IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 2, 2020, accepted April 21, 2020, date of publication April 27, 2020, date of current version May 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2990726

Pool & Discard Algorithm for Chance Constrained Optimization Problems

JAKUB KŮDELA[1] **AND PAVEL POPELA**[2]
[1]Institute of Computer Science and Automation, Faculty of Mechanical Engineering, Brno University of Technology, 61669 Brno, Czech Republic
[2]Institute of Mathematics, Faculty of Mechanical Engineering, Brno University of Technology, Brno, Czech Republic

Corresponding author: Jakub Kůdela (jakub.kudela@vutbr.cz)

This work was supported in part by the Ministry of Education, Youth and Sports of the Czech Republic INTER-COST Project under Grant LTC18053, and in part by the Project "Computer Simulations for Effective Low-Emission Energy" by Operational Program Research, Development and Education, Priority Axis 1: Strengthening Capacity For High-Quality Research under Grant CZ.02.1.01/0.0/0.0/16_026/0008392.

ABSTRACT In this paper, we describe an effective algorithm for handling chance constrained optimization problems, called the Pool & Discard algorithm. The algorithm utilizes the scenario approximation framework for chance constrained optimization problems, and the warm-start and problem modification features of modern solvers. The exploitation of the problem structure and efficient implementation allows us to considerably speed up the computations, especially for large instances, when compared with conventional methods.

INDEX TERMS Chance constrained programming, scenario approximation, P&D algorithm, stochastic programming, constraint removal.

I. INTRODUCTION

This article describes a novel method for handling chance constrained optimization problems that was developed in the author's dissertation [1]. The introduction into the topic of chance constrained optimization is derived (more or less directly) from [2] – with most of the used notation adapted from [2] 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:

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

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

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

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

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

where y is a scalar variable.

In the CCP_ϵ (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 [3]. In [3], however, only individual chance constraints were considered. Joint probabilistic constraints, as in (1), were first considered by Miller and Wagner, [4], in an independent context, while a general theory is due to Prékopa, see [5], [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Sun-Yuan Hsieh.

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. The books [7] and [8] 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 [9], while convex approximations of chance constrained problems are considered in [10], [11], and [12]. Stability of the solution under perturbation of the chance constrained problem is studied in [13] and [14]. Although chance constrained problems can be efficiently solved in some special cases, 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.

II. SAMPLE COUNTERPART APPROACH

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

Assumption 2.1 requires convexity only with respect to the design variable x, while generic nonlinear dependence with respect to ξ is allowed.

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 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 [15]. Any such solution is here named an "ϵ-level" solution:

Definition 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. The approach utilized in this paper uses a surrogate model called "Scenario 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 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

$$\begin{aligned} \mathrm{SDP}_S : \underset{x \in \mathcal{X}}{\text{minimize}}\ & c^T x \\ \text{subject to } & g(x, \xi^i) \leq 0, \quad i = 1, \ldots, S. \end{aligned} \tag{2}$$

The acronym SDP_S refers to the fact that (2) is a convex program with S constraints. Here we assume the following technical condition on the scenario problem:

Assumption 2.5 (Feasibility): For all possible extractions $\xi^1, \ldots, \xi^S$, the optimization problem (2) is either infeasible, or, if feasible, it attains a unique optimal solution.

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 [16].

The relationship between the number of sampled scenarios S and the probability of violation of the optimal solution to corresponding Scenario Design Problem $\mathcal{V}(\hat{x}_S)$ was investigated in [17] and [18] – for a chosen ϵ we can always find S large enough such that $\hat{x}_S$ is ϵ-level feasible for the original problem (1) with arbitrarily high confidence. There is, however, no guarantee, that the resulting optimal objective value of (2) will be anywhere close to the true optimal value J^*_ϵ. The results derived in [18] show that the distribution function of $\mathcal{V}(\hat{x}_S)$ is bounded by a beta distribution with parameters n_x and $S - n_x + 1$, and imposing that $\mathcal{V}(\hat{x}_S) \leq \epsilon$ holds with high confidence implies that $\mathcal{V}(\hat{x}_S)$ will be much less than ϵ in many cases, resulting in a conservative solution.

