fully loaded).

The histograms in Fig 3.3 represent the results for the 1,000 generated scenarios and show in detail the impact of building the transfer stations. The expected costs are 8% lower on average when building the transfer stations, the costs for transportation alone are 21% lower. The expected total distance traveled by all vehicles is reduced by 9% on average when building the transfer stations. However, this

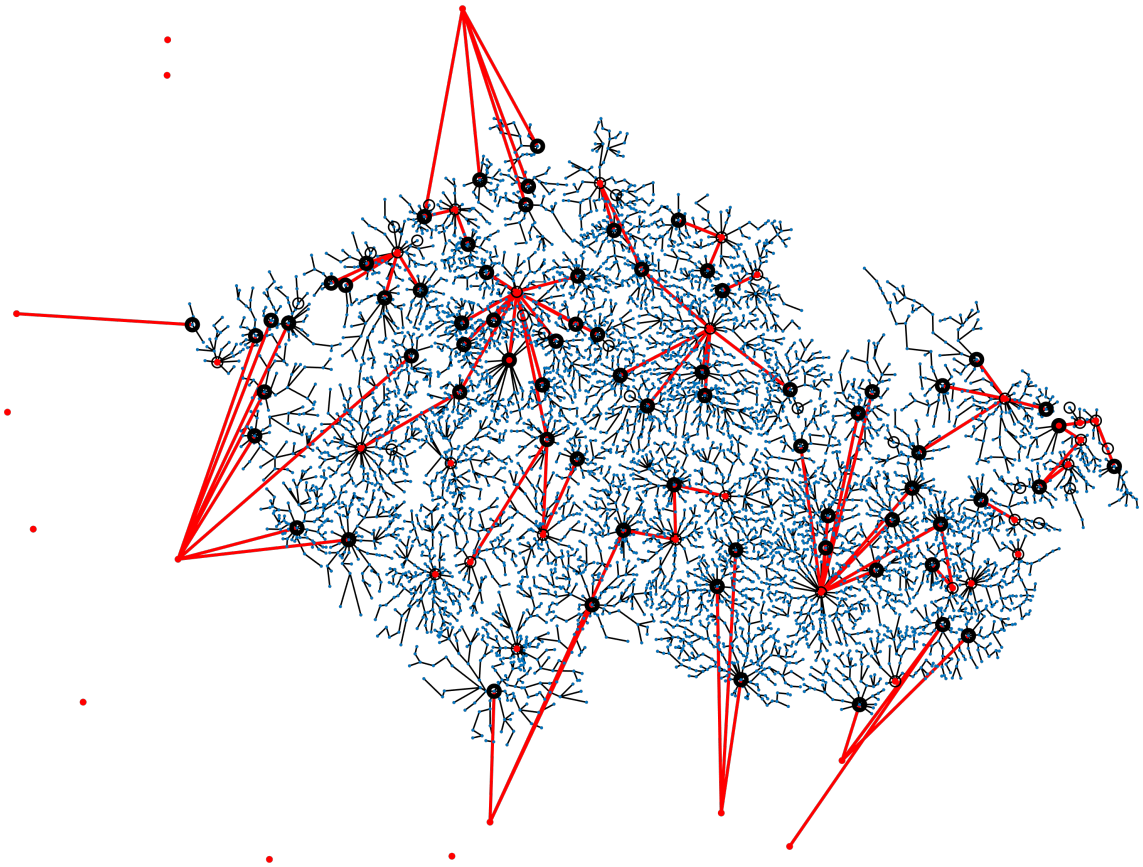


Fig. 3.2: A map showing the results for one of the scenarios – thick black rings correspond to the selected places for transfer stations, red lines are flows from these transfer stations, black lines are regular flows.

quantity has a much higher variance and, in some scenarios, is worse than the situation with no transfer stations. This inconvenience stems from the objective focusing only on costs – if some form of trade-off between costs and total distance was added to the objective function, the results would be more favourable towards lower total distance (at the price of increased costs). This might represent the situation when taking into account the environmental aspects is more important than the overall cost.

3.4 | Conclusion

In this paper, the mathematical model for grid design of transfer stations is proposed. The planning was modelled by a two-stage mixed-integer stochastic programming problem. The uncertainty is included in the cost of treatment, which corresponds to the possible future development of legislation and government support. An approach was tested through a case study on the current situation and possible legislation changes regarding waste management in the Czech Republic. It was scaled on the

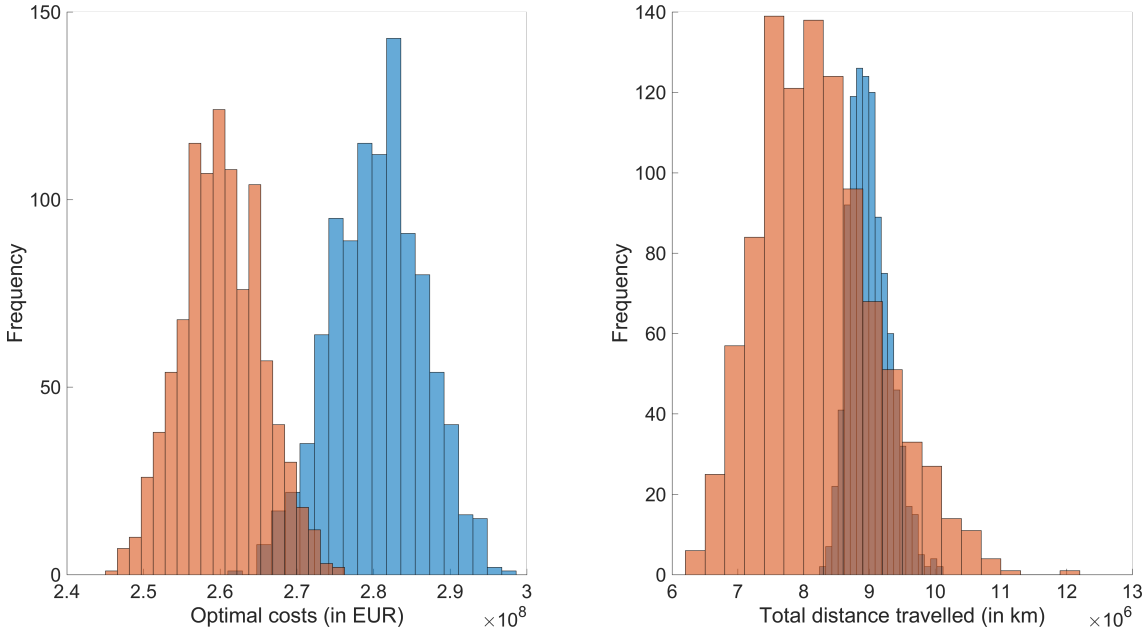


Fig. 3.3: Histograms of optimal cost and total distance travelled – the red one is using the transfer stations, the blue one is not.

micro-regional level where the network had 24,770 roads connecting the 6,258 waste producers and treatment plants. With these features, the robust transfer station grid design was proposed. The realization of these projects takes into consideration possible investments and decides also about the capacity of the facility.

The output is in the form of recommendation for possible investors, municipalities and/or stakeholders from the field of waste management. The optimal solution with the 1.5% gap was to design 71 sustainable projects, while the total expected cost was 260.14M EUR and the expected total distance travelled by all vehicles was 8.23M km. The possible extension for the proposed model would lead to consider the environmental aspect as the additional criterion or to calculate with the uncertain future waste production of the municipalities.

Acknowledgement

The authors gratefully acknowledge the financial support provided by Technology Agency of the Czech Republic within the research project No. TE02000236 “Waste-to-Energy (WtE) Competence Centre”, the funding from the MEYS under the National Sustainability Programme I (Project LO1202) and the standard specific research projects FSI-S-17-4526 and FSI-S-17-4464.

This page is intentionally left blank.

CHAPTER 4

Chance Constrained Problems

In this chapter, we describe a new algorithm aimed at handling the chance constrained problems described in Section 1.3. It uses much of the theory of probabilistic robust design developed by Calafiore, Campi and Garatti in [18], [19], [20],[21], and [22] – this theory is summarized in the first, introductory, part of the chapter (Section 4.1 and Section 4.2). In the remaining sections, the algorithm is described, examined and compared to other techniques for handling chance constrained problems (the ones presented in [1], [82], and [100]).

4.1 | Introduction

The introduction into the topic is derived (more or less directly) from [21] – with most of the used notation adapted from [21] as well. Let $\mathcal{X} \subseteq \Re^{n_x}$ be a convex and closed domain of optimization and consider a family of constraints $x \in \mathcal{X}_\xi$ parameterized in $\xi \in \Xi$. The uncertain parameter ξ describes different instances of an uncertain optimization scenario. We adopt a probabilistic description of uncertainty and suppose that the support Ξ for ξ is endowed with a σ-algebra $\mathcal{D}$ and that a probability measure $\mathcal{P}$ is defined over $\mathcal{D}$. The probability measure $\mathcal{P}$ describes the probability with which the uncertain parameter ξ takes value in Ξ. Then, a chance constrained optimization program is written as:

$$\begin{array}{rll} \mathrm{CCP}_\epsilon : & \underset{x \in \mathcal{X}}{\text{minimize}} & c^T x \\ & \text{subject to} & \mathcal{P}\{\xi : x \in \mathcal{X}_\xi\} \geq 1 - \epsilon. \end{array} \tag{4.1.1}$$

Here, we assume that the σ-algebra $\mathcal{D}$ is large enough, so that $\{\xi : x \in \mathcal{X}_\xi\} \in \mathcal{D}$, i.e. $\{\xi : x \in \mathcal{X}_\xi\}$ is a measurable set. Also, linearity of the objective function can be assumed without loss of generality, since any objective of the form

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad c(x),$$

where $c(x) : \mathcal{X} \to \Re$ is a convex function, can be re-written as

$$\underset{x \in \mathcal{X}, y \geq c(x)}{\text{minimize}} \quad y,$$

where y is a scalar variable.

In the CCP_ϵ (4.1.1), constraint violation is tolerated, but the violated constraint set must be no larger than ϵ. The parameter ϵ allows us to trade robustness (in terms of the probability of constraint violation) for performance (in terms of the optimal objective value): the optimal objective value J_ϵ^* of CCP_ϵ is a decreasing function of ϵ and provides a quantification of such a trade-off. Depending on the particular application (the range of applications is quite wide), ϵ can take different values and has not necessarily to be thought of as a "small" parameter.

Chance constrained programming has been around for more than half a century, at least since the work of Charnes, Cooper and Symonds in the fifties, see [23]. In [23], however, only individual chance constraints were considered. Joint probabilistic constraints, as in (4.1.1), were first considered by Miller and Wagner, [77], in an independent context, while a general theory is due to Prékopa, see [90], [91]. Prékopa was also the one to introduce the convexity theory based on logconcavity, which was a fundamental step toward solvability of a large class of chance constrained problems (see the appropriate section in Chapter 1). The books [92] and [98] provide an excellent and broad overview on logconcavity theory in stochastic programming, and related results. Yet another study about the convexity of chance constrained problems is [45], while convex approximations of chance constrained problems are considered in [9], [81], and [82] (some of the ideas of convex approximation of chance constriants are presented in Section 4.5). Stability of the solution under perturbation of the chance constrained problem is studied in [43] and [44]. Although chance constrained problems can be efficiently solved in some special cases (that were outlined above), it remains true that the feasible set of CCP_ϵ is in general non-convex in spite of the convexity of the sets $\mathcal{X}_\xi$. Therefore, an exact numerical solution of CCP_ϵ is, at least in general, extremely hard to find.

4.2 | Sample Counterpart

We can view the variable $x \in \mathcal{X} \subseteq \Re^{n_x}$ as the "design variable". The family of possible instances is parameterized by an "uncertainty vector" $\xi \in \Xi \subseteq \Re^{n_\xi}$. Then, the prototype optimization problem consists in minimizing a linear objective $c^T x$, subject to that x satisfies the constraints $g(x, \xi) \leq 0, \forall \xi \in \Xi$, where $g(x, \xi) : \mathcal{X} \times \Xi \to [-\infty, \infty]$ is a scalar-valued function that specifies the constraints. Note that considering scalar-valued constraint functions can be assumed without loss of generality, since multiple constraints $g_1(x, \xi) \leq 0, \ldots, g_m(x, \xi) \leq 0$ can be expressed by a single scalar-valued constraint by the position $g(x, \xi) = \max_{i=1,\ldots,m} g_i(x, \xi)$. Although convexity is preserved by this operation, other valuable properties, such as linearity or differentiability, are lost. In typical situations, Ξ has infinite cardinality, i.e., it contains an infinite number of possible instances for ξ.

Assumption 4.2.1 (Convexity)
For each $\xi \in \Xi$ the sets $\mathcal{X}_\xi$ are convex and closed.

Assumption 4.2.1 requires convexity only with respect to the design variable x, while generic nonlinear dependence with respect to ξ is allowed. Now we come to the point of distinguishing between two possible approaches (whose usage is, more than anything else, complementary):

a) *Worst-Case (or Robust) Design*: In worst-case design, one aims at enforcing the constraint $g(x, \xi) \leq 0$ for all possible values of the uncertainty $\xi \in \Xi$. However, a fundamental problem is encountered along this approach: Obtaining worst-case solutions has been proven to be computationally hard; explicit results on the NP-hardness of several worst-case design problems are for instance found in [16], and [80]. In addition, a second issue applies to a worst-case design: Seeking guarantees against the (possibly highly improbable) worst-case can introduce undesirable conservatism in the design, since all the design focus is on a special "ill" situation, which could be highly unrepresentative of the majority of admissible situations.

b) *Probabilistic Robust (or Chance constrained) Design*: In the probabilistic design framework, we assume that a probability measure $\mathcal{P}$ over the uncertainty set Ξ is given. Then, for a given probability level $\epsilon \in (0, 1)$, we look for a design variable x that minimizes $c^T x$ while satisfying all constraints but a small fraction of them whose probability is no larger than the prescribed level ϵ. It is possible to view this approach as a relaxation of the worst-case approach where one allows a risk level ϵ and looks for a design variable such that the performance specification is violated by at most a fraction of the plants in the uncertainty family.

Depending on the situation at hand, the measure $\mathcal{P}$ can have different interpretations. On one hand, it can be the actual probability with which the uncertainty parameter ξ takes on value in Ξ. On the other hand, $\mathcal{P}$ can simply describe the relative importance we assign to different uncertainty instances. We have the following definition:

Definition 4.2.2 (Probability of Violation)
Let $x \in \mathcal{X}$ be given. The probability of violation of x is defined as

$$\mathcal{V}(x) = \mathcal{P}\{\xi \in \Xi : g(x, \xi) > 0\}.$$

For example, if we assume a uniform probability density, then $\mathcal{V}(x)$ measures the "volume of bad" parameters ξ such that the constraint $g(x, \xi) \leq 0$ is violated. A solution x with small associated $\mathcal{V}(x)$ is feasible for most of the problem instances, i.e., it is approximately feasible for the worst-case problem. This concept of approximate feasibility has been introduced in the context of robust control in [5]. Any such solution is here named an "ϵ-level" solution:

Definition 4.2.3 (ϵ-Level Solution)
Let $\epsilon \in (0, 1)$. We say that $x \in \mathcal{X}$ is an ϵ-level robustly feasible (or, more simply, an ϵ-level) solution, if $\mathcal{V}(x) \leq \epsilon$.

Our ultimate goal is to devise an algorithm that returns a ϵ-level solution, where ϵ is any fixed small reliability level, and that is worst-case optimal over the set of satisfied

constraints. To this purpose, we now introduce the "scenario" version of the worst-case design problem. By scenario it is here meant any possible realization or instance of the uncertainty parameter. In the "scenario design" we optimize the objective subject to a finite number of these randomly selected scenarios.

Definition 4.2.4 (Scenario Design Problem)
Assume that S independent identically distributed samples $\xi^1, \ldots, \xi^S$ are drawn according to probability $\mathcal{P}$. A scenario design problem is given by the convex program

$$\begin{array}{lll} \mathrm{SDP}_S: & \underset{x \in \mathcal{X}}{\text{minimize}} & c^T x \\ & \text{subject to} & g(x, \xi^i) \leq 0, \quad i = 1, \ldots, S. \end{array} \tag{4.2.1}$$

The acronym SDP_S refers to the fact that (4.2.1) is a robust convex program with S constraints. To avoid mathematical clutter, we here assume the following technical condition on the scenario problem:

Assumption 4.2.5 (Feasibility)
For all possible extractions $\xi^1, \ldots, \xi^S$, the optimization problem (4.2.1) is either infeasible, or, if feasible, it attains a unique optimal solution.

In contrast to the worst-case design problem, the scenario problem SDP_S is a standard convex optimization problem with a finite number of constraints S and, hence, its optimal solution $\hat{x}_S$ is (usually) efficiently computable by means of numerical algorithms [17]. Moreover, since only S constraints are imposed in SDP_S, it is clear that the optimal solution of SDP_S is super-optimal for the robust convex problem with all constraints in place, meaning that the objective corresponding to $\hat{x}_S$ outperforms the one achieved with the solution of the worst-case design program. In this way, the scenario approach reduces the conservatism of the worst-case approach.

The fundamental question that now needs to be addressed is: what guarantee can we give on the level of feasibility of the solution $\hat{x}_S$ of SDP_S? The following key Theorem 4.2.6 answers this question.

Before stating the theorem, we need to mention one important fact. Since the constraints $g(x, \xi^i) \leq 0$ are randomly selected, the resulting optimal solution $\hat{x}_S$ is a random variable that depends on the multi-sample extraction $(\xi^1, \ldots, \xi^S)$. Therefore, $\hat{x}_S$ can be a ϵ-level solution for a given random extraction and not for another. In the theorem, the parameter β bounds the probability that is not a ϵ-level solution. Thus, β is the risk of failure, or confidence, associated to the randomized solution algorithm. In other words, we are looking for S big enough, such that the following inequality holds:

$$\sum_{i=0}^{n_x - 1} \binom{S}{i} \epsilon^i (1 - \epsilon)^{S-i} \leq \beta, \tag{4.2.2}$$

where n_x is the dimension of the design variable $x \in \mathcal{X} \subseteq \Re^{n_x}$. The precise reasoning behind this expression can be found in [19]. Although the best (smallest) value of S can

be found by trial and error, the computation of the left-hand-side of (4.2.2) can be a bit difficult because of the evaluation of the binomial coefficient. The best (smallest) explicit bound for this quantity to date can be found in [3]:

Theorem 4.2.6 (Feasibility I)
Let Assumption 4.2.5 be satisfied. Fix two real numbers $\epsilon \in (0,1)$ (level parameter) and $\beta \in (0,1)$ (confidence parameter). If

$$S \geq \left\lceil \frac{1}{\epsilon}\left(\ln\frac{1}{\beta} + n_x + \sqrt{2n_x \ln\frac{1}{\beta}}\right)\right\rceil \tag{4.2.3}$$

($\lceil\cdot\rceil$ denotes the smallest integer greater than or equal to the argument) then, with probability no smaller than $1-\beta$, either the scenario problem SDP_S is infeasible and, hence, also the initial robust convex program is infeasible; or, SDP_S is feasible, and then its optimal solution $\hat{x}_S$ is ϵ-level robustly feasible.

In this theorem, probability $1-\beta$ refers to the S-fold probability $\mathcal{P}^S (= \mathcal{P}\times\cdots\times\mathcal{P}, S$ times) in $\Xi^S = \Xi\times\cdots\times\Xi$, which is the set to which the extracted multisample $(\xi^1,\ldots,\xi^S)$ belongs. Here and elsewhere, the measurability of $\{\mathcal{V}(\hat{x}_S) \leq \epsilon\}$, as well as that of other sets in Ξ^S, is taken as an assumption. The proof of Theorem 4.2.6 can be found in [3].

Theorem 4.2.6 states that if S (specified by (4.2.3)) random scenarios are drawn, the optimal solution of SDP_S is ϵ-level feasible according to Definition 4.2.3, with high probability $(1-\beta)$. Parameter β is important in theory since, if β is pushed down to zero, S goes to infinity. However, for a practical use, we can observe that β plays a very marginal role. The reason is that β shows up in (4.2.3) under the sign of logarithm so that it can be made very small (10^{-10} or even 10^{-20}) without significantly increasing S. The scenario approach thus provides us with a viable and implementable way to make a nominal design more robust up to a desired level ϵ.

Remark 4.2.7 *($\mathcal{P}$-Independent Bound) In many applications, probability $\mathcal{P}$ is not explicitly known, and the scenarios are directly made available as "observations". This could for example be the case when the instances of ξ are actually related to various measurements or identification experiments made at different times and/or different operating conditions. In this connection, it is important to emphasize that the bound (4.2.3)) is probability independent, i.e., it holds irrespective of the underlying probability $\mathcal{P}$, and can therefore be applied even when $\mathcal{P}$ is unknown.*

Next we introduce a concept that is crucial for the success of the upcoming algorithm. Of the S generated scenarios, only some of these S will be "bounding" in the sense that they prevent the solution from "falling" to a lower objective value.

Definition 4.2.8 (Support Scenario)
Scenario $\xi^i, i \in \{1,\ldots,S\}$, is a support scenario for the scenario problem SDP_S if its removal changes the optimal solution of SDP_S.

The following theorem, whose proof can be found in [20] or in a different form in [66], gives us the bound on the number of support scenarios:

Theorem 4.2.9 (Number of Support Scenarios)
The number of support scenarios for SDP_S is at most n_x, the size of x.

What is most important about this result is the fact that the number of support scenarios does not depend on the number of generated scenarios S. If all the S constraints are enforced, however, one cannot expect that good approximations of chance constrained solutions are obtained (cf. the numerical examinations in the following sections). Thus, we want to allow the solution to violate part of the sampled constraints to improve its objective value. A general removal procedure is formalized in the following definition:

Definition 4.2.10 (Constraint Removal Algorithm)
Let $k < S$. An algorithm $\mathcal{A}$ for constraints removal is any rule by which k constraints out of a set of S constraints are selected and removed. The output of $\mathcal{A}$ is the set $\mathcal{A}\{\xi^1, \ldots, \xi^S\} = \{i_1, \ldots, i_k\}$ of the indexes of the k removed constraints.

The sample-based optimization program where k constraints are removed as indicated by $\mathcal{A}$ is expressed as

$$\begin{array}{lll} \mathrm{SDP}^{\mathcal{A}}_{S,k}: & \underset{x \in \mathcal{X}}{\text{minimize}} & c^T x \\ & \text{subject to} & g(x, \xi^i) \leq 0, \quad i \in \{1, \ldots, S\} \setminus \mathcal{A}\{\xi^1, \ldots, \xi^S\}, \end{array} \tag{4.2.4}$$

and its solution will be hereafter indicated as $x^*_{S,k}$. We introduce the following assumptions:

Assumption 4.2.11 (Constraint Violation)
Almost surely with respect to the multi-sample $(\xi^1, \ldots, \xi^S)$, the solution $x^*_{S,k}$ of the sample-based optimization program $\mathrm{SDP}^{\mathcal{A}}_{S,k}$ violates all the k constraints that $\mathcal{A}$ has removed.

This assumption requires that the algorithm $\mathcal{A}$ chooses constraints whose removal improves the solution by violating the removed constraints, and it rules out for example algorithms that remove inactive constraints only, or algorithms that remove constraints at random. Thus, this assumption is very natural and reflects the fact that we want to remove the constraints that improve the optimal objective value.

The next Theorem (proved in [21]) provides theoretical guarantees that $\mathcal{V}(x^*_{S,k}) \leq \epsilon$, i.e. that the optimal solution $x^*_{S,k}$ of the optimization program $\mathrm{SDP}^{\mathcal{A}}_{S,k}$ is feasible for the CCP_ϵ.

Theorem 4.2.12 (Feasibility)
Let $\beta \in (0, 1)$ be any small confidence parameter value. If S and k are such that

$$\binom{k + n_x - 1}{k} \sum_{i=0}^{k+n_x-1} \binom{S}{i} \epsilon^i (1 - \epsilon)^{S-i} \leq \beta, \tag{4.2.5}$$

then $\mathcal{P}^S\{\mathcal{V}(x^*_{S,k}) \leq \epsilon\} \geq 1 - \beta$.

The final result establishes that the objective value of CCP_ϵ (whose optimal objective value will be denoted as J^*_ϵ) can be approached at will, provided that sampled constraints are optimally removed. Let $\mathcal{A}_{opt}$ be the optimal constraints removal algorithm which leads – among all possible eliminations of k constraints out of S – to the best possible improvement in the cost objective; further, let $x^*_{S,k,opt}$ and $J^*_{S,k,opt}$ be the corresponding optimal solution and objective value. We have the following theorem (again, proved in [21]).

Theorem 4.2.13 (Optimality)
Let $\beta \in (0,1)$ be any small confidence parameter value, and let $\nu \in (0,\epsilon)$ be a performance degradation parameter value. If S and k are such that

$$\binom{k+n_x-1}{k}\sum_{i=0}^{k+n_x-1}\binom{S}{i}\epsilon^i(1-\epsilon)^{S-i}+\sum_{i=k+1}^{S}\binom{S}{i}(\epsilon-\nu)^i(1-\epsilon+\nu)^{S-i}\leq\beta, \quad (4.2.6)$$

then
(i) $\mathcal{V}(x^*_{S,k}) \leq \epsilon$
(ii) $J^*_{S,k,opt} \leq J^*_{\epsilon-\nu}$
simultaneously hold with probability at least $1-\beta$.

One optimal way of removing constraints consists in discarding those constraints that lead to the largest possible improvement of the cost function. This approach is implemented by the following integer program, which has been described and investigated in [69], [70] and [87]:

$$\begin{array}{lll} \underset{x\in\mathcal{X}}{\text{minimize}} & c^T x & \\ \text{subject to} & g(x,\xi^i)-Mz_i\leq 0, & i=1,\ldots,S, \\ & \sum_{i=1}^{S} z_i \leq k, & z\in\{0,1\}^S. \end{array} \quad (4.2.7)$$

where M is a constant large enough so that, if $z_i = 1$, then the constraint is satisfied for any candidate solution x. For $k = 0$, the formulations (4.2.1) and (4.2.7) are equivalent. By construction, problem (4.2.7) provides a framework for optimally selecting the constraints to be removed based on the inequality (4.2.6). However, solving (4.2.7) may be computationally challenging due to the increase in complexity from (4.2.1) to (4.2.7) that arises from the introduction of one binary variable per each of the S scenarios. In recent years, there have been developed strengthening procedures (see [105] and [2]) for some special structured problems, that significantly improve upon the formulation (4.2.7).

4.3 | Pool & Discard Algorithm

In this section we describe a novel approach for solving the SDP_S formulation (4.2.1) and an constraint removal algorithm that improves upon the obtained solution. We call this procedure the Pool & Discard algorithm (P&D).

4.3.1 | Pooling Part

The idea behind the Pooling part of the algorithm is the following: if one were to verbally describe the problem (4.2.1), the one word that came to our mind was "long", as there are much more constraints than decision variables. Moreover, the number of support constraints (or support scenarios), that the optimal solution of (4.2.1) depends upon is very small, when compared to the overall number of constraints (or scenarios).

The method consists of solving (4.2.1) by the following procedure. First, we start by completely neglecting the constraints in (4.2.1) that correspond to the different scenarios and solve this relaxed optimization problem. Then we find the most violated constraints (by computing the slacks), add them to the relaxed problem and find a new optimal solution – this step heavily exploits warm-starting, cf. Section 4.3.4. The Pooling part can be summarized as follows:

Step 0. Set $\delta > 0$, $\mathcal{I} = \emptyset$.

Step 1. Solve the following problem:

$$\begin{aligned} \underset{x \in \mathcal{X}}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & g(x, \xi^i) \leq 0, \quad i \in \mathcal{I}, \end{aligned} \tag{4.3.1}$$

and obtain a solution $\hat{x}$.

Step 2. Check feasibility of the solution by computing the slacks s^i:

$$s^i = g(\hat{x}, \xi^i), \quad i \in \{1, \ldots, S\}. \tag{4.3.2}$$

Step 3. If $\max\limits_{i \in \{1,\ldots,S\}} s^i > \delta$, find the associated index of the maximum value $\hat{i} = \operatorname{argmax}\limits_{i \in \{1,\ldots,S\}} s^i$, add it to the set $\mathcal{I}$ and return to Step 1. Otherwise, set $x^* = \hat{x}, \mathcal{I}^* = \mathcal{I}$ and terminate.

The parameter δ, can (theoretically) be set to zero, but there are implementation issues that would lead to unfavourable results, cf. Section 4.3.4. It is important to remark that by the end of this procedure, we not only get the optimal solution of (4.2.1), but also an index set $\mathcal{I}$ that contains the support scenarios – this will be very significant for the success of second part of the algorithm.

4.3.2 | Discarding Part

The Discarding part of the algorithm consists of utilizing the index set $\mathcal{I}$, finding the support scenarios among this set and finding the one scenario, whose removal decreases the optimal objective value the most – this is repeated k times, where k is either set a priori (by Theorem 4.2.12), or is terminated once an estimate of the probability of violation of obtained solution $\mathcal{V}(x)$ reaches certain threshold. This approach is very similar to the one discussed in [88], with the distinction that our algorithm utilizes the Pooling step throughout the iterations and as such can be rather effective (cf. the sections with numerical examinations and examples). The Discarding part can be summarized as follows:

Step 0. Solve the pooling part described above to obtain $\mathcal{I}^*$ and x^*. Set $\gamma > 0, k > 0, \mathcal{I}_p = \emptyset$.

Repeat k times, or terminate once an estimate of $\mathcal{V}(x^*)$ reaches a threshold:

Step 1. Find the set of support scenarios $\mathcal{I}_r \subset \mathcal{I}^*$ – either by examining the slacks ($s^i > -\gamma$) or the associated dual variables ($\mu^i > \gamma$).

Step 2. For each of the support scenarios $i_r \in \mathcal{I}_r$, solve the following problem:

$$\begin{aligned} &\underset{x \in \mathcal{X}}{\text{minimize}} && c^T x \\ &\text{subject to} && g(x, \xi^i) \leq 0, \quad i \in \{1, \ldots, S\} \setminus \{i_r \cup \mathcal{I}_p\}, \end{aligned} \tag{4.3.3}$$

using the Pooling part, warm-started by using $\mathcal{I} = \mathcal{I}^* \setminus \{i_r\}$ and $x = x^*$. Denote the solution to (4.3.3) as $x^*_{i_r}$, its optimal objective function value v^*_r and its final set of scenarios $\mathcal{I}^*_r$.

Step 3. Find the index with the best optimal objective value: $i^* = \underset{i_r}{\operatorname{argmin}}\, v^*_r$. Set $x^* = x^*_{i^*}$, $\mathcal{I}^* = \mathcal{I}^*_{i^*}$ and add the corresponding scenario to the set of permanently discarded ones $\mathcal{I}_p$.

In a similar fashion to the parameter δ in the Pooling part, the parameter γ can be, in theory, set to 0. What discourages us from doing so are the implementation issues discussed in Section 4.3.4. It should be added, that Step 2. of the Discarding part can be fully parallelized to work more efficiently on multi-core machines.

4.3.3 | Linearized Modification

There are situations, where one does not know the constraint functions $g(x, \xi^i)$ explicitly, but can get the function value and a subgradient at any point – the resulting optimization models are called oracle, black-box or subroutine models [17]. Other times one may not have access to powerful enough solvers/software to solve the optimization problems in the P&D algorithm efficiently. For these two situations, we propose a linearized variant of the P&D algorithm – this extension involves a standard cutting-plane method [51].

As before, $g(x, \xi^i)$ is assumed convex in x for any $i \in \{1, \ldots, S\}$. Suppose that for any $i \in \{1, \ldots, S\}$ and for any arbitrarily chosen $\bar{x}$, we can evaluate the function $g(\bar{x}, \xi^i)$ and can obtain a subgradient $d \in \partial g(\bar{x}, \xi^i)$, where $\partial g(\bar{x}, \xi^i)$ is the subdifferential (see [47]) of $g(x, \xi^i)$ at $\bar{x}$. Since $g(x, \xi^i)$ is convex in x, the following inequality holds for any x:

$$g(x, \xi^i) \geq g(\bar{x}, \xi^i) + d^T(x - \bar{x}).$$

Instead of modeling the feasible set explicitly by enforcing $g(x, \xi^i) \leq 0$, $i \in \{1, \ldots, S\}$ (to which we either do not have access or cannot model using the available solver), we iteratively construct an outer polyhedral approximation of this feasible set. The Pooling step of the algorithm is changed a following way:

Step 0. Set $\delta > 0$, $\mathcal{I} = \emptyset$, $J = 0$ (i.e., E is an empty matrix and e, f are empty vectors).

Step 1. Solve the following problem:

$$\begin{aligned} &\underset{x\in\mathcal{X}}{\text{minimize}} && c^T x \\ &\text{subject to} && E_j x + e_j \leq 0, \quad j = 1, \ldots, J, \end{aligned} \tag{4.3.4}$$

where E_j is the j-th row of the matrix E, and obtain a solution $\hat{x}$.

Step 2. Check feasibility of the solution by computing the slacks s^i:

$$s^i = g(\hat{x}, \xi^i), \quad i \in \{1, \ldots, S\}. \tag{4.3.5}$$

Step 3. If $\max\limits_{i\in\{1,\ldots,S\}} s^i > \delta$, find the associated index of the maximum value $\hat{i} = \operatorname{argmax}\limits_{i\in\{1,\ldots,S\}} s^i$, increase J by 1, compute $d \in \partial g(\hat{x}, \xi^{\hat{i}})$, add a new row to the matrix $E_J = d^T$, a new value to the vectors $e_J = g(\hat{x}, \xi^{\hat{i}}) - d^T\hat{x}$, $f_J = \hat{i}$ and return to Step 1. Otherwise, set $x^* = \hat{x}$ and terminate.

The main difference between the normal and the linearized variant (apart from the polyhedral approximation) is that we need to store for each row of E and e the index of the scenario that generated it (as a single scenario can generate multiple cuts), this information is stored in f – although the Pooling part would work just fine without this storage, the Discarding part would not.

To accommodate the different structure, the Discarding part is changed a following way:

Step 0. Solve (4.2.1) using the linearized Pooling part described above to obtain E, e, f, J and x^*. Set $\gamma > 0, k > 0, \mathcal{I}_p = \emptyset$.

Repeat k times, or terminate once an estimate of $\mathcal{V}(x^*)$ reaches a threshold:

Step 1. Find the set of support scenarios $\mathcal{I}_r$ within the indexes stored in f – either by examining the slacks ($s^i > -\gamma$) or the associated dual variables ($\mu^i > \gamma$).

Step 2. For each of the support scenarios $i_r \in \mathcal{I}_r$, find the set $\mathcal{J}_r$ of indexes of f, which have the value i_r and solve (4.2.1) using the linearized Pooling, neglecting the scenarios $i_r \cup \mathcal{I}_p$ in the computation of slacks, starting from:

$$\begin{aligned} &\underset{x\in\mathcal{X}}{\text{minimize}} && c^T x \\ &\text{subject to} && E_j x + e_j \leq 0, \quad j \in \{1, \ldots, J\} \setminus \mathcal{J}_r, \end{aligned} \tag{4.3.6}$$

and get x_r^*, E_r, e_r, f_r and the optimal objective function value v_r^*.

Step 3. Find the index with the best optimal objective value: $i^* = \operatorname{argmin}\limits_{i_r} v_r^*$. Set $x^* = x_{i^*}^*, E = E_{i^*}, e = e_{i^*}, f = f_{i^*}$ and add the corresponding scenario to the set of permanently discarded ones $\mathcal{I}_p$.

4.3.4 | Implementation

In this section we describe the little nuances in implementation, that make rather significant contribution to the applicability of the P&D algorithm. The Pooling part of the P&D

algorithm consists of, essentially, solving a very similar problem many times – just adding one constraint at a time. So, when one implements such a procedure, the natural thing is to exploit the "closeness" of the problems to solve. There are solvers that allow and encourage a kind of problem modification, that is much faster than building and solving a new (although similar) model from scratch every time. One of these solvers is the one we used in the implement the P&D algorithm, `CPLEX 12.7` [49].

Even though the solver itself supports the feature of modifying the problem, it does not mean that the modeling system we write the problem in (and use it as the "intermediary" between our model and the solver) supports the feature as well. For example, at the time of writing neither `AIMMS` [15], `MATLAB` nor `GAMS` [35] support problem modification – this means that the performance of the P&D algorithm, when written either entirely in or using these systems, would be much worse than the performance presented in the upcoming numerical sections. For our implementation we chose a relatively new programming language `Julia` [12], that is designed for high-performance numerical computing, and a modeling system `JuMP`, which is domain-specific modeling language for mathematical optimization embedded in `Julia`. `JuMP` supports a wide variety of solvers, model modifications, warm-stars, and even different solver callbacks (lazy constraints, etc.) that, even they are not useful for the P&D algorithm, make it a very powerful modeling tool.

Now we get to the two parameters in P&D that need explanation, namely δ in the Pooling part and γ in the Discarding part. The first of these two parameters, δ, is the required feasibility of the solution. In theory, this could (and should) be set to zero, to guarantee that the solution of the Pooling part "really" solves the SDP_S formulation (4.2.1). The problem is that some solvers, when given an optimization problem to solve, the "optimal" solution they provide is not always strictly feasible. Among these solvers are, for example, `CPLEX` and `GUROBI` [41]. In `CPLEX`, the parameter that sets the tolerance for the feasibility of the optimal solution (which is an extremely uncomfortable term) is `CPX_PARAM_EPRHS`, has a default value of 10^{-6} and can be set anywhere between 10^{-9} and 10^{-1}, but not to 0. In `GUROBI`, this parameter is called `FeasibilityTol`, has a similar range and a description that says: "All constraints must be satisfied to a tolerance of `FeasibilityTol`." This is the reason we need a nonzero δ, because when set to zero, the Pooling part can (and, when we tried to set it to 0, sometimes did) end up in an infinite loop, because it cannot produce (at least when using these solvers) a feasible point. Unless stated otherwise, the parameter δ was set to 10^{-7}.

The second parameter, γ in the Discarding part, controls which scenarios will be treated as possible support scenarios. From complementary slackness [17] we know that for any primal optimal x^* and dual optimal μ the following holds:

$$\mu^i g(x^*, \xi^i) = 0, i \in \{1, \dots, S\}.$$

We can express this, equivalently as

$$\mu^i > 0 \Longrightarrow g(x^*, \xi^i) = 0,$$

or

$$g(x^*, \xi^i) < 0 \Longrightarrow \mu^i = 0.$$

Depending on whether or not we have access to the dual optimal μ (in most solvers we do), we inspect either the slacks (if they are 0) or the dual variables μ (if they are greater then 0), to find a set of possible support scenarios. The issue of setting γ to 0 is one of numerical computing (and the feasibility tolerance mentioned earlier) – when reporting the optimal dual variables μ the solvers rarely return exactly 0, more often, we get values ranging from 10^{-8} to 10^{-16}. If we did set γ to 0 we would (likely) have to consider all the scenarios as possible support scenarios and the execution of the algorithm would be significantly prolonged. Unless stated otherwise, the parameter γ was set to 10^{-5}.

The last thing to mention concerns the machine, on which we conducted the numerical examples. It was a PC with 3.2 GHz i5-4460 CPU, 16 GB RAM, NVIDIA GeForce GTX 770, running on 64-bit Windows 10.

4.4 | Linear Example – Optimal Asset Allocation

The first numerical example we chose to demonstrate the utility of the P&D algorithm is the (by now, almost canonical) asset allocation problem. Suppose we have n assets $x_1, \ldots, x_n$ that we want to invest in. The returns $r_1, \ldots, r_n$ of these assets are random variables. Our goal is to allocate our resources to these different assets, in order to maximize the ϵ quantile (often called the Value at Risk, or VaR) of the returns. This formulation neglects several of the important real-world issues – we do not allow short position, do not consider more than one trading period, etc. – the example is, above all else, intended to show the capabilities of the P&D algorithm. Our asset allocation problem can be summarized as follows:

$$\begin{array}{ll} \underset{x \geq 0, t \in \Re}{\text{maximize}} & t \\ \text{subject to} & \mathcal{P}\{t \leq \sum_{j=1}^{n} r_j x_j\} \leq \epsilon, \\ & \sum_{i=j}^{n} x_j \leq 1. \end{array} \tag{4.4.1}$$

Our ability to solve (with no quotation marks) this problem depends heavily on the distribution of the returns $r_1, \ldots, r_n$ and the chosen quantile ϵ. Thanks to [61], we know that the feasible set of a scalar chance constraint

$$\mathcal{P}\{a^T x \leq b\} \leq \epsilon,$$

is convex, provided that the vector $(a^T, b)^T$ of the coefficients has symmetric logarithmically concave density and $\epsilon < 1/2$. We will use this result and model the returns r as random variables that are independent and normally distributed (and, hence, have a symmetric logarithmically concave density). More precisely, the return r_j has the following distribution

$$r_j \sim \mathcal{N}(\mu_j, \sigma_j), \qquad \mu_j = 1 + 0.1\frac{j-1}{n-1}, \qquad \sigma_j = 0.1\frac{j-1}{n-1},$$

i.e., the first return is "deterministic", with return $r_1 = 1$, and the nth return has mean $\mu_n = 1.1$ and standard deviation $\sigma_n = 0.1$. Because of the chosen distribution of returns, the problem (4.4.1) can be transformed into the following second order cone problem (SOCP, see [17]):

$$\begin{array}{ll} \underset{x\geq 0, t\in\Re}{\text{maximize}} & t \\ \text{subject to} & \sum_{j=1}^{n}(1+\mu_j)\cdot x_j \geq t + \Phi^{-1}(1-\epsilon)\cdot||(\sigma_1\cdot x_1, \ldots, \sigma_n\cdot x_n)||_2, \\ & \sum_{j=1}^{n} x_i \leq 1, \end{array} \tag{4.4.2}$$

where $\Phi^{-1}(1-\epsilon)$ is the $1-\epsilon$ quantile of the standard normal distribution. As an SOCP, this problem falls into the category of "easy" to solve (we can compute the optimal solution with little effort for large values of n – well into thousands) and as such provides the perfect ground for illustrating the capacities of the P&D algorithm.

The sample (or scenario) approach, as discussed in Section 4.2, works with a sample of S scenarios of the returns $r_j^i, j = 1, \ldots, n, i = 1, \ldots, S$. Using these scenarios, the sample counterpart to (4.4.1) has the following form:

$$\begin{array}{ll} \underset{x\geq 0, t\in\Re}{\text{maximize}} & t \\ \text{subject to} & t \leq \sum_{j=1}^{n} r_j^i x_j, \quad i \in \{1, \ldots, S\} \\ & \sum_{j=1}^{n} x_j \leq 1. \end{array} \tag{4.4.3}$$

First of all, we will investigate on (4.4.3) the dependence of computation time of the Pooling part of the P&D algorithm for varying number of assets n and scenarios S. We compare this with solving the formulation (4.4.3) as is (i.e., passing the full problem to `CPLEX`). The results are summarized in the Table 4.1, and, in more detail, in Figure 4.1 and Figure 4.2. The "setting up" time accounts for the problem construction only, i.e. passing the structure and data of the problem to `CPLEX`.

These results show that the Pooling part, although it is slower for smaller number of scenarios, scales much better and becomes much more practical when one has to deal with large number of scenarios. Our cap on number of considered scenarios were the memory requirements for solving the full model, which were extremely higher than the memory requirements of the Pooling part of the P&D algorithm. For example, for the $n = 30$, $S = 10^6$ case, the full problem required 8GB of RAM, while the Pooling part required only 300MB of RAM, of which 210MB were the problem data (30×10^6 matrix, double-precision format).

n	S	setting up [s]	full [s]	Pooling [s]	iter.	obj. diff.
30	100	>0.001	>0.01	0.01	23	$1.9{\cdot}10^{-16}$
	2,154	0.006	0.04	0.03	45	$1.3{\cdot}10^{-16}$
	46,416	0.140	2.56	0.14	64	$2.6{\cdot}10^{-16}$
	1,000,000	3.091	83.03	2.76	86	$2.2{\cdot}10^{-16}$
300	100	0.002	0.01	0.05	54	$2.6{\cdot}10^{-16}$
	1,000	0.018	0.18	0.25	139	$5.3{\cdot}10^{-16}$
	10,000	0.248	5.29	1.03	220	$4.6{\cdot}10^{-16}$
	100,000	2.399	105.60	6.20	304	$6.6{\cdot}10^{-16}$
1,000	100	0.004	0.04	0.16	70	$3.3{\cdot}10^{-16}$
	618	0.036	0.59	1.71	213	$5.5{\cdot}10^{-16}$
	8,799	0.502	12.14	6.73	374	$8.4{\cdot}10^{-16}$
	31,623	2.741	88.65	21.29	527	$8.6{\cdot}10^{-16}$

Table 4.1: Computation time for setting up the full problem, solving the full problem, using the Pooling part, and number of Pooling part iterations, and difference in optimal objective function value between the two. Varying number of variables and scenarios. Average over 10 runs.