Next we introduce a concept that is crucial for the success of the Pooling part 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 2.6 (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 [18] or in a different form in [19], gives us the bound on the number of support scenarios:

Theorem 2.7 (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. The first main contribution of this paper is an efficient way of solving (2), with the use of Theorem 2.7.

III. POOLING PART OF THE POOL & DISCARD ALGORITHM

The idea behind the Pooling part of the algorithm is the following: if one were to verbally describe the problem (2), 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 (2) depends upon is very small, when compared to the overall number of constraints (or scenarios).

The method consists of solving (2) by the following procedure. First, we start by completely neglecting the constraints in (2) 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.

The Pooling part can be summarized as follows:

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

Step 1. Solve the following problem:

$$\begin{aligned} &\underset{x\in\mathcal{X}}{\text{minimize}}\ c^T x \\ &\text{subject to } g(x, \xi^i) \leq 0, \quad i \in \mathcal{I}, \end{aligned} \tag{3}$$

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

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

$$s^i = g(\hat{x}, \xi^i), \quad i \in \{1, \ldots, S\}. \tag{4}$$

Step 3. If $\max\limits_{i\in\{1,\ldots,S\}} s^i > 0$, find the associated index of the maximum value $\hat{i} = \underset{i\in\{1,\ldots,S\}}{\text{argmax}}\ s^i$, add it to the set $\mathcal{I}$ and return to Step 1. Otherwise, set $x^* = \hat{x}, \mathcal{I}^* = \mathcal{I}$ and terminate.

It is important to remark that by the end of this procedure, we not only get the optimal solution of (2), but also an index set $\mathcal{I}$ that contains the support scenarios – this will be very significant for the success of the Discarding part of the P&D algorithm.

Another equally important remark concerns the efficient implementation of the algorithm. In Step 1, we are sequentially solving problems that are extremely similar, only differing in a single constraint. The use of warm-starts (when made possible by a proper choice of solution method) or even problem modification[1] (when supported by our choice of a solver) have immense effect on the efficiency of the Pooling part. For the implementation of the numerical examples that are investigated in this paper, we have chosen the JuMP package [20] for modeling optimization in the Julia language [21] and the CPLEX 12.7 solver [22]. This combination allowed us to use the algorithm to its full extent.[2] The machine, on which we conducted the numerical examples, was a PC with 3.6 GHz AMD Ryzen 5 2600X Six-Core CPU, 32 GB RAM, NVIDIA GeForce GTX 1050 Ti, running on 64-bit Windows 10.

A. NUMERICAL EXAMINATION – ASSET ALLOCATION PROBLEM

The first numerical example we chose to demonstrate the utility of the Pooling part of the P&D algorithm is the (by now, almost canonical) asset allocation problem [8]. 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:

$$\begin{aligned} &\underset{x\geq 0, t\in\Re}{\text{maximize}}\ t \\ &\text{subject to } \mathcal{P}\{t \leq \sum_{j=1}^{n} r_j x_j\} \geq 1-\epsilon, \\ &\qquad \sum_{i=j}^{n} x_j \leq 1. \end{aligned} \tag{5}$$

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 [23], we know that the feasible set of a scalar chance constraint

$$\mathcal{P}\{a^T x \leq b\} \geq 1-\epsilon,$$

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

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

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,

[1] https://www.ibm.com/support/knowledgecenter/SSSA5P_12.5.0/ilog.odms.cplex.help/CPLEX/OverviewAPIs/topics/Modify.html