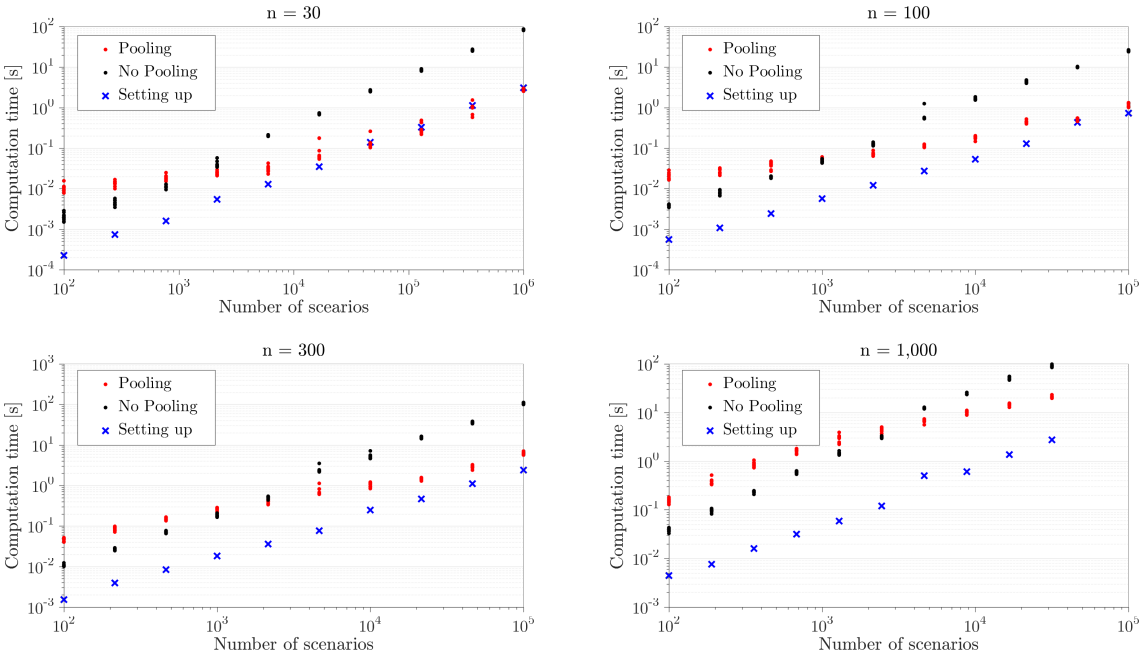


Fig. 4.1: Dependence of the computation time on number of scenarios for the pooling step (red). Comparison with setting up (blue) and solving (black) the problem without pooling.

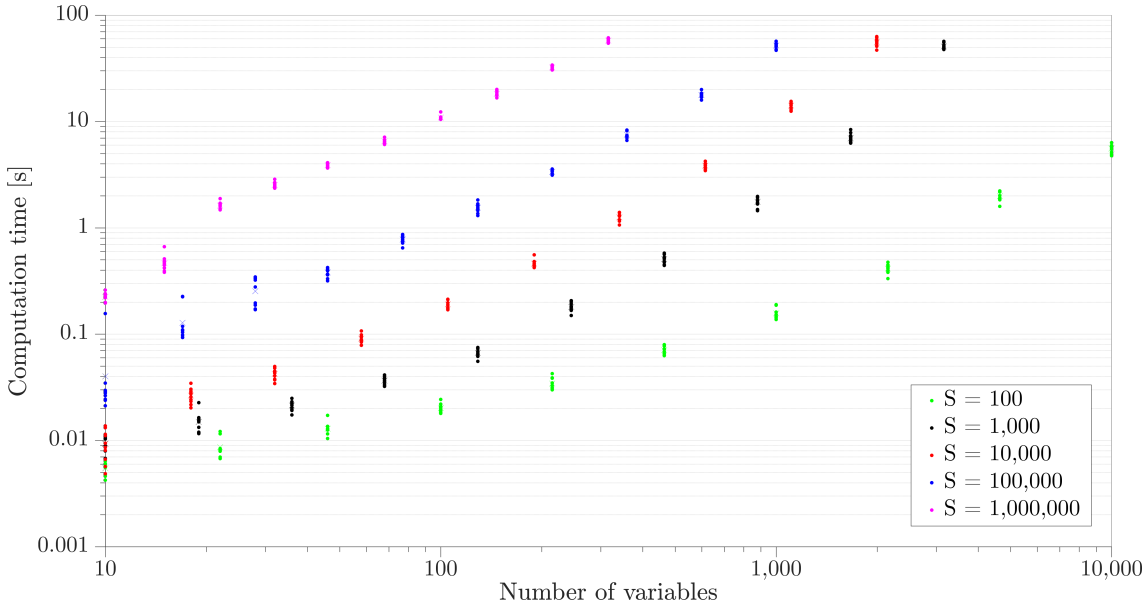


Fig. 4.2: Dependence of the computation time on number of variables for the pooling step. Different number of scenarios.

In the log-log plots in Figure 4.1, the dependence of computation time on the number of scenarios S for a fixed number of variables n appears to have a linear trend, which suggests a polynomial dependence. The same cannot be said about the dependence of computation time on the number of variables n for a fixed number of scenarios in Figure 4.2.

n	S	t [s]	n	S	t [s]	n	S	t [s]
30	100	0.04	50	100	0.06	100	100	0.14
	1,000	0.61		1,000	1.11		501	1.64
	10,000	14.17		10,000	30.32		2,512	15.99
	100,000	723.32		100,000	1,494.59		12,589	176.62
300	100	0.50	500	100	0.95	1,000	100	2.82
	464	9.34		464	26.70		271	40.04
	2,154	133.46		2,154	492.98		736	377.35
	10,000	1,749.09		10,000	5,585.00		1,995	2,394.35

Table 4.2: Computation time for the P&D algorithm. Different number of variables and scenarios, $\epsilon = 0.01$. Average over 10 runs.

Next, we examine the computational time for the whole P&D algorithm for varying number of variables and scenarios. In the Discarding part of the algorithm, we decided to discard $\lfloor 0.01\, S \rfloor$ scenarios – note that this choice does not guarantee, that the resulting solution obtained by the P&D algorithm will be a 0.01-level robustly feasible (see Definition 4.2.3), not to mention having the objective value close to the optimal value objective $J^*_{0.01}$. The results, summarized in Table 4.2 and Figure 4.3, are very encouraging – the linear trends of the log-log plot suggest a polynomial dependence on the number of scenarios. Compare

this to the computational requirements of the mixed-integer formulation (4.2.7) – this formulation achieves the optimal constraint removal, but computational requirements grow incomparably faster.

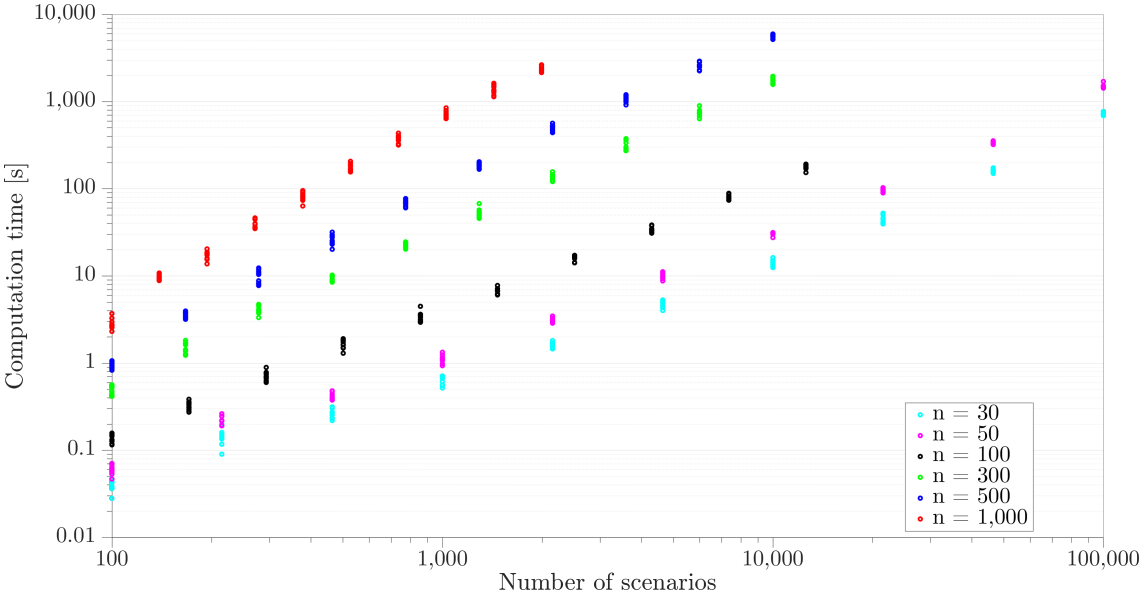


Fig. 4.3: Dependence of the computation time of the P&D algorithm on the number of variables and scenarios. The number of scenarios to discard was $\lfloor 0.01\, S \rfloor$.

The real crux of the matter, however, is the following: "How good a solution (in terms of ϵ-level feasibility robust feasibility and objective value) do we get by using the P&D algorithm?" The remarkable thing about our optimal asset allocation problem is that for a chosen value of ϵ, we can get the optimal solution by solving the SOCP (4.4.2). Moreover, for every asset allocation x, we can find the corresponding ϵ quantile of the returns exactly. Or, alternatively, we can for a given value of the returns t and a given asset allocation x compute (again, exactly) the probability $\mathcal{P}\{t \leq \sum_{j=1}^{n} r_j x_j\}$ (i.e., the smallest value of ϵ, for which our choice of x and t is feasible).

For the examination, we chose a problem with $n = 30$ assets and $\epsilon = 0.01$. The optimal objective value (obtained by solving (4.4.2)) was 1.0309. The results are summarized in Table 4.3, Figure 4.4 and Figure 4.5. Using Theorem 4.2.6, with $\beta = 10^{-10}$, we get that to obtain a feasible solution to this problem with high probability $(1 - \beta)$, we need to solve (4.4.3) with at least $S = 8{,}547$ scenarios (without any discarding). The solution to this problem had the objective value 1.0179 (second column of Table 4.3), with the reliability 0.0029 (i.e. $\mathcal{P}\{t \leq \sum_{j=1}^{n} r_j x_j\} = 0.0029$) – i.e. we obtained a feasible solution, but with a rather poor objective value. Afterwards, we ran the Discarding part of the algorithm, discarding $\lfloor \epsilon S \rfloor$ scenarios. The objective value improved to 1.0318 (fourth column of the table), but the corresponding reliability dropped to 0.0138 – meaning that the combination of x and t (obtained after discarding) was no longer feasible. However, during the Discarding part of the algorithm we stored the particular solutions in each iteration. This allows us to find the last admissible (feasible) solution and find its corresponding objective

and a number of scenarios that we discarded to get it – in this case the objective value was 1.0291 (sixth column of the table) with 31 discarded scenarios. An interesting thing to note is that even for 5,000 scenarios we still get a feasible solution (with reliability 0.0041) and can remove some scenarios, but for the lower numbers of scenarios even the "robust" solution is not feasible.

When we vary the number of scenarios several interesting phenomena appears. Firstly, when increasing the number of scenarios S we get a smaller value of the "robust" solution objective (the solution after the Pooling part) and the higher the corresponding reliability (both of these are rather intuitive). Secondly, when we increase the number of scenarios, the reliability of the solution after discarding $\lfloor \epsilon S \rfloor$ scenarios approaches ϵ and the number of removed scenarios for an admissible solution gets closer to $\lfloor \epsilon S \rfloor$. Thirdly, and most impressively, the admissible solution objective gets surprisingly close to the optimal value of (4.4.2).

Another feature of the P&D algorithm is that since we remove one scenario at a time, we can use the successive results to construct an approximation of the trade-off between reliability and optimal objective function value. This is best shown on Figure 4.4, where we can see the progression of the P&D algorithm for different number of scenarios – each point corresponds to a solution with different number of removed scenarios (typically, more removed scenarios correspond to points more up and to the right). We included the optimal trade-off curve obtained by solving the SOCP (4.4.2) for different values of ϵ (called "exact solution" in the legend of Figure 4.5).

It must be emphasized that the P&D algorithm does not in any way incorporate any knowledge about the underlying distribution of the random variables. All it "sees" are the realizations in the form of individual scenarios.

number of scenarios S	"robust" solution objective	"robust" solution reliability	objective after discarding $\lfloor \epsilon S \rfloor$ scenarios	reliability after discarding $\lfloor \epsilon S \rfloor$ scenarios	admissible solution objective	number of removed scenarios for admissible solution
500	1.0324	0.0291	1.0400	0.0466	–	–
1,000	1.0292	0.0183	1.0377	0.0368	–	–
2,000	1.0289	0.0120	1.0336	0.0220	–	–
5,000	1.0231	0.0041	1.0326	0.0160	1.0303	31
8,547	1.0179	0.0029	1.0318	0.0138	1.0291	53
20,000	1.0167	0.0014	1.0319	0.0128	1.0304	147
50,000	1.0140	0.0004	1.0310	0.0111	1.0305	463
100,000	1.0129	0.0003	1.0309	0.0102	1.0308	980
250,000	1.0101	0.0001	1.0308	0.0101	1.0308	2,487

Table 4.3: The "quality" of the solutions produced by the P&D algorithm, $n = 30, \epsilon = 0.01$. Varying number of scenarios. The true optimal objective function value of (4.4.2)) was 1.0309.

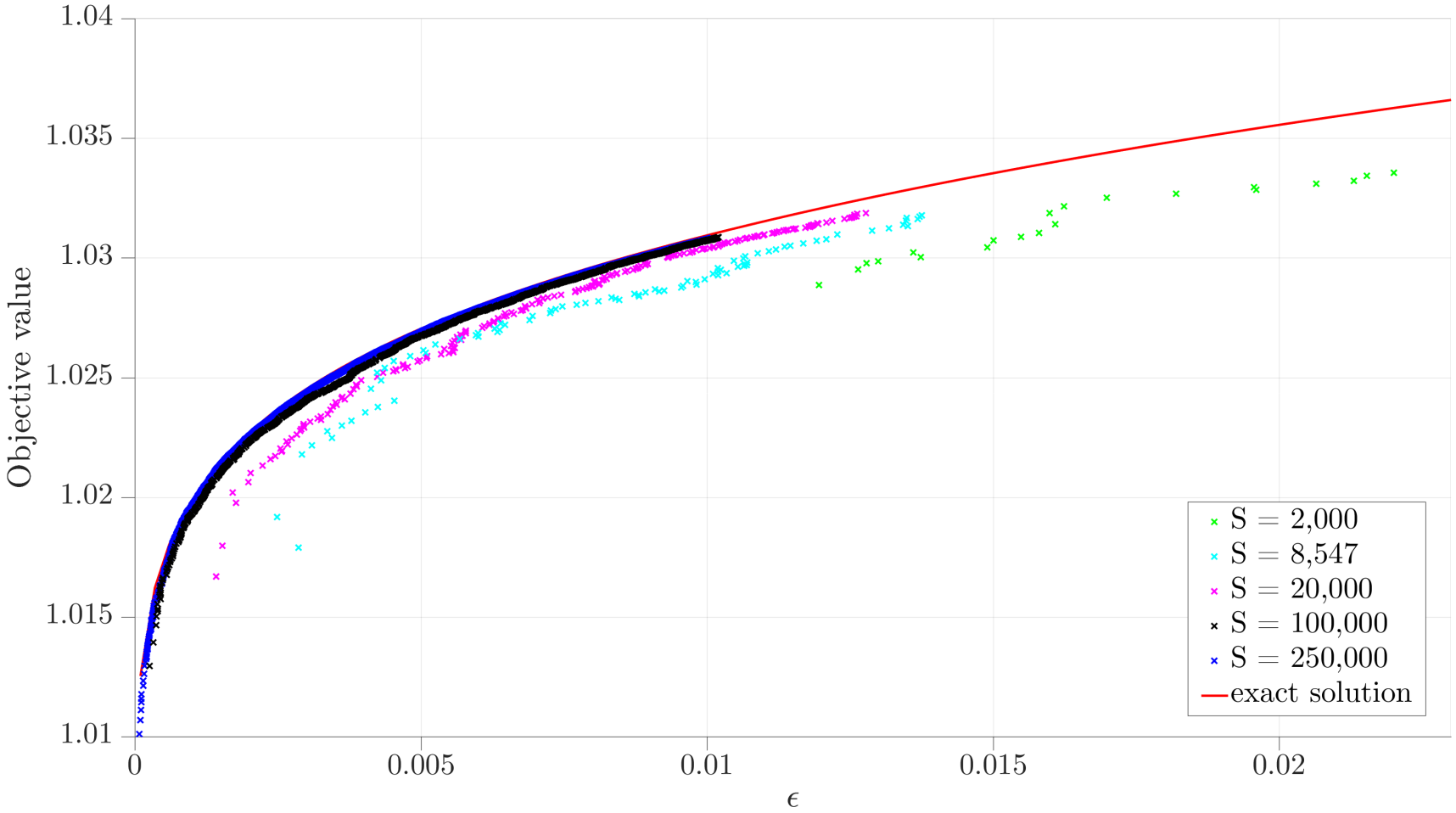


Fig. 4.4: The "quality" of the solutions produced by the P&D algorithm, $n = 30$. Varying number of scenarios.

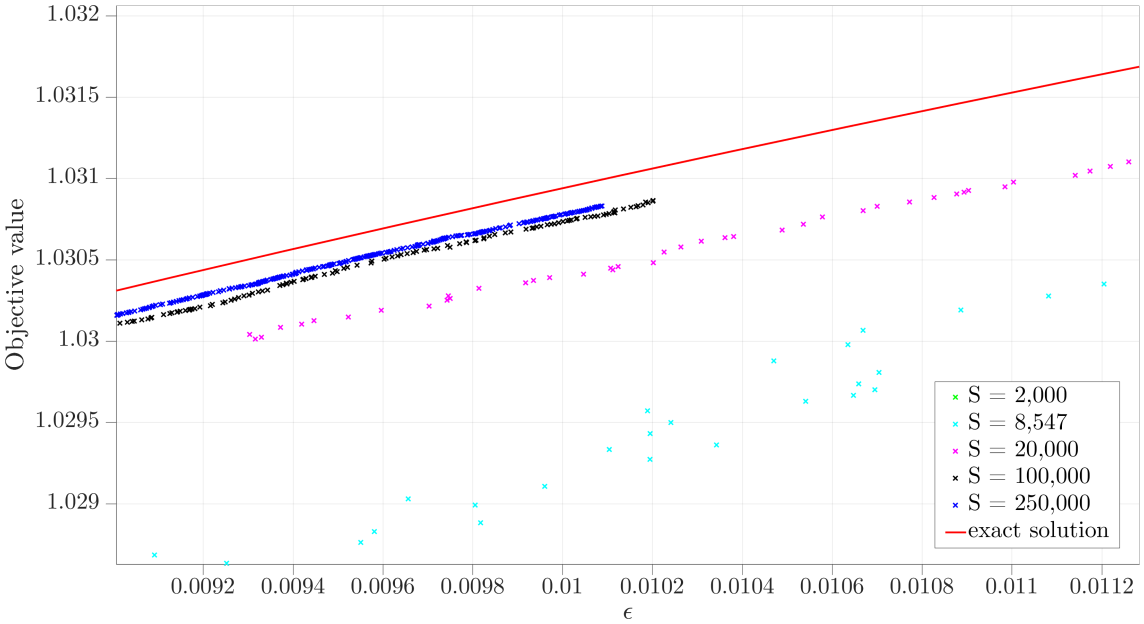


Fig. 4.5: The "quality" of the solutions produced by the P&D algorithm, $n = 30$. Varying number of scenarios. Close up on $\epsilon = 0.01$.

4.5 | Comparison with Bernstein Approximation

The Bernstein approximation, developed in [82] and [81], is a convex approximation of the chance constrained program, that is computationally tractable – it transforms the problem into an efficiently solvable deterministic optimization program with the feasible set contained in the chance constrained problem (and, hence, it is a conservative approximation). The (considerably shortened) description of the Bernstein approximation that follows (is Sections 4.5.1 and 4.5.2) is taken directly from [82].

This approximation is suited for chance constrained problems in the following form:

$$\begin{array}{ll} \underset{x \in \mathcal{X}}{\text{minimize}} & c^T x \\ \text{subject to} & \mathcal{P}\{F(x,\xi) \leq 0\} \geq 1 - \epsilon. \end{array} \tag{4.5.1}$$

Here, ξ is a random vector with probability distribution $\mathcal{P}$ supported on a set $\Xi \subset \Re^{n_\xi}$, the components ξ_j of ξ have probability distribution $\mathcal{P}_j$ and are supported on $\Xi_j \subset \Re, j = 1, \ldots, n_\xi$, $\mathcal{X} \subset \Re^{n_x}$ is a convex set, $F = (f_1, \ldots, f_m) : \Re^{n_x} \times \Xi \to \Re^m$.

4.5.1 | Convex Approximations of Chance Constrained Problems

Before defining the Bernstein approximation, we first look at a generic convex approximation of the chance constraint in (4.5.1). All of the following material is taken from

[82]. First, let us investigate the scalar case of $m = 1$ – the chance constraint in (4.5.1) is equivalent to the constraint

$$p(x) = \mathcal{P}\{F(x,\xi) > 0\} \leq \epsilon. \tag{4.5.2}$$

Let $\mathbf{1}_A$ be the indicator function of a set A, i.e., $\mathbf{1}_A(z) = 1$ if $z \in A$ and $\mathbf{1}_A(z) = 0$ if $z \notin A$.

Let $\psi : \Re \to \Re$ be a nonnegative valued, nondecreasing, convex function satisfying: $\psi(z) > \psi(0) = 1$ for any $z > 0$. This function $\psi(z)$ is called a (one-dimensional) generating function. It follows from its definition that for $t > 0$ and random variable Z,

$$\mathbb{E}[\psi(tZ)] \geq \mathbb{E}[\mathbf{1}_{[0,+\infty)}(tZ)] = \mathcal{P}\{tZ \geq 0\} = \mathcal{P}\{Z \geq 0\}.$$

By taking $Z = F(x,\xi)$ and changing t to t^{-1}, we get that

$$p(x) \leq \mathbb{E}[\psi(t^{-1}F(x,\xi))] \tag{4.5.3}$$

holds for all x and $t > 0$. Let us denote

$$\Psi(x,t) = t\mathbb{E}[\psi(t^{-1}F(x,\xi))]. \tag{4.5.4}$$

If there exists $t > 0$ such that $\Psi(x,t) \leq t\epsilon$, then $p(x) \leq \epsilon$. This statement can be strengthened to

$$\inf_{t>0}[\Psi(x,t) - t\epsilon] \leq 0 \quad \text{implies} \quad p(x) \leq \epsilon. \tag{4.5.5}$$

Moreover, if we assume that for every $\xi \in \Xi$ the function $F(\cdot,\xi)$ is convex, then $G(x,t) = \Psi(x,t) - t\epsilon$ is convex.

Under the assumption that $\mathcal{X}$ and $F(\cdot,\xi)$ are convex, the following problem

$$\begin{array}{ll} \underset{x\in\mathcal{X},t>0}{\text{minimize}} & c^T x \\ \text{subject to} & \inf_{t>0}[\Psi(x,t) - t\epsilon] \leq 0, \end{array} \tag{4.5.6}$$

gives a convex conservative approximation of the chance constraint problem (4.5.1).

The construction of the approximation depends on a choice of the generating function $\psi(z)$. This raises a natural question: "What would be a best choice of $\psi(z)$?" If we consider this question from the point of view of a better (tighter) approximation of the chance constraint in (4.5.1), then the smaller is $\psi(\cdot)$, the better is bound (4.5.3). In this sense, the best one (up to scaling $z \leftarrow z/a$) is the function

$$\psi^*(z) = [1 + z]_+, \tag{4.5.7}$$

where $[a]_+ = \max\{a, 0\}$. For this choice of a generating function, the approximate constraint (4.5.5) takes the form

$$\inf_{t>0}[\mathbb{E}[[F(x,\xi) + t]_+] - t\epsilon] \leq 0. \tag{4.5.8}$$

If we replace in the left-hand side $\inf_{t>0}$ with $\inf_t$, we do not affect the validity of the relation. This means that we can rewrite (4.5.8) equivalently as

$$\inf_{t\in\Re}[\mathbb{E}[[F(x,\xi)+t]_+]-t\epsilon]\leq 0. \tag{4.5.9}$$

In that form the constraint is related to the concept of conditional value at risk (CVaR)

$$\mathrm{CVaR}_{1-\epsilon}(Z)=\inf_{\tau\in\Re}[\tau+\frac{1}{\epsilon}\mathbb{E}[Z-\tau]_+]. \tag{4.5.10}$$

$\mathrm{CVaR}_{1-\epsilon}(Z)$ is a convex and monotone functional on the space of random variables with finite first moment, and the $(1-\epsilon)$ quantile (the value at risk)

$$\mathrm{VaR}_{1-\epsilon}(Z)=\inf_{t\in\Re}[\mathcal{P}\{Z\leq t\}\geq 1-\epsilon].$$

of the distribution of Z is a minimizer of the right-hand side in (4.5.10), so that it always holds that $\mathrm{CVaR}_{1-\epsilon}(Z)\geq\mathrm{VaR}_{1-\epsilon}(Z)$. Since the chance constraint in (4.5.1) is nothing but $\mathrm{VaR}_{1-\epsilon}[F(x,\xi)]\leq 0$, the constraint

$$\mathrm{CVaR}_{1-\epsilon}[F(x,\xi)]\leq 0 \tag{4.5.11}$$

defines a convex conservative approximation of the chance constraint. The idea of using CVaR as a convex approximation of VaR comes from Rockafellar and Uryasev [94].

One of the possible disadvantages of using the "optimal" generating function ψ^* (as compared with the exponential $\psi(z)=e^z$, which will be used in the Bernstein approximation) in the above approximation scheme is that it is unclear how to compute efficiently the corresponding function $\Psi(x,t)$ even in the simple case $F(x,\xi)=f_0(x)+\sum_{j=1}^{n_\xi}\xi_j f_j(x)$ of affine in ξ function $F(x,\xi)$ and independent-of-each-other random variables ξ_j with known and simple distributions.

There are several ways how the presented construction can be extended for $m>1$ (joined chance constraints). One simple way is to replace the constraints $f_i(x,\xi)\leq 0, i=1,\ldots,m$, with one constraint $f(x,\xi)\leq 0$, say by taking $f(x,\xi)=\max\{f_1(x,\xi),\ldots,f_m(x,\xi)\}$. Note, however, that this may destroy a simple, e.g., affine in ξ, structure of the constraint mapping $F(x,\xi)$, rendering the Bernstein approximation unusable.

There is another possible extension of the above approximation scheme for $m>1$. Let $\epsilon_1,\ldots,\epsilon_m$ be positive numbers such that $\epsilon_1+\cdots+\epsilon_m\leq\epsilon$. The chance constraint of (4.5.1) is equivalent to $\mathcal{P}\{\cup_{i=1}^m\{\xi: f_i(x,\xi)>0\}\}<\epsilon$. Since

$$\mathcal{P}\{\cup_{i=1}^m\{\xi: f_i(x,\xi)>0\}\}\leq\sum_{i=1}^m\mathcal{P}\{f_i(x,\xi)>0\},$$

it follows that the system of constraints

$$\mathcal{P}\{f_i(x,\xi)>0\}\leq\epsilon_i,\quad i=1,\ldots,m, \tag{4.5.12}$$

is more conservative then the original chance constraint. We can now apply the one-dimensional construction to each individual constraint of (4.5.12) to obtain the following

convex conservative approximation of the chance constrained problem (4.5.1):

$$\begin{array}{ll} \underset{x\in\mathcal{X},t>0}{\text{minimize}} & c^T x \\ \text{subject to} & \inf_{t>0}[\Psi_i(x,t) - t\epsilon_i] \le 0, \quad i = 1,\dots,m, \end{array} \tag{4.5.13}$$

where $\Psi_i(x,t) = t\mathbb{E}[\psi_i(t^{-1}f_i(x,\xi))]$, and each $\psi_i(\cdot), i = 1,\dots,m$ is a one-dimensional generating function.

Remark 4.5.1 *An important open question related to the approximation (4.5.13) is how to choose ϵ_i. It would be very attractive to treat these quantities in (4.5.13) as design variables (subject to the constraints $\epsilon_i > 0$ and $\sum_i \epsilon_i \le \epsilon$) rather than as parameters. Unfortunately, such an attempt destroys the convexity of (4.5.13) and thus makes the approximation seemingly intractable. The simplest way to resolve the issue in question is to set $\epsilon_i = \epsilon/m, i = 1,\dots,m$.*

4.5.2 | Bernstein Approximation

One of the downsides of using the piecewise linear generating functions of the form (4.5.7) is that the corresponding constraint function may be difficult to compute even for relatively simple functions $F(x,\xi)$. For this reason we consider the (one-dimensional) generating function $\psi(z) = e^z$. The assumptions are the following:

A1. The components $f_i(x,\xi)$ in the constraint mapping $F(x,\xi)$ are affine in ξ:

$$f_i(x,\xi) = f_{i0}(x) + \sum_{j=1}^{n_\xi} \xi_j f_{ij}(x), \quad i = 1,\dots,m, \tag{4.5.14}$$

and the functions $f_{ij}(x), j = 0,1,...,n_\xi$ are well defined and convex on $\mathcal{X}$. Besides this, for every $j \ge 1$ such that $\Xi_j \not\subset \Re_+$, all functions $f_{ij}(x), i = 1,\dots,m$ are affine.

A2. The components $\xi_j, j = 1,\dots,n_\xi$, of the random vector ξ are independent of other random variables (so that the support of the distribution $\mathcal{P}$ of ξ is $\Xi = \Xi_1 \times \cdots \times \Xi_{n_\xi}$). Furthermore, let

$$M_j(t) = \mathbb{E}[e^{t\xi_j}] = \int \exp(tz) dP_j(z),$$

be the moment generating function, and $\Lambda_j(t) = \log M_j(t)$ be the logarithmic moment generating function of ξ_j.

A3. The moment generating functions $M_j(t), j = 1,\dots,n_\xi$, are finite valued for all $t \in \Re$ and are efficiently computable.

The problem (4.5.1) that satisfies the assumptions A1-A3 is called an affinely perturbed convex chance constrained problem. Let $z = (z_0, z_1, \dots, z_{n_\xi}) \in \Re^{n_\xi+1}$. The function

$$\Phi(z) = \log(\mathbb{E}[\exp\{z_0 + \sum_{j=1}^{n_\xi} \xi_j z_j\}]) = z_0 + \sum_{j=1}^{n_\xi} \Lambda_j(z_j)$$

is well defined, continuous in z and convex. Moreover, $\Phi(z)$ is monotone in z_0 and in every z_j with $j \in J = \{j \ge 1 : \Xi_j \subset \Re_+\}$. Finally, one has for $t > 0$ and $p(z) =$

$\mathcal{P}\{z_0 + \sum_{j=1}^{n_\xi} \xi_j z_j > 0\}$ that

$$\Phi(t^{-1}z) \geq \log p(z).$$

Therefore, for every $\epsilon \in (0,1)$,

$$\inf_{t>0}[t\Phi(t^{-1}z) - t\log\epsilon] \leq 0 \quad \text{implies} \quad p(z) \leq \epsilon. \tag{4.5.15}$$

Now consider an affine chance constrained problem with real-valued constraint mapping

$$F(x,\xi) = f_0(x) + \sum_{j=1}^{n_\xi} \xi_j f_j(x).$$

By (4.5.15), the problem

$$\begin{array}{ll} \underset{x\in\mathcal{X}}{\text{minimize}} & c^T x \\ \text{subject to} & \inf_{t>0}[f_0(x) + \sum_{j=1}^{n_\xi} t\Lambda_j(t^{-1}f_j(x)) - t\log\epsilon_i] \leq 0 \end{array} \tag{4.5.16}$$

is a conservative approximation of the chance constrained problem (4.5.1). Furthermore, this approximation is convex.

4.5.3 | Numerical Examination

Just as in [82], the Bernstein approximation will be compared with the (now enhanced) scenario approach, on the same example as in the aforementioned paper.

The test problem is (once again) optimizing value at risk. There are $n+1$ assets $0, 1, \ldots, n$ with random returns. The problem is to distribute \$1 between the assets in order to maximize the upper ϵth quantile of the profit. The corresponding model is the chance constrained linear programming problem

$$\begin{array}{ll} \underset{x\geq 0, t\in\Re}{\text{minimize}} & -t \\ \text{subject to} & \mathcal{P}\{t \leq \sum_{i=0}^n r_i x_i\} \leq \epsilon, \\ & \sum_{i=0}^n x_i \leq 1, \end{array} \tag{4.5.17}$$

where x_i is the capital invested in asset i, and r_i is the return of this asset. The data in this problem are as follows:

- There are $n+1 = 65$ assets; asset #0 ("money") has deterministic return $r_0 = 1$, while the returns r_i of the remaining 64 "true" assets are random variables with expectations $\mathbb{E}[r_i] = 1 + \rho_i$, with nominal profits ρ_i varying in $[0, 0.1]$ and growing with i. The spacing between the individual values ρ_i is not explicitly stated in the paper [82]; we used even spacing.
- The random variables $r_i, 1 \leq i \leq 64$, are of the form

  $$r_i = \eta_i + \sum_{l=1}^{8} \gamma_{il}\zeta_l,$$

  where $\eta_i \sim \mathcal{LN}(\mu_i, \sigma_i^2)$(that is, $\log\eta_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$) is the individual noise in ith return, $\zeta_l \sim \mathcal{LN}(\nu_l, \theta_l^2)$ are the "common factors" affecting all returns, and $\gamma_{il} \geq 0$

are deterministic "influence coefficients". All "primitive" random variables (64 of η_i's and 8 of ζ_l's) are independent of each other.

We used $\nu_l = 0, \theta_l = 0.1, \mu_i = \sigma_i$ (that is, the more promising an asset at average, the more risky it is). The influence coefficients γ_{il} and the parameters μ_i were chosen in a such a way that $\mathbb{E}[\sum_{l=1}^{8} \gamma_{il}\zeta_l] = \rho_i/2$ and $\mathbb{E}[\eta_i] = 1 + \rho_i/2$ for all i. Since there are multiple ways to choose γ_{il} (and the paper [82] does not give an explicit formula), we chose $\gamma_{il} = \gamma_i$, for all l (with γ_i appropriately chosen for each i).

The random returns r_i are linear combinations of independent random variables $\eta_1, \ldots, \eta_{64}$, $\zeta_1, \ldots, \zeta_8$, so that the structure of (4.5.17) allows for applying Bernstein approximation. The difficulty, however, is that the random variables in question are log-normal and thus the corresponding moment generating functions are $+\infty$ outside of the origin. This difficulty can be circumvented, by using a discretization (thoroughly described in [82]). The resulting problem has the following structure:

$$\begin{array}{ll} \underset{x \geq 0, t \in \Re}{\text{minimize}} & -t \\ \text{subject to} & \inf_{\tau>0}[t + \sum_{i=0}^{64} \tau \log(a_i \exp\{-b_i x_i/\tau\} + \\ & \qquad \sum_{j=1}^{m_i} c_{ij} \exp\{-d_{ij} x_i/\tau\}) - \tau \log \varepsilon] \leq 0, \\ & \sum_{i=0}^{64} x_i \leq 1, \end{array} \tag{4.5.18}$$

where constants a_i, b_i, c_{ij}, d_{ij} come from the discretization, which has m_i points for each r_i (around 400 in our implementation). The implementation of the Bernstein approximation was programmed in `Julia` [68] using the modelling enviroment `JuMP` [28] and the `MOSEK` solver [79]. As described in [82], for best results the Bernstein approximation should be tuned – the conservatism of the approximation can be relaxed by changing (increasing) ε and using simulation to estimate the reliability of the chance constraint. The simulation evaluating the solutions for different approaches (values of ε in Bernstein approximation and different number of considered and deleted scenarios for the other approach) had 10^6 scenarios.

The first target quantile for the maximization was chosen as $\epsilon = 0.01$. The results are best described by the following Tables 4.4 and 4.5, Figure 4.6 and Figure 4.7.

ε	t^*	$F(t^*)$	$F^{-1}(0.01)$
0.01	1.0530	$2.19{\cdot}10^{-4}$	1.0621
0.02	1.0557	$7.08{\cdot}10^{-4}$	1.0626
0.03	1.0575	0.0014	1.0629
0.05	1.0598	0.0032	1.0632
0.10	1.0634	0.0098	1.0636

Table 4.4: Results for the Bernstein approximation, target quantile $\epsilon = 0.01$.

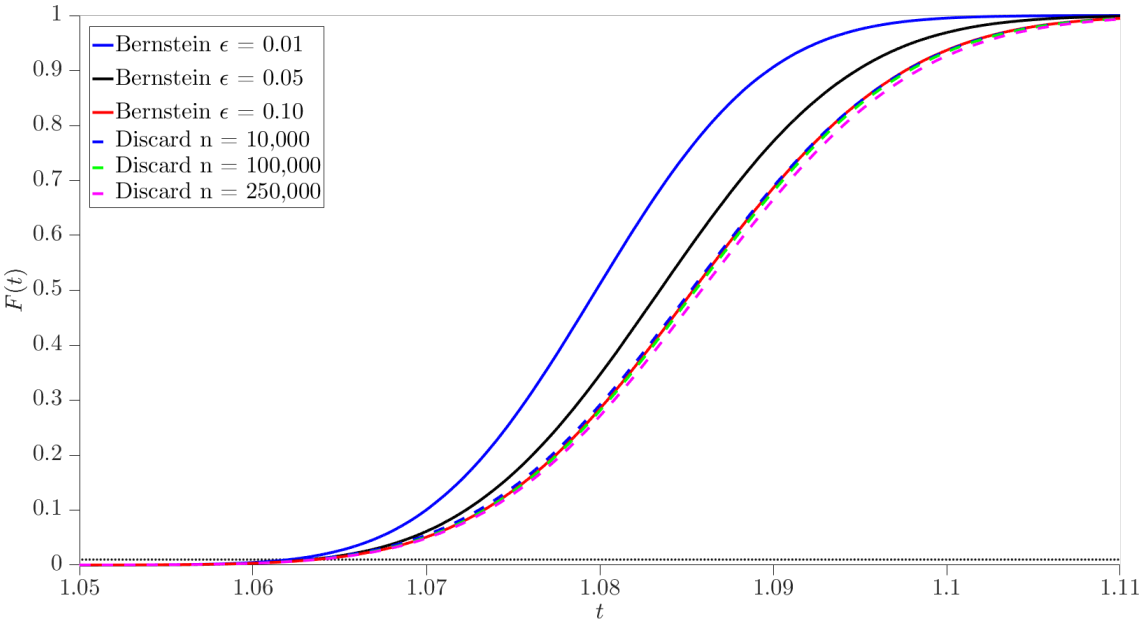


Fig. 4.6: Cumulative distribution functions of returns for different approaches, target quantile $\epsilon = 0.01$.

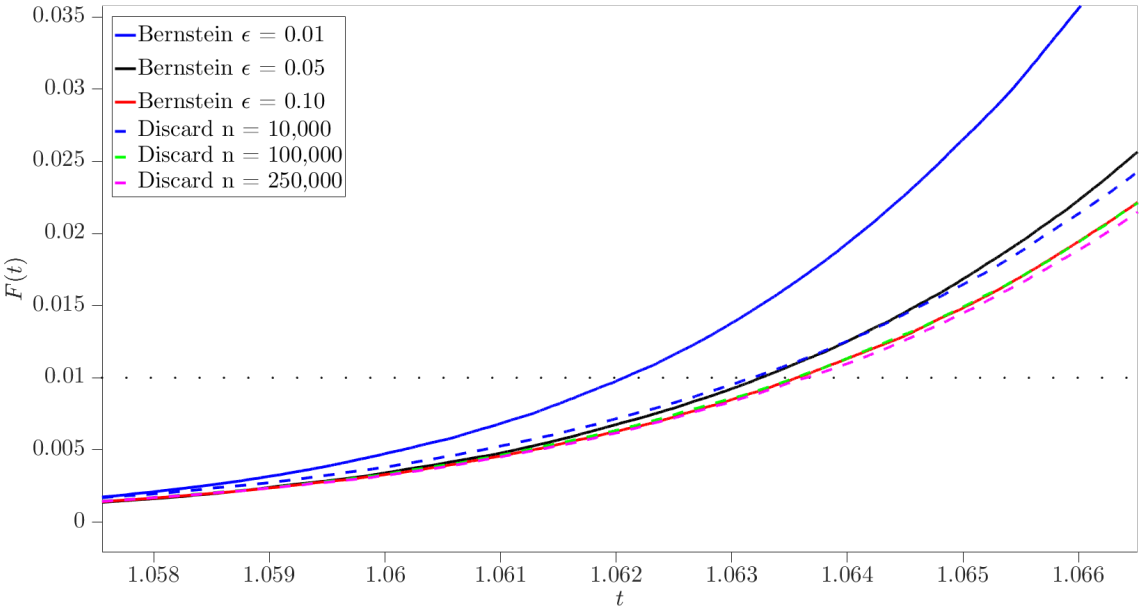


Fig. 4.7: Closer look at the target value.

number of scenarios	discarded	t^*	$F(t^*)$	$F^{-1}(0.01)$
10,000	62	1.0629	0.0094	1.0632
50,000	467	1.0635	0.01	1.0636
100,000	988	1.0636	0.01	1.0636
250,000	2,424	1.0636	0.01	1.0636

Table 4.5: Results for P&D, target quantile $\epsilon = 0.01$.

The second target quantile for the maximization was chosen as $\epsilon = 0.05$. The results are best described by the following Tables 4.6 and 4.7, Figure 4.8 and Figure 4.9.

ε	t^*	$F(t^*)$	$F^{-1}(0.05)$
0.05	1.0598	0.0032	1.0691
0.10	1.0634	0.0096	1.0699
0.15	1.0658	0.0177	1.0703
0.20	1.0677	0.0274	1.0706
0.25	1.0693	0.0380	1.0707
0.29	1.0705	0.0476	1.0708

Table 4.6: Results for the Bernstein approximation, target quantile $\epsilon = 0.05$.

number of scenarios	discarded	t^*	$F(t^*)$	$F^{-1}(0.05)$
10,000	468	1.0703	0.05	1.0703
50,000	2,487	1.0707	0.05	1.0707
100,000	4,939	1.0707	0.05	1.0707

Table 4.7: Results for P&D, target quantile $\epsilon = 0.05$.

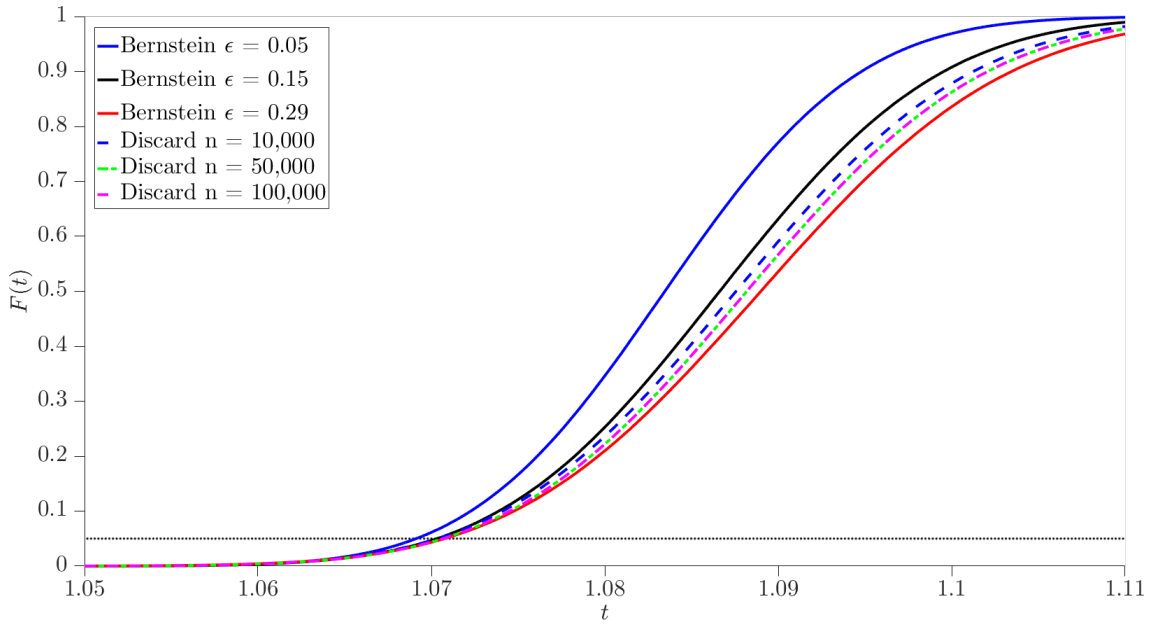


Fig. 4.8: Cumulative distribution functions of returns for different approaches, target quantile $\epsilon = 0.05$.