[2] The implementation of all of the presented numerical examples can be found on the authors GitHub: https://github.com/JakubKudela89

the problem (5) can be transformed [8] into the following second order cone problem (SOCP, see [16]):

$$\begin{aligned} &\underset{x\geq 0, t\in\Re}{\text{maximize}}\ t \\ &\text{subject to} \sum_{j=1}^{n} \mu_j \cdot x_j \geq t \\ &\qquad +\Phi^{-1}(1-\epsilon)\cdot||(\sigma_1\cdot x_1,\ldots,\sigma_n\cdot x_n)||_2, \\ &\sum_{j=1}^{n} x_i \leq 1, \end{aligned} \tag{6}$$

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 scenario approach, 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 (5) has the following form:

$$\begin{aligned} &\underset{x\geq 0, t\in\Re}{\text{maximize}}\ t \\ &\text{subject to } t \leq \sum_{j=1}^{n} r_j^i x_j, \quad i \in \{1,\ldots,S\} \\ &\qquad \sum_{j=1}^{n} x_j \leq 1. \end{aligned} \tag{7}$$

First of all, we will investigate on (7) the dependence of computation time of the Pooling part (CTPP) of the P&D algorithm for varying number of assets n and scenarios S. Additionally, we provide the computation time for the Pooling part without the use of warm-starts and problem modification (CnoWS) and the computation time for solving the problem (7) conventionally (CTC), i.e., passing it to the solver (CPLEX) with all the scenarios.

The results of the computations are summarized in Table 1 and clearly demonstrate the effectiveness of the Pooling part of the P&D algorithm. As the number of scenarios grows, CTPP grows very slowly when compared to CTC, becoming over 20 times faster for the largest number of considered scenarios. The main factor in the effectiveness of the Pooling part is the low growth in the number of iterations needed to solve the problems with more scenarios – this should not be too surprising, since the number of support scenarios stays the same (for the same n). The variant without warm-start or problem modification CnoWS eventually (for high values of n) suffers from too big of an overhead when constructing the corresponding optimization problem, but can still outperform CTC in large number of instances.

TABLE 1. Results of the computation. Average over 10 runs.

n	S	CTC [s]	CnoWS [s]	CTPP [s]	iterations
10	100	0.002	0.008	0.002	12.7
	1,000	0.012	0.013	0.004	18.9
	5,000	0.062	0.011	0.005	17.4
	10,000	0.125	0.014	0.005	15.3
	20,000	0.301	0.013	0.005	13.5
	50,000	0.920	0.024	0.020	11.9
	100,000	1.897	0.033	0.044	10.7
20	100	0.004	0.013	0.005	17.3
	1,000	0.021	0.030	0.008	29.3
	5,000	0.105	0.044	0.011	37.3
	10,000	0.239	0.057	0.014	40.4
	20,000	0.557	0.079	0.026	46
	50,000	1.928	0.106	0.071	49.9
	100,000	4.169	0.137	0.192	50.5
30	100	0.003	0.021	0.004	21.8
	1,000	0.030	0.046	0.011	38.4
	5,000	0.158	0.077	0.016	50.9
	10,000	0.299	0.109	0.028	54.2
	20,000	0.866	0.131	0.054	59.1
	50,000	2.816	0.164	0.102	67.8
	100,000	6.165	0.230	0.192	70.5
50	100	0.004	0.031	0.006	26.3
	1,000	0.036	0.093	0.018	51.7
	5,000	0.239	0.198	0.031	72.6
	10,000	0.708	0.234	0.048	73.2
	20,000	1.496	0.251	0.142	83.3
	50,000	4.542	0.359	0.163	92.6
	100,000	10.088	0.492	0.472	98.9
100	100	0.008	0.078	0.012	36.1
	1,000	0.078	0.322	0.039	78.6
	5,000	0.648	0.596	0.072	104.9
	10,000	1.374	0.796	0.136	118.6
	20,000	3.328	0.936	0.181	129.8
	50,000	9.255	1.234	0.427	142.6
	100,000	21.359	1.698	0.686	155.9
200	100	0.012	0.216	0.027	48.1
	1,000	0.172	1.197	0.105	114.3
	5,000	1.421	2.350	0.223	161
	10,000	3.125	2.872	0.418	174.7
	20,000	7.236	3.911	0.534	195.9
	40,000	16.130	5.066	0.864	215.7
	80,000	36.252	6.525	1.547	233.8
300	100	0.021	0.426	0.046	56.7
	1,000	0.291	2.363	0.194	136.6
	5,000	2.538	5.368	0.515	197.8
	10,000	5.222	7.187	0.779	223.2
	20,000	11.847	9.573	1.149	248.1
	40,000	26.222	12.496	1.670	272.5
	80,000	58.863	15.490	2.924	296.6
500	100	0.039	0.890	0.073	62.1
	1,000	0.603	6.538	0.473	173.2
	5,000	5.094	16.820	1.292	261.3
	10,000	11.616	23.065	1.805	295.3
	20,000	22.105	28.940	2.699	324.6
	50,000	66.267	42.398	4.929	371
1,000	100	0.084	2.806	0.201	75.2
	1,000	1.311	29.882	1.894	239
	5,000	14.877	97.380	6.016	385.4
	10,000	32.063	134.708	8.987	440.4
	20,000	65.047	174.532	12.640	492.6
	50,000	155.843	244.268	20.227	561.8