For all practical purposes, the best objective values for both the Bernstein approximation and the P&D algorithm are the same. Although one would prefer the decision obtained by P&D with 250,000 in the first case (Figure 4.6) and the decision obtained by the Bernstein approximation with $\varepsilon = 0.29$ in the second case (Figure 4.9) (because of the "dominating" nature of the returns obtained by these decisions), since this was not a part of the objective, we will not examine it any further.

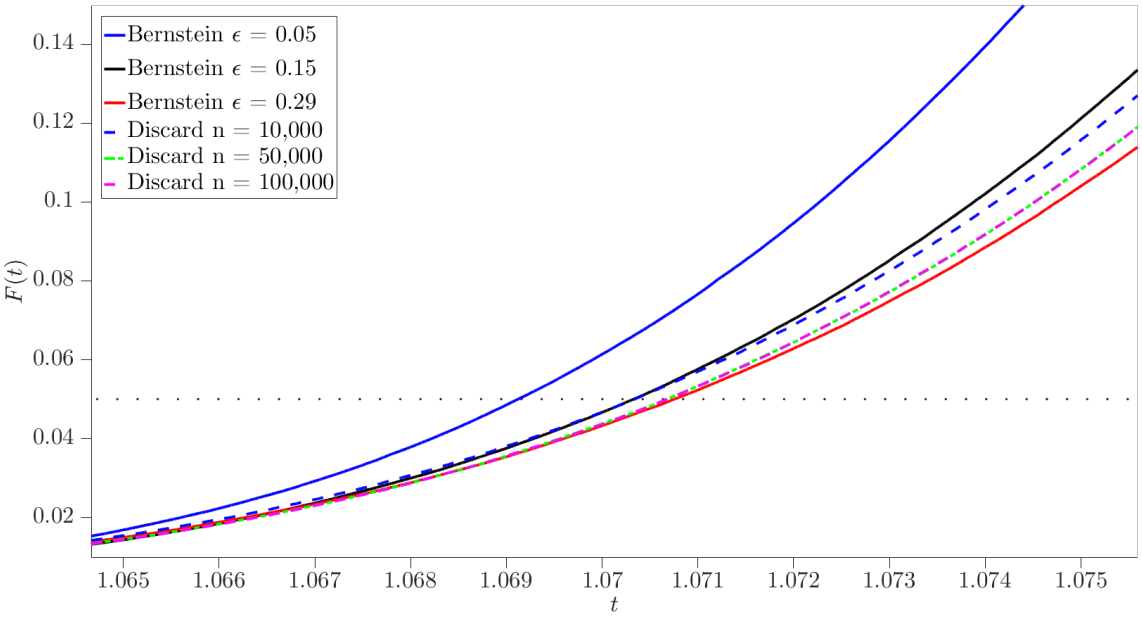


Fig. 4.9: Closer look at the target value.

The computational times are, however, rather incomparable. While the Bernstein approximation takes about 0.5 s to compute (for each value of ε), the time it takes to terminate the P&D algorithm grows in the manner described in the previous sections (it takes over an hour for the maximum number of scenarios). The important thing to bear in mind is that the example above has a single chance constraint. In the case of several joint chance constraints, the Bernstein approximation reformulates the problem as one with separate chance constraints (4.5.12) each having different reliability ϵ_i. The task of tuning this approximation (choosing the values ϵ_i) turns to problem into a global optimization one (convexity is lost), and gets (much) harder to solve, when one has to deal with increasing number of (joint) chance constraints. On the other hand, the P&D algorithm does not change at all and retains its properties (convexity), even when moving from the single to the joint chance constraint setting.

4.6 | Nonlinear Example

In this section we investigate the performance of the algorithm on nonlinear example that appeared in the numerical sections of the state-of-the-art methods in [100] and [1]. Both of these methods are scenarios (or sample) based and use the indicator function approximation (although they approach it in different ways). In the method described in [100], the constraints need to be convex in x and the problem can be a joint chance constrained one. In the method described in [1], the constraints do not have to be convex, but must be continuously differentiable in x and the authors deal with a single chance constraint only. The problem both papers have chosen for the numerical examination is

the following one:

$$\begin{array}{ll} \underset{x \geq 0}{\text{minimize}} & -\sum_{j=1}^{n} x_j \\ \text{subject to} & \mathcal{P}\{\sum_{j=1}^{n} \xi_{ij}^2 x_j^2 - b \leq 0,\ i = 1, \ldots, m\} \geq 1 - \epsilon, \end{array} \tag{4.6.1}$$

where $\xi_{ij}, i = 1, \ldots, m$ and $j = 1, \ldots, n$ are independent and identically distributed standard normal random variables, $b \in \Re$. In the case of [1], $m = 1$.

Optimal solution x^* of the problem (4.6.1), derived in [100], is:

$$x_1^* = x_2^* = \cdots = x_n^* = \left[b / F_{\chi_n^2}^{-1}((1-\epsilon)^{\frac{1}{m}}) \right]^{\frac{1}{2}}, \tag{4.6.2}$$

where $F_{\chi_n^2}^{-1}$ is the inverse chi-squared distribution function with n degrees of freedom.

Since the problem (4.6.2) is quadratic, we can use the `CPLEX` solver [49] in `JuMP` in the implementation of P&D. The non-default parameter values were `CPX_PARAM_BAREPCOMP` $= 10^{-8}$ (the tolerance on complementarity for convergence), `CPX_PARAM_EPOPT` $= 10^{-8}$ (the reduced-cost tolerance for optimality), `CPX_PARAM_EPRHS` $= 10^{-8}$ (the feasibility tolerance) and $\delta = 10^{-7}$ (the P&D tolerance, the same was used for the linearized version).

We start the numerical examination with the same setting as [1]: $n = 10, m = 1, b = 10, \epsilon = 0.05$. Using the formula (4.6.2), the optimal objective value of this problem is -7.390. We generate a number of scenarios S (the values were log-spaced between 10^2 and 10^4) and set the P&D algorithm to discard $\lfloor \epsilon S \rfloor$ of them. After that we estimate the reliability of the obtained solution using 10^5 new scenarios. The results of the computations are summarized in Table 4.8 and in Figure 4.10.

scenarios	objective	reliability	time [s]
100	-8.042	0.8535	0.44
167	-7.968	0.8700	1.08
278	-7.736	0.9039	1.87
464	-7.564	0.9255	4.17
774	-7.528	0.9322	7.94
1,292	-7.539	0.9356	14.80
2,154	-7.467	0.9422	27.53
3,594	-7.440	0.9454	51.09
5,995	-7.400	0.9491	92.14
10,000	-7.397	0.9497	174.75

Table 4.8: Number of scenarios, optimal objective value, reliability of the solution and computation time. Average values over 10 runs.

Unsurprisingly, the more scenarios are taken into account, the better (closer to the theoretical optimum) the result. The computational time is quite good, considering [1] report around 500 s as the computational time for their algorithm (that uses 500 scenarios) and the big-M mixed-integer formulation does not converge in an hour [1] (again, using "just" 500 scenarios).

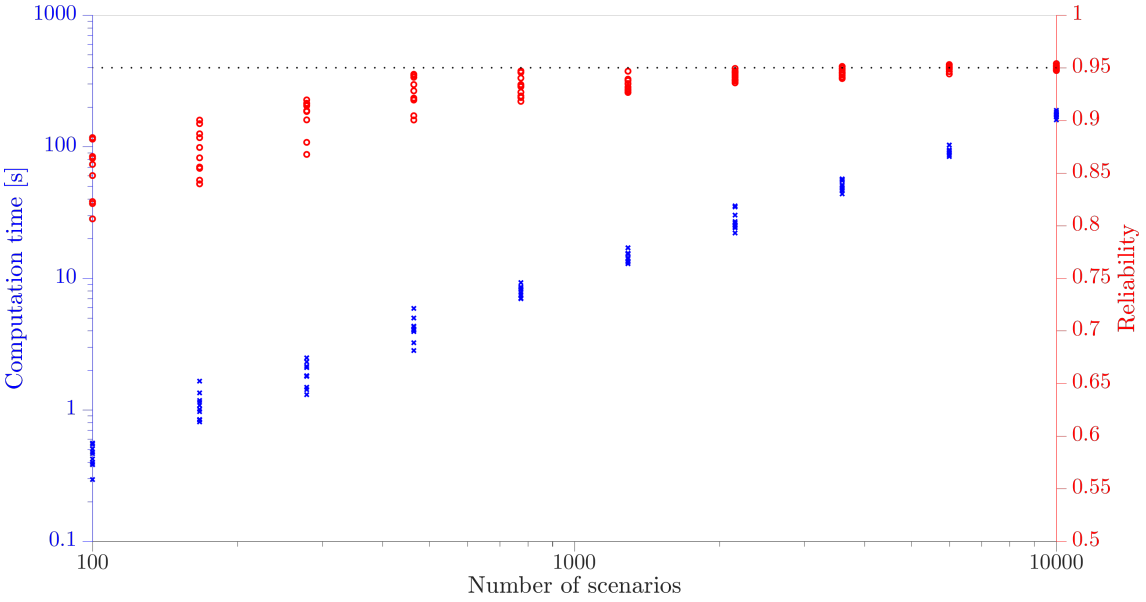


Fig. 4.10: Dependence of reliability of the solution and computation time of the P&D on the number of scenarios.

n	scenarios	P&D [s]	P&D Lin. [s]	obj. dif.	reliab. dif.
5	100	0.17	0.30	$1.2{\cdot}10^{-6}$	$2.0{\cdot}10^{-6}$
	316	0.57	0.63	$1.3{\cdot}10^{-6}$	$1.0{\cdot}10^{-6}$
	1,000	1.85	1.64	$8.1{\cdot}10^{-7}$	$2.0{\cdot}10^{-6}$
	3,162	6.07	4.57	$1.1{\cdot}10^{-6}$	0
	10,000	22.71	16.10	$1.2{\cdot}10^{-6}$	0
10	100	0.55	2.16	$3.4{\cdot}10^{-6}$	$2.0{\cdot}10^{-6}$
	316	2.45	5.36	$3.1{\cdot}10^{-6}$	0
	1,000	11.91	12.82	$3.2{\cdot}10^{-6}$	$2.0{\cdot}10^{-6}$
	3,162	46.93	28.46	$8.9{\cdot}10^{-4}$	$6.8{\cdot}10^{-5}$
	10,000	180.67	93.23	$1.8{\cdot}10^{-3}$	$1.8{\cdot}10^{-4}$
20	100	4.14	43.30	$1.3{\cdot}10^{-5}$	$6.0{\cdot}10^{-6}$
	158	8.22	70.24	$5.1{\cdot}10^{-6}$	$9.0{\cdot}10^{-6}$
	316	22.36	144.31	$8.1{\cdot}10^{-6}$	$3.0{\cdot}10^{-6}$
	501	49.93	222.23	$7.0{\cdot}10^{-6}$	$6.0{\cdot}10^{-6}$
	1,000	143.27	400.81	$7.7{\cdot}10^{-6}$	$3.0{\cdot}10^{-6}$
	2,000	398.01	619.38	$1.2{\cdot}10^{-5}$	$6.0{\cdot}10^{-6}$
	5,000	739.82	1,045.96	$4.5{\cdot}10^{-4}$	$1.1{\cdot}10^{-4}$

Table 4.9: Computational times for the P&D and linearized P&D, absolute differences in the optimal objective function values and absolute differences in reliability. Average values over 10 runs.

Next, we investigate the dependence of the computational time on n and compare the P&D algorithm with its linearized version. In the Table 4.9, we report the computation times for both of these methods and the absolute differences in the optimal objective

value and reliability of the solution. A graphical comparison is provided by Figure 4.11. The differences in the objective and reliability are rather negligible – on the order of $10^{-4} - 10^{-6}$. The computational time was better for the normal version of P&D most of the time. Only when the number of variables n was low and the number of scenarios S was high, performed the linearized version better. Another way to say this is that the linearized version scales better with the number of scenarios (the resulting linear problem is easier to solve than the quadratic one), but scales much worse in the number of variables n – possibly because linear approximations in higher dimension grow costly very rapidly.

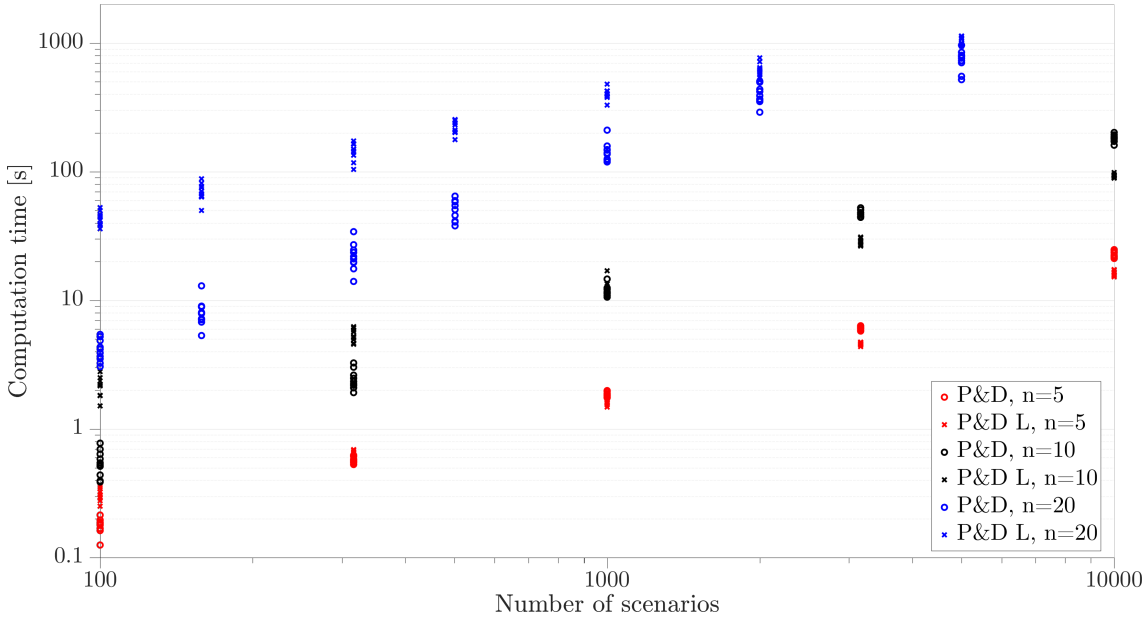


Fig. 4.11: Dependence of computational time on number of scenarios and number of variables for P&D and linearized P&D.

The second setting we investigate is from [100]: $n = 10, m = 10, b = 100, \epsilon = 0.1$. The optimal objective value, using (4.6.2), is -20.82. Note that in this setting we are dealing with a "proper" joint chance constraint problem, with nonlinear (but convex) constraint functions. The number of scenarios used in the examination ranged between 10^2 and 10^4 and the number of scenarios to discard was set to $\lfloor \epsilon S \rfloor$. The results of the computations are listed in the following Table 4.10:

scenarios	objective	reliability	time [s]
100	-21.46	0.8042	1.63
251	-21.42	0.8305	5.60
631	-21.03	0.8720	18.02
1,585	-20.96	0.8853	55.40
3,981	-20.84	0.8962	166.14
10,000	-20.83	0.8975	559.82

Table 4.10: Number of scenarios, optimal objective value, reliability of the solution and computational time. Average values over 10 runs.

To compare the results with the ones achieved in [100], where they used 10,000 scenarios for the computations – the numbers the authors report are a bit vague:
"Our algorithm typically requires less than 10 iterations to converge to the optimal value, and each iteration approximately takes 6 s on average."
The main objections being that their algorithm was presented on two problems with different dimensions (the one presented here and a smaller one), and that, at least judging from the figures (as there is no other way to find the value), their "optimal solution" was around -20.4, which is rather far from the real one.

An important remark is that the P&D algorithm does not produce just one solution – as a sort of a by-product it generates a sequence of decisions, that are "optimal" with respect to an increasing number of discarded scenarios. When we estimate the reliability of these solutions, we get an approximation of the trade-off between the reliability level ϵ and optimal objective value. Naturally, this approximation gets better as we increase the number of scenarios. The approximation of the trade-off for the setting described above is depicted in Figure 4.12 – it shows just one run of the P&D algorithm for different number of scenarios and the optimal values computed using the formula (4.6.2). The reliability of the solution is estimated using 10^5 different scenarios. If we stopped the algorithm with 200 scenarios once the estimate of the reliability of the solution gets over the desired level ϵ and use the previous value, we would discard only 12 scenarios with the objective value -20.47 and estimated reliability 0.9143 – this takes around 3 s. Note that the robust solution for this problem, i.e. the solution for $\epsilon = 0$, is clearly 0, since each ξ_{ij}^2 can attain any nonnegative value.

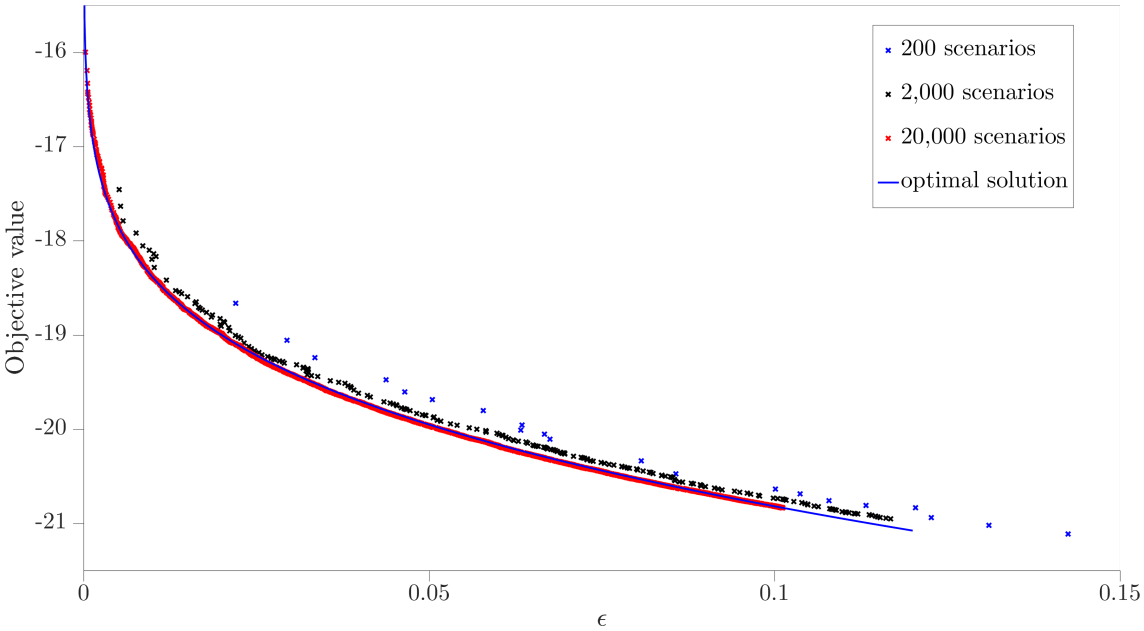


Fig. 4.12: Approximation of trade-off between reliability and optimal objective value.

CHAPTER 5

Chance Constrained Optimal Beam Design: Convex Reformulation and Probabilistic Robust Design

This is the paper that will appear in [58]. It describes a novel (convex) reformulation of otherwise rather involved engineering problem. It can be also seen as the first application of the P&D algorithm with a trivial Pooling part (see the transformation from (5.4.6)-(5.4.7) to (5.4.10)-(5.4.11)). The subsection 5.5.1 was originally not included in the final paper (mainly because of a space restriction) – we decided to add it for its additional insight.

5.1 | Introduction

Optimal design problems in engineering frequently lead to optimization problems involving differential equations. One of the classes of these problems is shape optimization [42]. The particular shape optimization problem considered in this paper is the optimal design of a beam (be it a fixed beam, a cantilever beam, etc.) subjected to some kind of loading. Since shape optimization problems are inherently non-convex, most approaches use metaheuristics such as genetic algorithms [65] or cuckoo search [36]. A closely related field of topology optimization (where the size and shape of the structure can be manipulated) has developed a multitude of successful methods (level set, homogenization, topological derivative, etc.), see [96].

This problem was previously also examined in [99] and [115], where the authors used the finite element method (FEM) and the finite difference method to approximate the ordinary differential equations (ODE) and solve the problem. Our paper shows that this beam design problem can be formulated as a geometric programming problem, which can be further transformed into a convex one, and thus can be efficiently solved. Geometric programming problems with random coefficients (although without chance constraints)

were investigated in [30].

An important issue regarding the design is its reliability (see [62]). In the context of this paper the reliability of the design will mean that the constraints in the resulting optimization program should hold with high probability. Depending on how reliable design is required, we can destinguish between the so-called chance constrained (or probabilistic constrained) optimization problems (see [98]) and the robust optimization problems (see [9]). Current approaches dealing with reliability constrained beam design, such as [8] and [116] use simple (point) loads and Gaussian distribution of the unknown parameters. In this paper we investigate the chance constrained beam design problem under more complicated random loads. We utilize the sampling approach (called Probabilistic Robust Design) developed in [19], [21] and [22] to obtain a manageable approximation of the chance constrained problem and use a scenario-deletion method to compute a trade-off between the reliability of the design and the objective value.

5.2 | Problem Formulation

The problem is best described by Fig. 5.1. We consider a fixed beam of length l with rectangular cross-section that is subjected to a load $h(x)$ (with the opposite direction than the axis y), which is depicted in Fig. 5.1a. The task is to find the optimal design, in terms of the cross-section dimensions a and b (Fig. 5.1b), that minimizes the weight of the beam.