IV. CONSTRAINT REMOVAL ALGORITHM

If all the S constraints are enforced, however, one cannot expect that good approximations of chance constrained solutions are obtained. To get a less conservative solution we use the framework introduced in [2] for relaxing problem (7). Their approach allows us to remove k constraints out of the S scenario constraints. A general removal procedure is formalized in the following definition:

Definition 4.1 (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

$$\begin{aligned} \mathrm{SDP}^{\mathcal{A}}_{S,k} : \underset{x\in\mathcal{X}}{\text{minimize}}\ & c^T x \\ \text{subject to } & g(x, \xi^i) \leq 0, \\ & i \in \{1, \ldots, S\} \setminus \mathcal{A}\{\xi^1, \ldots, \xi^S\}, \end{aligned} \tag{8}$$

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

Assumption 4.2 (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 [2]) 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 4.3 (Feasibility): Let $\beta \in (0, 1)$ be any small confidence parameter value. If S and k are such that

$$\binom{k+n_x-1}{k} \sum_{i=0}^{k+n_x-1} \binom{S}{i} \epsilon^i (1-\epsilon)^{S-i} \leq \beta, \tag{9}$$

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 [2]).

Theorem 4.4 (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

$$\begin{aligned} \binom{k+n_x-1}{k} \sum_{i=0}^{k+n_x-1} \binom{S}{i} \epsilon^i (1-\epsilon)^{S-i} \\ + \sum_{i=k+1}^{S} \binom{S}{i} (\epsilon-\nu)^i (1-\epsilon+\nu)^{S-i} \leq \beta, \end{aligned} \tag{10}$$

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 [24], [25] and [26]:

$$\begin{aligned} \underset{x\in\mathcal{X}}{\text{minimize}}\ & c^T x \\ \text{subject to } & g(x, \xi^i) - M z_i \leq 0, \quad i = 1, \ldots, S, \\ & \sum_{i=1}^{S} z_i \leq k, \quad z \in \{0, 1\}^S. \end{aligned} \tag{11}$$

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 (2) and (11) are equivalent. By construction, problem (11) provides a framework for optimally selecting the constraints to be removed based on the inequality (10). However, solving (11) may be computationally challenging due to the increase in complexity from (2) to (11) 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 [27] and [28]) for some special structured problems, that significantly improve upon the formulation (11).