Naturally, given a load $h(x)$ the beam will deflect and will be subjected to a bending stress. The requirement for the design is that the maximum stress in the beam is less then a material-specific constant, that ensures that the design is safe (we use the value at which the material begins to deform plastically). The problem can be formulated as the following ODE-constrained optimization program:

$$\underset{a,b,v(x)}{\text{minimize}} \quad \rho abl \tag{5.2.1}$$

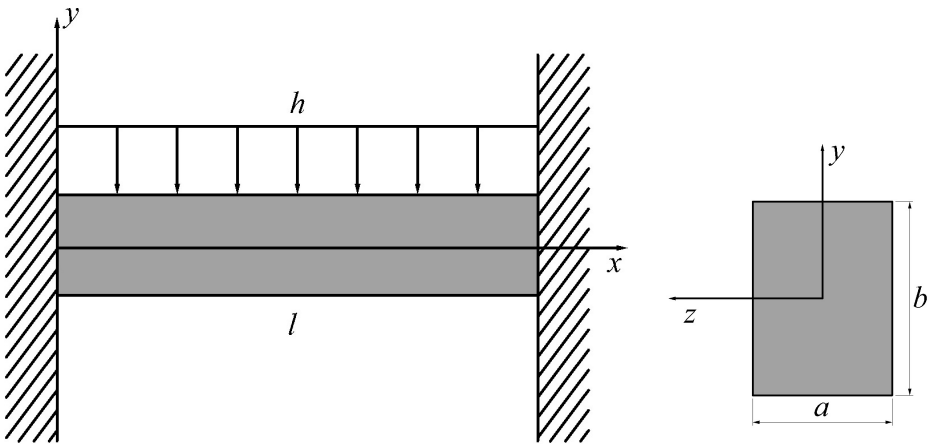
$$\text{subject to} \quad E\frac{ab^3}{12}\frac{d^4v}{dx^4}(x) = h(x), \qquad x \in [0,l], \tag{5.2.2}$$

$$|E\frac{b}{2}\frac{d^2v}{dx^2}(x)| \leq \sigma_M, \qquad x \in [0,l], \tag{5.2.3}$$

$$v(0) = 0, \;\; \frac{dv}{dx}(0) = 0, \;\; v(l) = 0, \;\; \frac{dv}{dx}(l) = 0, \tag{5.2.4}$$

$$a_L \leq a \leq a_U, \quad b_L \leq b \leq b_U, \tag{5.2.5}$$

where ρ is the density of the material, $v(x)$ is the deflection of the beam (with the opposite direction than the axis y) in a point $x \in [0,l]$, E is the Young modulus, σ_M is the maximum stress allowed, and a_L, a_U, b_L, b_U are the bounds on the cross-section dimensions. The constraint (5.2.2) is the ODE that governs the deflection of the beam $v(x)$ given a specific load $h(x)$. The constraint (5.2.3) is the maximum allowed stress in the beam. The constraint (5.2.4) defines the boundary conditions for the ODE (i.e. that we have a fixed beam).



(a) The scheme of loaded beam. **(b)** Beam cross-section.

Fig. 5.1: The problem geometry.

5.2.1 | FEM Problem Approximation and Solution

To tackle the problem (5.2.1)-(5.2.5) we use the FEM to approximate the ODE in (5.2.2) and (5.2.3). Following [103] (p. 25 – 27), we divide the one-dimensional beam with the space dimension x into N finite elements. We will denote the nodal value of the deflection $v(x)$ in the node x_e as $V_e = v(x_e)$ and the nodal value of its derivative in the same node as $\theta_e = \frac{dv}{dx}(x_e)$. The continuous variable v is approximated by $\tilde{v}$ in terms of nodal values as follows:

$$\tilde{v}_e = [N_1\, N_2\, N_3\, N_4][V_{e-1}\, \theta_{e-1}\, V_e\, \theta_e]^T$$

where $N_1, \ldots, N_4$ are the following cubic shape functions:

$$N_1 = \frac{1}{d^3}(d^3 - 3dx^2 + 2x^3), \qquad N_2 = \frac{1}{d^2}(d^2x - 2dx^2 + x^3),$$
$$N_3 = \frac{1}{d^3}(3dx^2 - 2x^3), \qquad N_4 = \frac{1}{d^2}(x^3 - dx^2),$$

and $d = \frac{l}{N}$ is the length of one element. Substitution in (5.2.2) and application of Galerkin's method lead to four element equations:

$$\int_0^d \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} E\frac{ab^3}{12}\frac{d^4}{dx^4}[N_1\, N_2\, N_3\, N_4]\, dx \begin{bmatrix} V_{e-1} \\ \theta_{e-1} \\ V_e \\ \theta_e \end{bmatrix} = \int_0^d \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} h(x)\, dx.$$

To avoid differentiating four times, the following approximation is used: $\int N_i \frac{d^4N_j}{dx^4}dx \approx -\int \frac{dN_i}{dx}\frac{d^3N_j}{dx^3}dx \approx \int \frac{d^2N_i}{dx^2}\frac{d^2N_j}{dx^2}dx$. The resulting system of linear equations has the form: $E\frac{ab^3}{12}\mathbb{K}\mathcal{V} = h$, where $\mathcal{V} = (V_0, \theta_0, \ldots, V_N, \theta_N)^T$. The dimensions of the stiffness matrix $\mathbb{K}$ are $(2N+2)\times(2N+2)$ and its precise description can be found in [99] or [103]. The order of accuracy of the finite element approximation is $\mathcal{O}(d^2)$.

Using this approximation of the deflection, the stress limit (5.2.3) on each element is given by

$$|E\frac{b}{2}[N_1'' \; N_2'' \; N_3'' \; N_4'']|[V_{e-1} \, \theta_{e-1} \, V_e \, \theta_e]^T| \leq \sigma_M.$$

This equation describing the stress in one specific node holds only for the end nodes belonging to one element (the first one at x_0 and the last one at x_N). Since the rest of the nodes belongs to two adjacent elements, the stresses are not equal. Therefore, we consider the average stress from this discontinuity:

$$E\frac{b}{2}\frac{1}{2}\left|[N_1''(0) \, N_2''(0) \, N_3''(0) \, N_4''(0)] \begin{bmatrix} V_{e-1} \\ \theta_{e-1} \\ V_e \\ \theta_e \end{bmatrix} + [N_1''(d) \, N_2''(d) \, N_3''(d) \, N_4''(d)] \begin{bmatrix} V_e \\ \theta_e \\ V_{e+1} \\ \theta_{e+1} \end{bmatrix}\right|.$$

The system of inequalities that approximates (5.2.4) can be written as $|E\frac{b}{2}\mathbb{C}\mathcal{V}| \leq \sigma_M$, where the matrix $\mathbb{C}$ has dimensions $(N+1)\times(2N+2)$ and its complete description can be found in [99].

The FEM approximation of the problem (5.2.1)-(5.2.5) is then the following (using the notation described above):

$$\begin{aligned} \underset{a,b,\mathcal{V}}{\text{minimize}} \quad & \rho abl && (5.2.6)\\ \text{subject to} \quad & E\frac{ab^3}{12}\mathbb{K}\mathcal{V} = h, && (5.2.7)\\ & |E\frac{b}{2}\mathbb{C}\mathcal{V}| \leq \sigma_M, && (5.2.8)\\ & a_L \leq a \leq a_U, \quad b_L \leq b \leq b_U. && (5.2.9) \end{aligned}$$

This problem has $2N$ variables ($2N+2$ in $\mathcal{V}$ of which 4 are fixed by boundary conditions, and 2 design variables), $2N+2$ constraints and a box constraints on a and b, and is non-convex, meaning that the certification of global optimality is computationally very demanding.

The crucial realization (the one that is absent in [99] and [115]) is that the stiffness matrix $\mathbb{K}$ is, by design, always invertible. In other words – given a, b and h, the equation describing the deflection of the beam has a unique solution. Using this fact, we can rewrite (5.2.7) as:

$$\mathcal{V} = \frac{12}{Eab^3}\mathbb{K}^{-1}h, \tag{5.2.10}$$

and (5.2.8) becomes:

$$|\frac{6}{ab^2}\mathbb{C}\mathbb{K}^{-1}h| = \frac{1}{ab^2}|6\mathbb{C}\mathbb{K}^{-1}h| \leq \sigma_M. \tag{5.2.11}$$

Let us denote as v_M the maximum of $|6\mathbb{C}\mathbb{K}^{-1}h|$ over all the nodes of the FEM discretization. Since σ_M is the same for all $N+1$ nodes, the $N+1$ inequalities (5.2.8) are equivalent to a single inequality:

$$\frac{v_M}{ab^2} \leq \sigma_M. \tag{5.2.12}$$

Utilizing these results and neglecting the constants ρ and l in the objective (5.2.6), we can reformulate the problem (5.2.6)-(5.2.9) as the following equivalent problem:

$$\underset{a,b}{\text{minimize}} \quad ab \tag{5.2.13}$$

$$\text{subject to} \quad \frac{v_M}{ab^2} \leq \sigma_M, \quad a_L \leq a \leq a_U, \quad b_L \leq b \leq b_U, \tag{5.2.14}$$

which is a geometric program, that can be transformed into a convex program (this transformation is utilized in the following sections), with 2 variables, 1 constraint and box constraints on variables. This problem has the following analytic solution (that is derived in the Appendix A):

- if $\frac{v_M}{a_U b_U^2} > \sigma_M$, the problem is infeasible,
- if $\frac{v_M}{a_L b_L^2} \leq \sigma_M$, the solution is $a^* = a_L, b^* = b_L$,
- if $b = \sqrt{\frac{v_M}{a_L \sigma_M}}$ is within the bounds, $b^* = b, a^* = a_L$,
- else $a = \frac{v_M}{b_U^2 \sigma_M}$ and $a^* = a, b^* = b_U$.

This can be readily seen from the problem structure – a percentage increase in both a and b has the same result on the objective function value. However, percentage increase in b causes the left hand side of the inequality (5.2.12) to decrease faster than an equal percentage increase in a, making it preferable to increase b as much as needed (i.e. satisfying the inequality or the box constraint) before increasing a.

This result covers some of the numerical examinations done in [99] and [115] (which were more focused on the illustration of the combination of FEM and stochastic programming), without the need for using any optimization software (the only value one has to compute numerically is v_M). Another advantage is that for the same geometry (i.e. the same boundary conditions and number of elements) we can precompute the FEM matrices $\mathbb{C}$ and $\mathbb{K}$ (or its appropriate factorization, see [103], Chapter 3) and use them to quickly get optimal solution for different values of the load h.

5.2.2 | Additional Variable, Constraints and Convex Reformulation

The structure of the problem allows us to consider the material constant E as a variable, without destroying the convexity of the upcoming reformulation. This means we can choose the quality of the material – higher E corresponding to better and more expensive one. To be able to perform the convex reformulation, the dependence of the cost on the material (per volume units) must be in the form cE^p, with $c > 0, p \in \mathbf{R}$. The objective function then becomes cE^pabl, where the constants c and l can be dropped during the optimization.

An additional restriction on the solution involves the maximum absolute deflection of the beam, which we denote as δ_M. In our FEM formulation, the vector $\mathcal{V}$ includes both the deflection of the beam and its first derivative in each node of the division. The condition

on maximum deflection involves only the odd components in $\mathcal{V}$:

$$|\mathcal{V}_i| \leq \delta_M, i = 1, 3, 5, \ldots, 2N+1, \tag{5.2.15}$$

which is equivalent to a single inequality

$$\max_{i=1,3,5,\ldots,2N+1} |\mathcal{V}_i| \leq \delta_M, \tag{5.2.16}$$

using (5.2.10) and denoting the maximum of the odd components of $|12\mathbb{K}^{-1}h|$ as w_M we get

$$\frac{w_M}{Eab^3} \leq \delta_M. \tag{5.2.17}$$

The final constraint restricts the ratio between b and a to be less then the maximum allowed r_M.

Adding these constraints to (5.2.13)-(5.2.14), treating E as a design variable and changing the objective yields the following geometric program (presented here in its standard form):

$$\begin{aligned}
\underset{a,b,E}{\text{minimize}} \quad & E^p ab && (5.2.18)\\
\text{subject to} \quad & \frac{v_M}{\sigma_M} a^{-1} b^{-2} \leq 1, && (5.2.19)\\
& \frac{w_M}{\delta_M} E^{-1} a^{-1} b^{-3} \leq 1, && (5.2.20)\\
& \frac{1}{r_M} b a^{-1} \leq 1, && (5.2.21)\\
& a_L a^{-1} \leq 1, \frac{1}{a_U} a \leq 1, \;\; b_L b^{-1} \leq 1, \frac{1}{b_U} b \leq 1, \;\; E_L E^{-1} \leq 1, \frac{1}{E_U} E \leq 1, && (5.2.22)
\end{aligned}$$

where all the coefficients of the monomials in (5.2.18)-(5.2.22) are clearly positive, meaning we can use the following transformation to derive an equivalent convex program. First, we transform the variables: $y_a = \log a, y_b = \log b, y_E = \log E$. Then, we can write every monomial $f(a, b, E) = ca^{\alpha_1} b^{\alpha_2} E^{\alpha_3}$, where $c > 0, \alpha_1, \alpha_2, \alpha_3 \in \mathbf{R}$ in the form

$$f(a, b, E) = f(e^{y_a}, e^{y_b}, e^{y_E}) = ce^{\alpha_1 y_a} e^{\alpha_2 y_b} e^{\alpha_3 y_E} = e^{\alpha_1 y_a + \alpha_2 y_b + \alpha_3 y_E + \log c},$$

turning a monomial function into the exponential of an affine function. Next we transform the objective and the constraints, by taking the logarithm. Since every function both in the objective and the constraints is a monomial, the transformation results in a linear program:

$$\begin{aligned}
\underset{y_a, y_b, y_E}{\text{minimize}} \quad & y_a + y_b + p \cdot y_E && (5.2.23)\\
\text{subject to} \quad & -y_a - 2y_b + \log v_M - \log \sigma_M \leq 0, && (5.2.24)\\
& -y_a - 3y_b - y_E + \log w_M - \log \delta_M \leq 0, && (5.2.25)\\
& -y_a + y_b - \log r_M \leq 0, && (5.2.26)\\
& \log a_L \leq y_a \leq \log a_U, \;\; \log b_L \leq y_b \leq \log b_U, \;\; \log E_L \leq y_E \leq \log E_U. && (5.2.27)
\end{aligned}$$

Remark 5.2.1 *If the dependence of the material cost was $\sum_{i\in I} c_i E^{p_i}$, $c_i > 0$, instead of the simple cE^p, there would still be a convex reformulation, but it would no longer result in a linear program – there would be a term involving a logarithm of a sum of exponentials in the objective (see [17], p. 160–162).*

5.3 | Random Loads and Robust Solution

Next we investigate how the problem changes, when we introduce uncertainty. The previous papers [99] and [115] dealt with the situation, when the Young modulus E was random. In this paper, we assume that the randomness is in the load h. Instead of specifying the distribution of h by its cumulative distribution function or moment generating function (that would allow us to use the Bernstein approximation [81]), we devised a mechanism that produces random samples/scenarios. The use of scenarios is typical for engineering applications because of the difficulty of identifying the probability distribution. In this way, we imitate the situation when one does not know the distribution of a certain random variable, but only has access to its realizations – in our experience a much more common case. The sampling procedure is the following ($\mathcal{U}(a,b)$ denotes a uniform distribution):

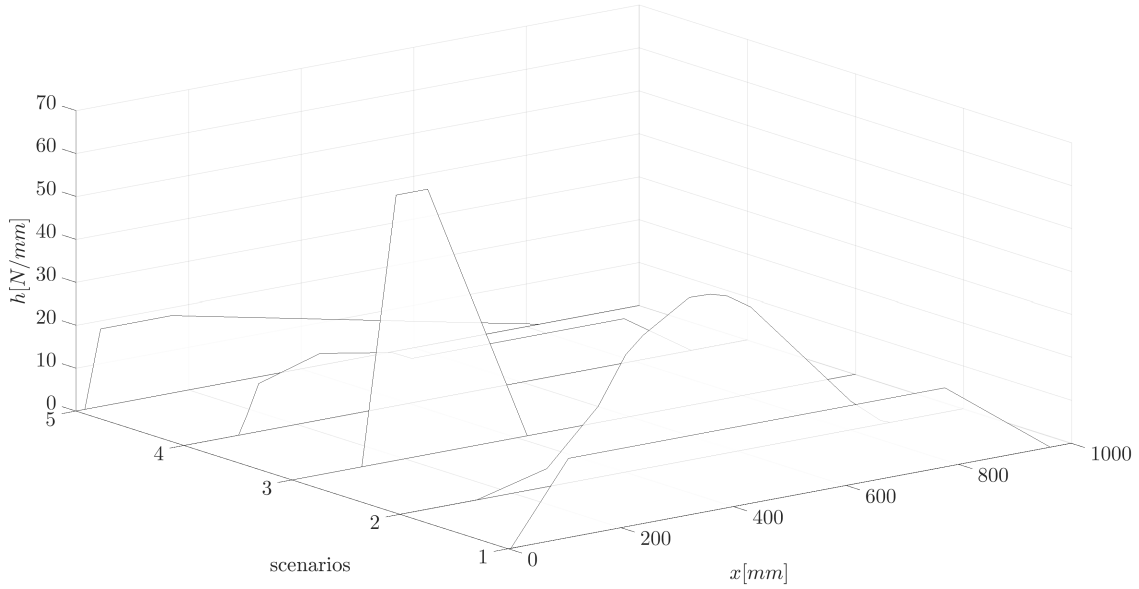


Fig. 5.2: A sample of 5 scenarios of the load $h(x)$.

0. Pick a random integer i between 1 and 4. Set $h(x) = 0$.
1. Repeat i times: Generate a Bernoulli trial.
   a) If 0, randomly pick 4 points $0 \le x_a \le x_b \le x_c \le x_d \le l$ and add to $h(x)$ a trapezoidal load $h_a(x)$ between x_a and x_d. Height of the trapezoid is $h_M \sim \mathcal{U}(0,1)$ (Fig. 5.2, scenarios 1 and 3).
   b) If 1, sample $h_\mu \sim \mathcal{U}(0,l), h_\sigma \sim \mathcal{U}(0,l)$ and add to $h(x)$ the bell curve load:

$$h_b(x) = \frac{1}{h_\sigma\sqrt{2\pi}} e^{\frac{-(x-h_\mu)^2}{2h_\sigma^2}}.$$

2. Normalize the load $h(x)$: Pick $H \sim \mathcal{U}(8000\,\mathrm{N}, 15000\,\mathrm{N})$. Compute $h_i = \int_0^l h(x)dx$, and set $h(x) = \frac{H}{h_i} h(x)$.

This sampling procedure generates very real-life like loads as can be seen in Fig. 5.2 (see [75]). Because we transformed the original problem (5.2.1)-(5.2.5) into the problem (5.2.23)-(5.2.27), we are much more interested in the values v_M and w_M resulting from the the different load scenarios, and the actual loads $h(x)$ are of little importance. In Fig. 5.3 we see the scatter plots and histograms of $\log v_M$ and $\log w_M$ using 2,000 scenarios of the load.

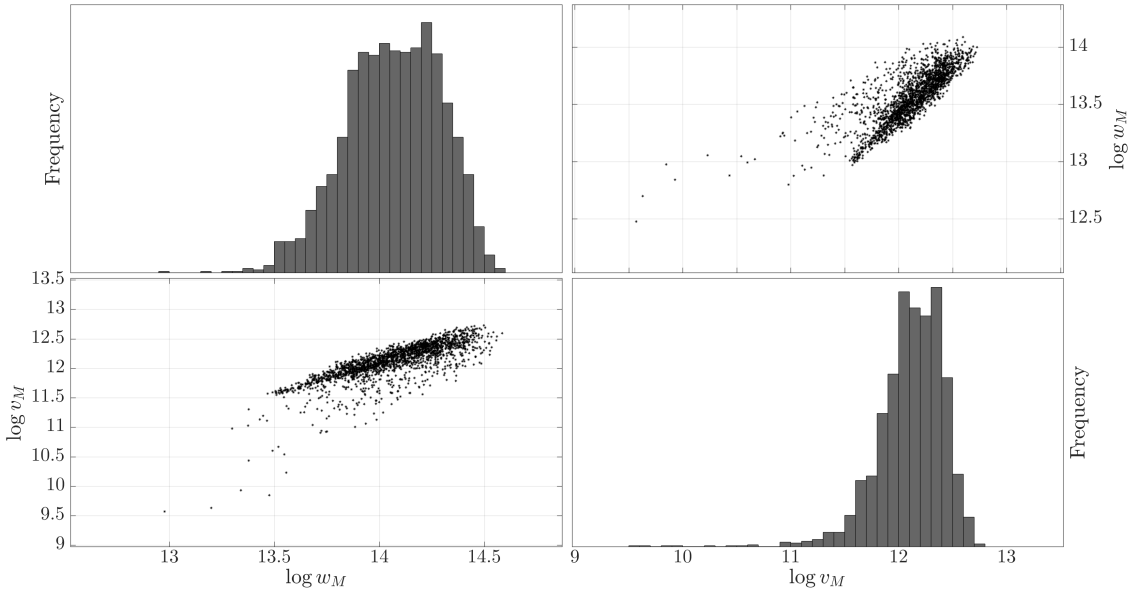


Fig. 5.3: Scatter plots and histograms of $\log v_M$ and $\log w_M$, 2,000 scenarios.

The important question is how to approach the optimization model (5.2.23)-(5.2.27) when some of its parameters, namely v_M and w_M, are random. One possibility is to use a so called robust formulation (see [9]), i.e. to enforce that the constraints will hold for any possible value of the random parameter. This results in the following formulation:

$$\begin{aligned}
\underset{y_a, y_b, y_E}{\text{minimize}} \quad & y_a + y_b + p \cdot y_E && && (5.3.1)\\
\text{subject to} \quad & -y_a - 2y_b + \log v_M(\xi) - \log \sigma_M \leq 0, && \forall \xi \in \Xi, && (5.3.2)\\
& -y_a - 3y_b - y_E + \log w_M(\xi) - \log \delta_M \leq 0, && \forall \xi \in \Xi, && (5.3.3)\\
& -y_a + y_b - \log r_M \leq 0, && && (5.3.4)\\
& \log a_L \leq y_a \leq \log a_U, \;\; \log b_L \leq y_b \leq \log b_U, \;\; \log E_L \leq y_E \leq \log E_U, && && (5.3.5)
\end{aligned}$$

where ξ is a random outcome from a sample space Ξ. This formulation is best suited for situation, when the violation of the constraints would have disastrous consequences.

Given our scenario generation procedure, the robust formulation requires us to find the scenarios that result in the highest values of v_M and w_M, and then optimize the design with respect to these extreme values. The generation procedure allows for point loads (setting all 4 point of the trapezoid into a single point) and the magnitude of the point

load is restricted to $15,000\,\mathrm{N}$ by the normalization step. This allows us to find the worst-case scenarios simply by using the formulas for the deflection and stress of a fixed beam under point load (these can be found in [86] and [113]). The analysis of the worst case situations is carried out in the Appendix B.

5.4 | Chance Constraints and Probabilistic Robust Design

The issue with the robust formulation is that it produces solutions that may be overly conservative. A different approach is to allow the possibility, that some of the constraints are violated, provided that the probability of violation is small. This corresponds to the following chance constrained (or probabilistic constrained, see [98]) formulation of the problem:

$$\underset{y_a,y_b,y_E}{\text{minimize}} \quad y_a + y_b + p \cdot y_E \tag{5.4.1}$$

$$\text{subject to} \quad P\begin{pmatrix} -y_a - 2y_b + \log v_M(\xi) - \log \sigma_M \leq 0, \\ -y_a - 3y_b - y_E + \log w_M(\xi) - \log \delta_M \leq 0 \end{pmatrix} \geq 1 - \epsilon, \tag{5.4.2}$$

$$-y_a + y_b - \log r_M \leq 0, \tag{5.4.3}$$

$$\log a_L \leq y_a \leq \log a_U, \;\; \log b_L \leq y_b \leq \log b_U, \;\; \log E_L \leq y_E \leq \log E_U, \tag{5.4.4}$$

where $1-\epsilon$ is the reliability level (or, alternatively, ϵ is the allowed violation probability). Except for some special cases, the formulation (5.4.1)-(5.4.4) is hard to solve exactly (see [98]).

One of the standard approaches (see [69]) to get an approximate solution is to fix the reliability level ϵ, draw a large number S of scenarios and construct a mixed-integer program, where for each scenario we have a binary decision variable, that corresponds to that scenario being neglected or not. One of the constraints then requires that we neglect less then ϵS scenarios. This method is clearly constrained by our ability to solve large mixed-integer programs. One of the most recent of the multiple approaches for solving the mixed-integer formulation was developed in [1].

In this paper we use a different approach based upon a method called Probabilistic Robust Design (see [19], [21] and [22]). This approach requires only that the objective is a convex function and that the constraint functions are convex for any realization of ξ – there are no other restrictions on the position of the random variable (such as only right-hand side, linearly perturbed, etc.). The first part of the method is, again, to draw

a large number S of scenarios (denoted by s) and solve the following problem:

$$\begin{aligned}
\underset{y_a,y_b,y_E}{\text{minimize}} \quad & y_a + y_b + p \cdot y_E && && (5.4.5)\\
\text{subject to} \quad & -y_a - 2y_b + \log v_M(s) - \log \sigma_M \leq 0, && s = 1,\dots,S, && (5.4.6)\\
& -y_a - 3y_b - y_E + \log w_M(s) - \log \delta_M \leq 0, && s = 1,\dots,S, && (5.4.7)\\
& -y_a + y_b - \log r_M \leq 0, && && (5.4.8)\\
& \log a_L \leq y_a \leq \log a_U, \;\; \log b_L \leq y_b \leq \log b_U, \;\; \log E_L \leq y_E \leq \log E_U, && && (5.4.9)
\end{aligned}$$

where the $2S$ constraints (5.4.6) and (5.4.7) can be reduced to the following 2 constraints:

$$-y_a - 2y_b + \max_s(\log v_M(s)) - \log \sigma_M \leq 0, \tag{5.4.10}$$

$$-y_a - 3y_b - y_E + \max_s(\log w_M(s)) - \log \delta_M \leq 0. \tag{5.4.11}$$

For a high enough choice of S, the optimal solution to (5.4.5)-(5.4.11) yields a feasible solution for the chance constrained problem (5.3.1)-(5.3.5) with high probability (see [19]). As investigated in [87], the approach tends to be overly conservative (i.e., the feasibility result holds, but we get a solution that is far from optimal for the chance constrained problem).

The result regarding optimality for this approach was added in [21], where the idea of discarding scenarios was developed. The main idea is, in addition to drawing S scenarios, to determine a number $k < S$, such that if we remove any k scenarios, the optimal solution of this modified problem is, again, feasible for the chance constrained problem with high probability. Furthermore, if the k scenarios are removed in an optimal fashion (i.e., we select those whose removal decreases the optimal objective value the most), there is a direct link between the optimal solution of the modified problem and the optimal solution of the chance constrained problem (in the sense that we get closer the more scenarios S we draw). Although this basically recovers the standard mixed-integer approach discussed above, there is a crucial difference in how the scenario-removal is achieved.

As discussed in [21], we can remove the k scenarios at once (the mixed-integer variant) or we can use a greedy approach that removes just one scenario at a time. In our case, the greedy approach makes perfect sense – there are only two scenarios (called support scenarios in [22]) whose removal can decrease the optimal objective value of (5.4.5)-(5.4.11):

$$s_1 = \operatorname*{argmax}_s(\log v_M(s)) \quad \text{and} \quad s_2 = \operatorname*{argmax}_s(\log w_M(s)).$$

To determine, which one of the two scenarios should be removed, we must solve two additional linear problems (with s_1 or s_2 temporarily removed) and compare their optimal objective values – this is repeated k times. The individual optimization problems have three variables and differ only in the value of one coefficient in (5.4.6) or (5.4.7) and as such can be efficiently solved by warm-starting the optimization algorithm with the last solution.

There is one different approach we will discuss, and that is the approximation of the joint chance constraint (5.4.2) by individual chance constraints:

$$P(-y_a - 2y_b + \log v_M(\xi) - \log \sigma_M \leq 0) \geq 1 - \epsilon_1, \tag{5.4.12}$$

$$P(-y_a - 3y_b - y_E + \log w_M(\xi) - \log \delta_M \leq 0) \geq 1 - \epsilon_2, \tag{5.4.13}$$

which become

$$-y_a - 2y_b + \Phi_v^{-1}(1-\epsilon_1) - \log \sigma_M \leq 0, \tag{5.4.14}$$

$$-y_a - 3y_b - y_E + \Phi_w^{-1}(1-\epsilon_2) - \log \delta_M \leq 0, \tag{5.4.15}$$

where Φ_v^{-1} and Φ_w^{-1} are the (empirical) quantile functions of $\log v_M(\xi)$ and $\log w_M(\xi)$, and ϵ_1, $\epsilon_2 > 0$ are appropriately chosen. The problem then becomes:

$$\underset{y_a, y_b, y_E}{\text{minimize}} \quad y_a + y_b + p \cdot y_E \tag{5.4.16}$$

$$\text{subject to} \quad -y_a - 2y_b + \Phi_v^{-1}(1-\epsilon_1) - \log \sigma_M \leq 0, \tag{5.4.17}$$

$$-y_a - 3y_b - y_E + \Phi_w^{-1}(1-\epsilon_2) - \log \delta_M \leq 0, \tag{5.4.18}$$

$$-y_a + y_b - \log r_M \leq 0, \tag{5.4.19}$$

$$\log a_L \leq y_a \leq \log a_U, \;\; \log b_L \leq y_b \leq \log b_U, \;\; \log E_L \leq y_E \leq \log E_U. \tag{5.4.20}$$

The choice of ϵ_1 and ϵ_2 is crucial – simply setting $\epsilon_1 = \epsilon_2 = \epsilon$ does not guarantee that the reliability of the optimal solution of (5.4.16)-(5.4.20) is better than $1-\epsilon$ (see Fig. 5.4). To obtain a safe approximation of the joint chance constraint (5.4.2), ϵ_1 and ϵ_2 must satisfy (see [81]): $\epsilon_1 + \epsilon_2 \leq \epsilon$, the simplest values being $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$.

5.5 | Numerical Results

Our goal is to obtain a trade-off curve between the optimal objective value (the weight of the beam) and the reliability of the design. To achieve this we used our scenarios generation technique to draw two large sets of scenarios, where the first one contained S_1 and the second S_2 scenarios. The first one was used for the optimization part (i.e. solving (5.4.5)-(5.4.11)), the second one was used for the estimate of the reliability level ϵ. The method proceeded as follows:

0. Generate the two sets of scenarios.
   Repeat k times:
1. Solve (5.4.5)-(5.4.11) using the first set of scenarios. Obtain an optimal design.
2. Estimate the reliability of the design using the second set of scenarios: given a design in the form of a, b and E, the constraints (5.4.6)-(5.4.7) either both hold, or at least one of them does not hold. This outcome describes a binomial random variable – compute its point estimate (a fraction of scenarios for which at least one of the constraints did not hold) and its 99.9% confidence interval (using the Clopper-Pearson interval).

3. Determine, which one of the two support scenarios to remove, and delete it from the first set of scenarios. Return to 1.

The problem setting under the numerical investigation was as follows: the length of the beam $l = 1\,\mathrm{m}$, number of elements for the FEM formulation $N = 1{,}000$, objective coefficient $p = \frac{1}{2}$, limits on the variables $a_L = b_L = 10^{-2}\,\mathrm{m}$, $a_U = b_U = 10^{-1}\,\mathrm{m}$, $E_L = 1.9{\cdot}10^5\,\mathrm{MPa}$ and $E_U = 2.2{\cdot}10^5\,\mathrm{MPa}$, maximum stress $\sigma_M = 120\,\mathrm{MPa}$, maximum deflection $\delta_M = 5{\cdot}10^{-4}\,\mathrm{m}$, maximum ratio between the variables $r_M = 5$, number of scenarios in the first set $S_1 = 50{,}000$, number of scenarios in the second set $S_2 = 100{,}000$, number of scenarios to discard $k = 2{,}500$.

The number of elements was chosen such that the length of one element $d = \frac{l}{N} = 10^{-3}\,\mathrm{m}$ results in accuracy $\mathcal{O}(10^{-6})$ of the FEM approximation, which is roughly of the same order as the accuracy of the optimization algorithm (termination criteria for optimality), that was set to 10^{-7}. The accuracy of the FEM was checked using the analytic results in the Appendix B and using ANSYS (commercial engineering simulation software, see [106]). The FEM formulation was programmed and solved in MATLAB, the optimization parts were computed using the CVX modeling system (see [39] and [40]).

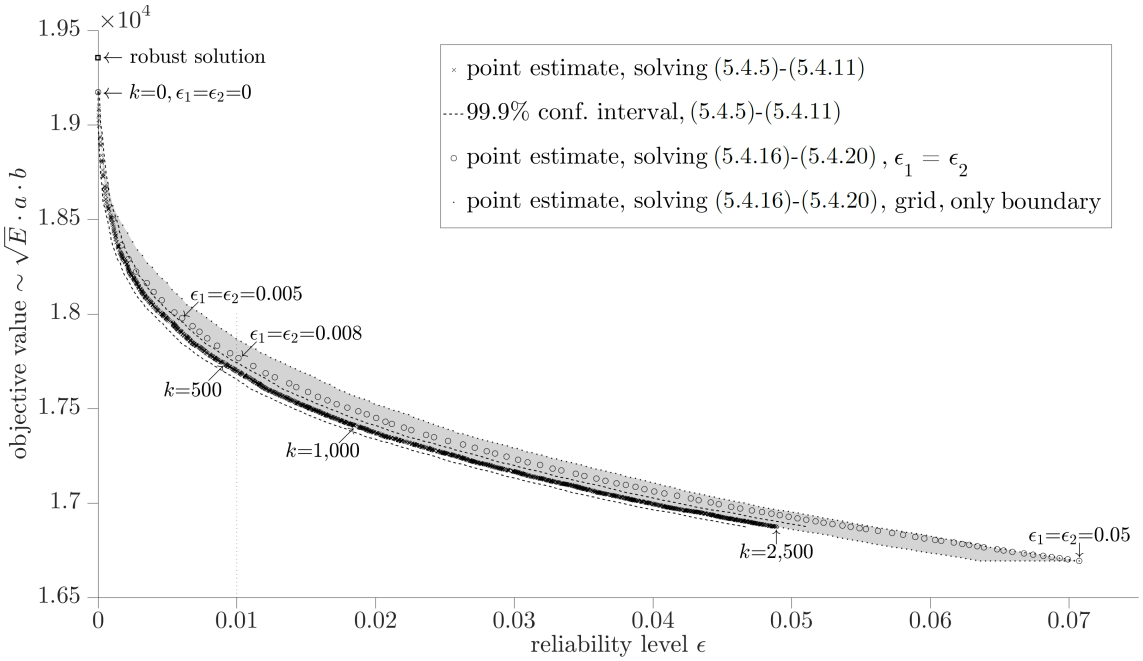


Fig. 5.4: The trade-off between reliability and optimal objective value.

In Fig. 5.4 is depicted the trade-off between the reliability level ϵ and the optimal objective value using the two approaches (5.4.5)-(5.4.11) and (5.4.16)-(5.4.20). In the first approach we gradually remove the scenarios (upto $k = 2{,}500$) – the computational time for each iteration (two optimization problems, scenario removal) was around 0.4 s. In the second approach (5.4.16)-(5.4.20) we vary the values of $\epsilon_1 = \epsilon_2$ between 0 and 0.05 – the computational time for each value was around 0.2 s. Furthermore, used a grid of 1,001 steps for ϵ_1 and ϵ_2 between 0 and 0.05 and computed the results for all of these grid values

(they fill the grey area in Fig. 5.4), this took 45 hours. The robust solution was computed using the results in the Appendix B (maximum point loads in $\frac{1}{2}l$ and $\frac{1}{3}l$).

The comparison between the two methods favours the scenario-removal one (5.4.5)-(5.4.11) over solving (5.4.16)-(5.4.20) with $\epsilon_1 = \epsilon_2$, as it produces designs with better objective value. For example, given the target (point estimate of) $\epsilon = 0.01$, the closest design produced by (5.4.16)-(5.4.20) is for $\epsilon_1 = \epsilon_2 = 0.008$, with the objective value $1.776{\cdot}10^4$, whereas the method using (5.4.5)-(5.4.11) with $k = 568$ deleted scenarios achieved the objective value $1.769{\cdot}10^4$. Moreover, the scenario-removal method (5.4.5)-(5.4.11) produced as good solutions as the best ones using the grid values for ϵ_1 and ϵ_2 and solving (5.4.16)-(5.4.20).

The shape of the trade-off heavily depends on the distribution of $h(x)$ (and, consequently, on the distribution of v_M and w_M). For the computation we used the scenario generation described earlier, which was constructed ad hoc to demonstrate the method. In a real situation (e.g., the one in [62]), the scenario generation will be swapped for the particular problem-specific outcomes.

5.5.1 | Additional Computations

The results in this subsection were not included in the original paper, but are rather interesting. At first, the "grid computations" were at first carried out on a sparser grid with only 51 steps – the resulting 2,601 optimization problems took roughly over half the time required for computing the scenario removal approach (2,500 removed scenarios, two optimization problems in each step). The results of this computation can be seen in Fig.

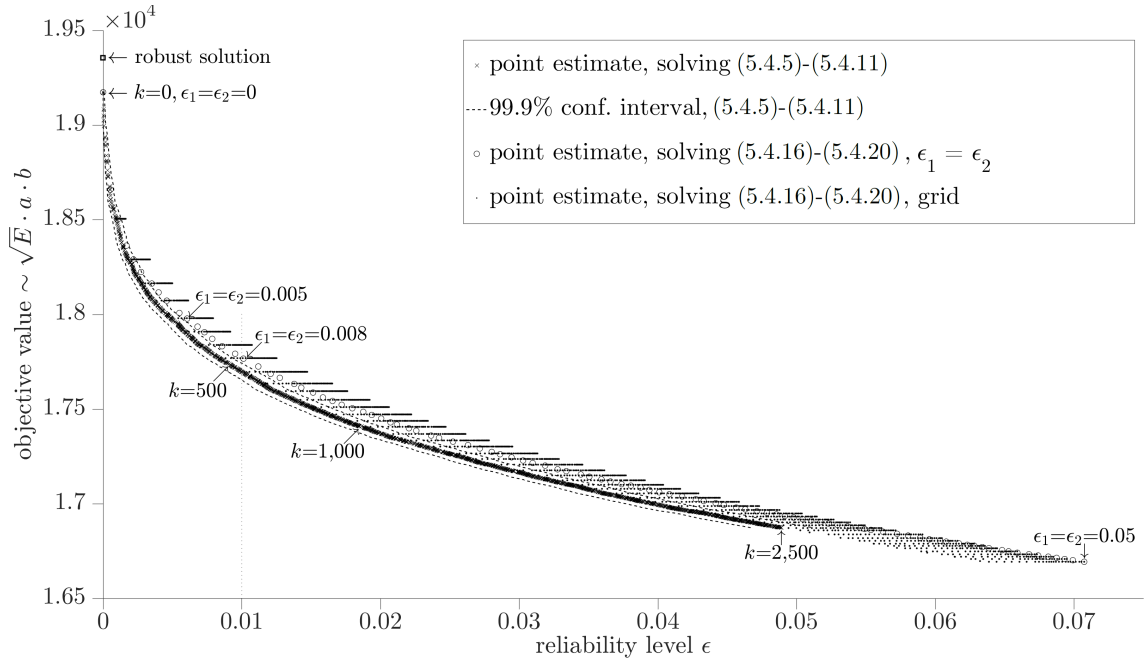


Fig. 5.5: The trade-off between reliability and optimal objective value. Sparser grid.

5.5 – there are quite large "gaps", especially for low values of the estimated reliability

level. Furthermore, although the minimal points (the ones most to the left and down – the best ones) were as good as the comparable points obtained by the scenario removal, the grid method produced a lot more "bad points" than it produced "good", or "close to good" ones.

This became even more apparent when we increased the number of steps from 51 to 1,001. The Fig. 5.5 shows this in more detail – the individual points obtained from the grid are plotted with a very transparent grey color, revealing a much "denser" concentration of grid points quite far away from the ones obtained by the scenario removal.

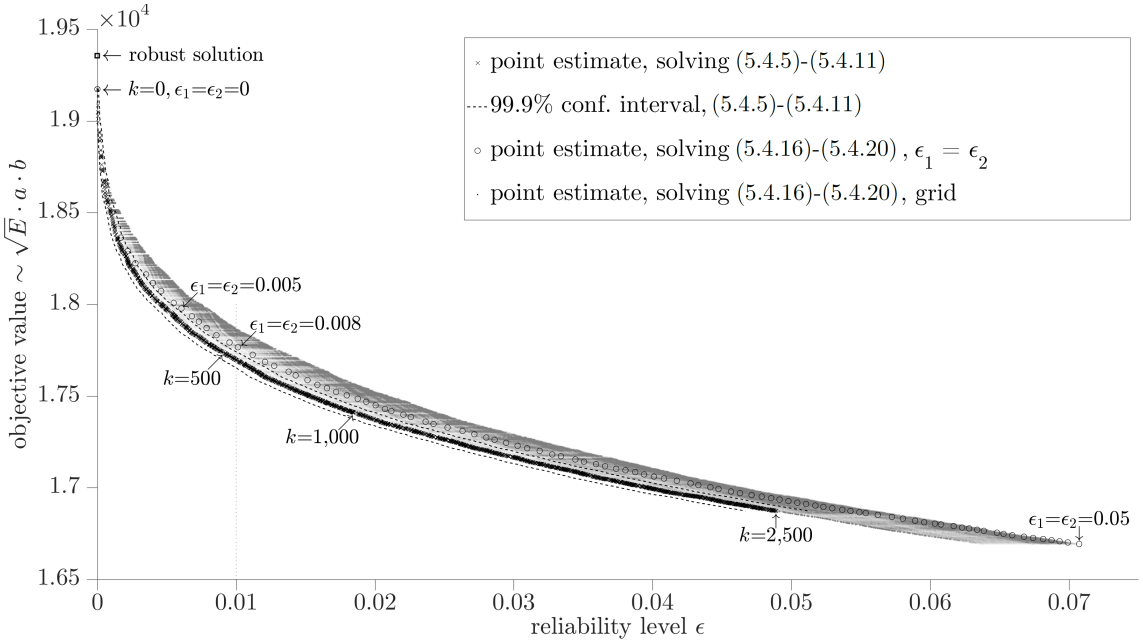


Fig. 5.6: The trade-off between reliability and optimal objective value. Denser grid.

5.6 | Conclusion

In this paper, we have presented new reformulation for the optimal beam design problem, that serves as a test example for a larger set of problems solvable by similar techniques as presented. This reformulation leads to a geometric program and as such can be solved to global optimality. We then used this reformulation and extended the problem by considering randomness in the load and presented the robust and chance constrained problems. The chance constrained variant was handled by the Probabilistic Robust Design approach. For the given scenario generation procedure we computed the trade-off between reliability and optimal objective value. Further research will be focused on situations, when the cross-section of the beam is not rectangular and the reformulation results in a possibly non-convex problem.

Acknowledgement

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic INTER-COST project LTC18053 and by the project "Computer Simulations for Effective Low-Emission Energy" funded as project No. CZ.02.1.01/0.0/0.0/16_026/0008392 by Operational Programme Research, Development and Education, Priority axis 1: Strengthening capacity for high-quality research.

Comments by the anonymous reviewers have significantly improved the paper and are also greatly acknowledged.

Appendix

A. The Analytic Solution

Here we derive the analytic solution for (5.2.13)-(5.2.14). We use the same convex reformulation as in (5.4.1)-(5.4.4) to derive an equivalent linear program:

$$\underset{y_a,y_b}{\text{minimize}} \quad y_a + y_b \tag{5.6.1}$$

$$\text{subject to} \quad -y_a - 2y_b + \log\frac{v_M}{\sigma_M} \le 0, \tag{5.6.2}$$

$$\log a_L \le y_a \le \log a_U, \quad \log b_L \le y_b \le \log b_U. \tag{5.6.3}$$

0. a) If $\log\frac{v_M}{\sigma_M} \le \log a_L + 2\log b_L$, we are done, $a^* = a_L, b^* = b_L$.
   b) If $\log\frac{v_M}{\sigma_M} > \log a_U + 2\log b_U$, the problem is infeasible.
1. Otherwise, we need $y_a, y_b : y_a + 2y_b = \log\frac{v_M}{\sigma_M}, \log a_L \le y_a \le \log a_U, \log b_L \le y_b \le \log b_U$.
2. The KKT conditions:

$$\begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix} + \nu\begin{pmatrix}1\\2\end{pmatrix} + \lambda_1\begin{pmatrix}-1\\0\end{pmatrix} + \lambda_2\begin{pmatrix}0\\-1\end{pmatrix} + \lambda_3\begin{pmatrix}1\\0\end{pmatrix} + \lambda_4\begin{pmatrix}0\\1\end{pmatrix}, \tag{5.6.4}$$

$$\lambda_1(\log a_L - y_a) = 0, \lambda_2(\log b_L - y_b) = 0, \lambda_3(y_a - \log a_U) = 0, \lambda_4(y_b - \log b_U) = 0, \tag{5.6.5}$$

$$\log a_L \le y_a \le \log a_U, \log b_L \le y_b \le \log b_U, y_a + 2y_b = \log\frac{v_M}{\sigma_M}, \lambda_i \ge 0, i = 1,\dots,4. \tag{5.6.6}$$

3. From complementary slackness condition (5.6.5) we get 16 different possible situations - corresponding to a or b being at the specific bounds. From the outset it is clear that the variables cannot be at the lower and upper bound at the same time: λ_1 and λ_3 cannot be both nonzero, the same holds for λ_2 and λ_4. This rules out 7 possibilities.
4. If $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0, \lambda_4 = 0$, from (5.6.4) we have

$$\nu = -1, \quad \text{from the first row,} \qquad \nu = -\frac{1}{2}, \quad \text{from the second row,}$$

which is not possible. This means that there cannot be an optimal solution such that $a_L < a^* < a_U$ and $b_L < b^* < b_U$ at the same.

5. If $\lambda_1 > 0, \lambda_2 = 0, \lambda_3 = 0, \lambda_4 = 0$, i.e. $a^* = a_L, y_a^* = \log a_L$. From (5.6.4) we have

$$\nu = -\frac{1}{2}, \quad \lambda_1 = \frac{1}{2} > 0,$$

meaning that $y_b^* = \frac{1}{2}\log\frac{v_M}{a_L\sigma_M}$ and $b^* = e^{y_b^*} = \sqrt{\frac{v_M}{a_L\sigma_M}}$ is a possible solution, provided $b_L < b^* < b_U$.

6. If $\lambda_1 > 0, \lambda_2 > 0, \lambda_3 = 0, \lambda_4 = 0$, i.e. $a^* = a_L, b^* = b_L$. This is the situation in 0. a).
7. If $\lambda_1 > 0, \lambda_2 = 0, \lambda_3 = 0, \lambda_4 > 0$, i.e. $a^* = a_L, b^* = b_U$. From (5.6.4) we have

$$\lambda_1 = 1 + \nu > 0 \Rightarrow \nu > -1, \quad \lambda_4 = -1 - 2\nu > 0 \Rightarrow \nu < -\frac{1}{2}, \quad \text{which is possible.}$$

This is the (arguably rare) situation when $\log a_L + 2\log b_U = \log\frac{v_M}{\sigma_M}$.

8. If $\lambda_1 = 0, \lambda_2 > 0, \lambda_3 = 0, \lambda_4 = 0$, i.e. $a^* = a_U$. From (5.6.4) we have

$$\nu = -1, \quad \lambda_2 = -1 > 0, \quad \text{which is not possible.}$$

9. If $\lambda_1 = 0, \lambda_2 > 0, \lambda_3 > 0, \lambda_4 = 0$, i.e. $a^* = a_U, b^* = b_L$ from (5.6.4) we have

$$\lambda_3 = -1 - \nu > 0 \Rightarrow \nu < -1, \quad \lambda_2 = 1 + 2\nu > 0 \Rightarrow \nu > -\frac{1}{2}, \quad \text{which is not possible.}$$

10. If $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 > 0, \lambda_4 = 0$, i.e. $b^* = b_L$. From (5.6.4) we have

$$\nu = -\frac{1}{2}, \quad \lambda_3 = -\frac{1}{2} > 0, \quad \text{which is not possible.}$$

11. If $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 > 0, \lambda_4 > 0$, i.e. $a^* = a_U, b^* = b_U$. From (5.6.4) we have

$$\lambda_3 = -1 - \nu > 0 \Rightarrow \nu < -1, \quad \lambda_4 = -1 - 2\nu > 0 \Rightarrow \nu < -\frac{1}{2}, \quad \text{which is possible.}$$

This is the situation when $\log a_U + 2\log b_U = \log\frac{v_M}{\sigma_M}$.

12. If $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0, \lambda_4 > 0$, i.e. $b^* = b_U$. From (5.6.4) we have

$$\nu = -1, \quad \lambda_4 = 1 > 0, \quad \text{which is possible, } y_a^* = \log\frac{v_M}{b_U^2\sigma_M}, a^* = \frac{v_M}{b_U^2\sigma_M}.$$

B. Worst Case Deflection and Stress for Point Load

The following results are using the known formulas for deflection and bending moment for fixed beam under a point load that can be found in [86] and [113]. The Fig. 5.7 depicts the situation and provides a graphical description of the used notation.

The maximum deflection of a fixed beam under point load is computed by the following formula (can be found in [113], p. 190):

$$\delta_M = \frac{2Hl_a^3l_b^2}{3EI(3l_a + l_b)^2}, \tag{5.6.7}$$

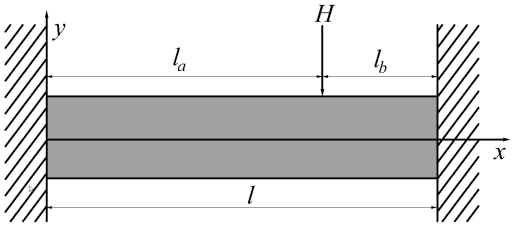


Fig. 5.7: Point load.

where l_a and l_b correspond to the location of the point load ($l_a + l_b = l$), I is the moment of inertia of the cross-section and E is the Young modulus. In our case $I = \frac{ab^3}{12}$. If we look at (5.6.7) as a function of the location l_a of the point load, its maximum occurs when $l_a = l_b = \frac{l}{2}l$.

The maximum stress for each point $x \in [0, l]$ in the beam can be expressed in the following terms: $\sigma_M(x) = \frac{M(x)}{I} y_M$, where $M(x)$ is the bending moment and $y_M = \pm\frac{b}{2}$. This allows us to use the formulas for maximum bending moment of fixed beam under point load to find the critical points (the signs in the formulas are neglected, since the constraint (5.2.3) restricts the absolute value of the stress). The bending moment of a beam under point load changes linearly between the points $0, l_a$, and l, so it suffices to compute the bending moment in these three points. Given a point load at $x = l_a$ bending moment at the ends of the beam ($x = 0$ and $x = l$) is

$$\text{left end: } M(x=0) = \frac{H l_a l_b^2}{l^2}, \quad \text{right end: } M(x=l) = \frac{H l_a^2 l_b}{l^2},$$

the maximum occurs when $l_a = \frac{1}{3}l$ (or $l_a = \frac{2}{3}l$) resulting in $M(x = 0 \text{ or } x = l) = \frac{4}{27}lH$.

The moment at the location of the point is $M(x = l_a) = \frac{2Hl_a^2 l_b^2}{l^3}$, for which the maximum occurs when $l_a = l_b = \frac{1}{2}l$, resulting in $M(x = \frac{1}{2}l) = \frac{1}{8}lH$. This means that worst case occurs, when the point load is located in $l_a = \frac{1}{3}l$ or $l_a = \frac{2}{3}l$.

Bibliography

[1] L. Adam and M. Branda. Nonlinear chance constrained problems: Optimality conditions, regularization and solvers. *Journal of Optimization Theory and Applications*, 170(2):419–436, 2016.

[2] S. Ahmed, J. Luedtke, Y. Song, and W. Xie. Nonanticipative duality, relaxations, and formulations for chance-constrained stochastic programs. *Mathematical Programming*, 162(1–2):51–81, 2017.

[3] T. Alamo, R. Tempo, A. Luque, and D. R. Ramirez. Randomized methods for design of uncertain systems: Sample complexity and sequential algorithms. *Automatica*, 52(1):160–172, 2015.

[4] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath. Coherent measures of risk. *Mathematical Finance*, 9:203–228, 1999.

[5] B. R. Barmish and P. S. Shcherbakov. On avoiding vertexization of robustness problems: The approximate feasibility concept. *IEEE Transactions on Automatic Control*, 47(5):819–824, 2002.

[6] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, 3rd edition, 2006.

[7] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society, Series B*, 17:173–184, 1955.

[8] A. T. Beck, W. J. S. Gomes, R. H. Lopez, and L. F. F. Miguel. A comparison between robust and risk-based optimization under uncertainty. *Structural and Multidisciplinary Optimization*, 52(3):479–492, 2015.

[9] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 1st edition, 2009.

[10] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. SIAM, 1st edition, 2001.

[11] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[12] J. Bezanson, A. Edelman, S. Karpinski, and Shah V. B. Julia: A fresh approach to numerical computing. *SIAM Review*, 59:65–98, 2017.

[13] J. R. Birge and F. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988.

[14] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, 1st edition, 1997.

[15] J. Bisschop. *AIMMS Optimization Modeling*. Lulu.com, 1st edition, 2006.

[16] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[17] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 1st edition, 2004.

[18] G. C. Calafiore and M. C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming, Ser. A*, 102(1):25–46, 2005.

[19] G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.

[20] M. C. Campi and S. Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal of Optimization*, 19(3):1211–1230, 2008.

[21] M. C. Campi and S. Garatti. A sampling-and-discarding approach to chance-constrained optimization: Feasibility and optimality. *Journal of Optimization Theory and Applications*, 148:257–280, 2011.

[22] A. Care, S. Garatti, and M. C. Campi. Scenario min-max optimization and the risk of empirical costs. *SIAM Journal on Optimization*, 25(4):2061–2080, 2015.

[23] A. Charnes, W. W. Cooper, and G. H. Symonds. Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil. *Management Science*, 4(3):235–263, 1958.

[24] G. B. Dantzig. Linear programming under uncertainty. *Management Science*, 1:197–206, 1955.

[25] G. B. Dantzig and A. Madansky. On the solution of two-stage linear programs under uncertainty. In *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 165–176, Berkeley, 1961. University of California Press.

[26] G. B. Dantzig and P. Wolfe. The decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[27] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82:421–439, 1956.

[28] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[29] J. Dupacova. The minimax approach to stochastic programming and an illustrative application. *Stochastics*, 20:73–88, 1987.

[30] J. Dupacova. Stochastic geometric programming with an application. *Kybernetika*, 46(3):374–386, 2010.

[31] H. A. Eiselt and V. Marianov. Location modeling for municipal solid waste facilities. *Computers & Operations Research*, 62:305–315, 2015.

[32] C. I. Fabian. Bundle-type methods for inexact data. *Central Eur. J. Oper. Res.*, 8:35–55, 2000.

[33] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications.* Oxford University Press, 1st edition, 1995.

[34] C. A. Floudas and P. M. Pardalos. *Encyclopedia of Optimization.* Springer, 2nd edition, 2009.

[35] GAMS Development Corporation. General algebraic modeling system (GAMS) release 24.2.1., 2013.

[36] A. H. Gandomi, X.-S. Yang, and A. H. Alavi. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, 29(1):17–35, 2013.

[37] H. Gassmann. MSLiP: a computer code for the multistage stochastic linear programming problem. *Mathematical Programming*, 47:407–423, 1990.

[38] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[39] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html.

[40] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, mar 2014.

[41] Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[42] J. Haslinger and R. A. E. Mäkinen. *Introduction to Shape Optimization: Theory, Approximation, and Computation (Advances in Design and Control).* SIAM, 1st edition, 2003.

[43] R. Henrion and W. Römisch. Metric regularity and quantitative stability in stochastic programs with probabilistic constraints. *Mathematical Programming*, 84(1):55–88, 1999.

[44] R. Henrion and W. Römisch. Hölder and Lipschitz stability of solution sets in programs with probabilistic constraints. *Mathematical Programming*, 100(3):589–611, 2004.

[45] R. Henrion and C. Strugarek. Convexity of chance constraints with independent random variables. *Computational Optimization and Applications*, 41(2):263–276, 2008.

[46] J. L. Higle and S. Sen. *Stochastic decomposition: A statistical method for large scale stochastic linear programming*. Kluwer, 1st edition, 1996.

[47] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2nd edition, 2004.

[48] D. Hrabec, P. Popela, J. Roupec, J. Mazal, and P. Stodola. Two-stage stochastic programming for transportation network design problem. *Advances in Intelligent Systems and Computing*, 378:17–25, 2015.

[49] IBM Corp. *IBM ILOG CPLEX Optimization Studio: CPLEX User's Manual. Version 12, Release 7.*, 2016.

[50] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley & Sons, 1st edition, 1994.

[51] J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[52] A. J. King and R. T. Rockafellar. Asymptotic theory for solutions in statistical estimation and stochastic programming. *Mathematics of Operations Research*, 18(1):148–162, 1993.

[53] A. J. King and S. W. Wallace. *Modeling with stochastic programming*. Springer, 1st edition, 2012.

[54] K. C. Kiwiel. *Methods of Descent for Nondifferentiable Optimization*. Springer, 1st edition, 1985.

[55] J. J. Klemes, P. S. Varbanov, V. Y. Fan, and H. L. Lam. Twenty years of PRES: Past, present and future – process integration towards sustainability. *Chemical Engineering Transactions*, 61:1–24, 2017.

[56] J. Kudela and P. Popela. Two-stage stochastic facility location problem: GA with Benders decomposition. *Mendel*, pages 53–58, 2015.

[57] J. Kudela and P. Popela. Warm-start cuts for generalized Benders decomposition. *Kybernetika*, 53(6):1012–1025, 2017.

[58] J. Kudela and P. Popela. Chance constrained optimal beam design: Convex reformulation and probabilistic robust design. *Kybernetika*, 54(6):1201–1217, 2018.

[59] J. Kudela, P. Popela, R. Somplak, M. Malek, A. Rychtar, and D. Hrabec. The L-shaped method for large-scale mixed-integer waste management decision making problems. *Chemical Engineering Transactions*, 61:1087–1092, 2017.

[60] J. Kudela, R. Somplak, V. Nevrly, and T. Lipovsky. Robust waste transfer station planning by stochastic programming. *Chemical Engineering Transactions*, 70(1):889–894, 2018.

[61] C. M. Lagoa, X. Li, and M. Sznaier. Probabilistically constrained linear programs and risk-adjusted controller design. *SIAM Journal on Optimization*, 15(1):938–951, 2005.

[62] I. Lanikova, P. Stepanek, and P. Simunek. Optimized design of concrete structures considering environmental aspects. *Advances in Structural Engineering*, 17(4):495–511, 2014.

[63] C. Lemarechal. *Nonsmooth optimization and descent methods.* IIASA, Laxenburg, Austria, 1978.

[64] C. Lemarechal, A. Nemirovski, and Y. Nesterov. New variants of bundle methods. *Mathematical Programming*, 69:111–147, 1995.

[65] M. Leps and M. Sejnoha. New approach to optimization of reinforced concrete beams. *Computers and Structures*, 81(1):1957–1966, 2003.

[66] V. L. Levin. Application of E. Helly's theorem to convex programming, problems of best approximation and related questions. *Mathematics of the USSR-Sbornik*, 8:235–247, 1969.

[67] J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Comput. Optim. Appl.*, 24:207–250, 2003.

[68] M. Lubin and I. Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[69] J. Luedtke and S. Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699, 2008.

[70] J. Luedtke, S. Ahmed, and G. L. Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming, Ser. A*, 122(2):247–272, 2010.

[71] A. Madansky. Inequalities for stochastic linear programming problems. *Management Science*, 6:197–204, 1960.

[72] F. Maggioni, E. Allevi, and M. Bertocchi. Bounds in multistage linear stochastic programming. *Journal of Optimization Theory and Applications*, 163(1):200–229, 2014.

[73] F. Maggioni and G. C. Pflug. Bounds and approximations for multistage stochastic programs. *SIAM Journal on Optimization*, 26(1):831–855, 2016.

[74] W.-K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24:47–56, 1999.

[75] P. Marek, J. Brozzetti, and M. Gustar. *Probabilistic Assessment of Structures using Monte Carlo Simulation.* TeReCo, 1st edition, 2001.

[76] H. M. Markowitz. Portfolio selection. *Journal of Finance*, 7:77–91, 1952.

[77] L. B. Miller and H. Wagner. Chance-constrained programming with joint constraints. *Operations Research*, 13(6):930–945, 1965.

[78] H. D. Mittelmann. The state-of-the-art in conic optimization software. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 671–686. Springer, 2012.

[79] MOSEK. *The MOSEK C optimizer API manual. Version 7.0.*, 2018.

[80] A. Nemirovski. Several NP-hard problems arising in robust stability analysis. *SIAM J. Matrix Anal. Appl.*, 6(2):99–105, 1993.

[81] A. Nemirovski. On safe tractable approximations of chance constraints. *European Journal of Operational Research*, 219(3):707–718, 2012.

[82] A. Nemirovski and Shapiro A. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):959–996, 2006.

[83] A. Nemirovski and A. Shapiro. *Continuous Optimization*, chapter On Complexity of Stochastic Programming Problems. Springer US, 2005.

[84] Y. Nestrov and A. Nemirovski. *Interior Point Polynomial Algorithms in Convex Programming.* SIAM, 1st edition, 1994.

[85] J. Nocedal and S. Wright. *Numerical Optimization.* Springer-Verlag New York, 2nd edition, 2006.

[86] E. Oberg, F. D. Jones, and H. H. Ryffel. *Machinery's Handbook Guide.* Industrial Press, 29th edition, 2012.

[87] B. K. Pagnoncelli, S. Ahmed, and A. Shapiro. Sample average approximation method for chance constrained programming: Theory and applications. *Journal of Optimization Theory and Applications*, 142(2):399–416, 2009.

[88] B. K. Pagnoncelli, D. Reich, and M. C. Campi. Risk-return trade-off with the scenario approach in practice: A case study in portfolio selection. *Journal of Optimization Theory and Applications*, 155(2):707–722, 2012.

[89] G. C. Pflug and W. Römisch. *Modeling, measuring and managing risk*. World Scientific, 1st edition, 2007.

[90] A. Prekopa. On probabilistic constrained programming. In *Proceedings of the Princeton Symposium on Mathematical Programming*. Princeton University Press, 1970.

[91] A. Prekopa. Contributions to the theory of stochastic programming. *Mathematical Programming*, 4(1):202–221, 1973.

[92] A. Prekopa. *Stochastic Programming*. Kluwer, 1st edition, 1995.

[93] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control Optim.*, 14:877–898, 1976.

[94] R. T. Rockafellar and S. Uryasev. Optimization of conditional Value-at-Risk. *The Journal of Risk*, 2(3):21–41, 2000.

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

[96] G. I. N. Rozvany and T. Lewinski, editors. *Topology Optimization in Structural and Continuum Mechanics*. CISM Courses and Lectures. Springer, 1st edition, 2014.

[97] A. Ruszczynski. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[98] A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming*, volume 10 of *Handbooks in Operations Research and Management Science*. Elsevier, 1st edition, 2003.

[99] Z. Sabartova and P. Popela. Beam design optimization model with FEM based constraints. *Mendel*, 1:422–427, 2012.

[100] F. Shan, L. Zhang, and X. Xiao. A smoothing function approach to joint chance-constrained programs. *Journal of Optimization Theory and Applications*, 163(1):181–199, 2014.

[101] A. Shapiro. Asymptotic analysis of stochastic programs. *Annals of Operations Research*, 30:169–186, 1991.

[102] A. Shapiro and A. Kleywegt. Minimax analysis of stochastic programs. *Optimization Methods and Software*, 17:523–542, 2002.

[103] I. M. Smith and D. V. Griffiths. *Programming the finite element method.* John Wiley & Sons,, 4nd edition, 2004.

[104] R. Somplak, V. Nevrly, M. Malek, M. Pavlas, and J. J. Klemes. Network flow based model applied to sources, sinks and optimal transport of combustible waste. *Chemical Engineering Transactions*, 61:991–996, 2017.

[105] Y. Song, J. Luedtke, and S. Küçükyavuz. Chance-constrained binary packing problems. *INFORMS Journal on Computing*, 26(4):735–747, 2014.

[106] Swanson Analysis Systems, Inc. *ANSYS User's Manual Vol. I*, 1994.

[107] S. Takriti and S. Ahmed. On robust optimization of two-stage systems. *Mathematical Programming*, 99:109–126, 2004.

[108] R. M. Van Slyke and R. J.-B. Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM J. Appl. Math.*, 17(4):638–663, 1969.

[109] A. Viktorin, D. Hrabec, and M. Pluháček. Multi-chaotic differential evolution for vehicle routing problem with profits. pages 991–996, 2016.

[110] T. G. Walmsley, P. S. Varbanov, and J. J. Klemes. Networks for utilising the organic and dry fractions of municipal waste: P-graph approach. *Chemical Engineering Transactions*, 61:1357–1362, 2017.

[111] W. Wiesemann, D. Kuhn, and M. Sim. Distributionally robust convex optimization. *Operations Research*, 62(6):1358–1376, 2014.

[112] C. Wolf, C. I. Fabian, A. Koberstein, and L. Suhl. Applying oracles of on-demand accuracy in two-stage stochastic programming: A computational study. *European Journal of Operational Research*, 239(2):437–448, 2014.

[113] W. C. Young, R. G. Budynas, and A. M. Sadegh. *Roark's Formulas for Stress and Strain.* McGraw-Hill Education, 8th edition, 2011.

[114] J. Zackova. On minimax solutions of stochastic linear programming problems. *Čas. Pěst. Mat.*, 91:423–430, 1966.

[115] E. Zampachova, P. Popela, and M. Mrazek. Optimum beam design via stochastic programming. *Kybernetika*, 46(3):571–582, 2010.

[116] X. Zhuang and R. Pan. A sequential sampling strategy to improve reliability-based design optimization with implicit constraint functions. *Journal of Mechanical Design*, 134(2), 2012.

[117] V. Zverovich, C. I. Fabian, E. F. D. Ellison, and G. Mitra. A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition. *Mathematical Programming Computation*, 4(3):211–238, 2012.