V. POOL & DISCARD ALGORITHM

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 4.3), or is terminated once an estimate of the probability of violation of obtained solution $\mathcal{V}(x)$ reaches certain threshold. This approach is almost identical to the one discussed in [29] (called greedy constraint removal), with the distinction that our algorithm utilizes the Pooling step and uses warm-starts (primarily utilizing $\mathcal{I}$) throughout the iterations and as such can be rather effective (this will be demonstrated in the following sections). The P&D algorithm 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:

$$\begin{aligned} \underset{x\in\mathcal{X}}{\text{minimize}}\ & c^T x \\ \text{subject to } & g(x, \xi^i) \leq 0, \\ & i \in \{1, \ldots, S\} \setminus \{i_r \cup \mathcal{I}_p\}, \end{aligned} \tag{12}$$

using the Pooling part, warm-started by using $\mathcal{I} = \mathcal{I}^* \setminus \{i_r\}$ and $x = x^*$. Denote the solution to (12)

as $x^*_{i_r}$, its optimal objective function value $v^*_{i_r}$ and its final set of scenarios $\mathcal{I}^*_{i_r}$.

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

The parameter γ can be, in theory, set to 0 – what discourages us from doing so are the implementation issues of numerical computing. 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} (the same goes for the slacks in the active constraints). 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^{-6}. It should be added, that Step 2. of the Discarding part can be fully parallelized to work more efficiently on multi-core machines or distributed computing environments.

A. NUMERICAL EXAMINATION – ASSET ALLOCATION PROBLEM CONTINUED

We will return to the same problem structure (7) again and 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 $k = \lfloor \epsilon S \rfloor$ scenarios – note that this choice does not guarantee, that the resulting solution obtained by the P&D algorithm will be a ϵ-level feasible, not to mention having the objective value close to the optimal value objective J^*_ϵ.

To examine the effect of the warm-start in the discarding part (using the best solution x^* and the index set $\mathcal{I}^*$ from the previous iteration), we will first compare the computational times of the P&D algorithm, an algorithm that uses just the Pooling part without warm-starts (denoted as "PnoD"), and an algorithm that uses neither Pooling nor Discarding (denoted as "noPnoD", which is essentially the one used in [29]), on a small-scale example ($n = 20, \epsilon = 0.01$).

The comparison is summarized in Table 2 – the utilization of Pooling and the warm-starts in Discarding combined provide immense computational savings compared to the other two methods (while arriving at the exact same solution). To further compare the effectivity of the P&D algorithm, we set the parameters n, S, and k to the same values that can be found in [29] and compare the computation times directly (although they used different distributions for the asset returns, the problem structure is exactly the same).

TABLE 2. Comparing the algorithms. Average over 10 runs.

n	ϵ	S	k	noPnoD [s]	PnoD [s]	P&D [s]
20	0.01	100	1	0.02	0.03	0.01
		1,000	10	2.35	1.10	0.23
		2,500	25	15.60	3.59	0.68
		5,000	50	78.19	8.90	1.66
		10,000	100	371.48	22.79	3.71
		20,000	200	1,931.58	59.14	8.64

The results of the computation are reported in Table 3 – for $n = 20$, the results reported in [29] are comparable with the noPnoD variant of the algorithm, with slight improvement that is most likely caused by a more powerful machine and a newer version of the optimization solver. In the largest instance, the P&D algorithm was more that 200 times faster. For the $n = 200$, the authors in [29] used a random scenario removal strategy, instead of the greedy one (removing one of the support scenarios at random, instead of the one whose removal decreased the optimal objective value the most) – this algorithm is $O(n)$ times faster than the greedy one, but results in an inferior solution. In this setting, the P&D variant with randomized removal (denoted as P&D*) was almost 500 times faster than the one in [29].

The real crux of the matter, however, is the following: "How good a solution (in terms of ϵ-level 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 (6). 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 (6)) was 1.0309. The results are summarized in Table 4, Figure 1 and Figure 2. Using the formula for the needed number of scenarios from [18], with $\beta = 10^{-10}$, we get that to obtain a feasible solution to this problem with high probability $(1 - \beta)$, we need to solve (7) with at least $S = 8{,}547$ scenarios (without any discarding). The solution to this problem had the objective value 1.0179 (third column of Table 4), with the probability of violation 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 (fifth column of the table), but the corresponding probability of violation increased 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 (seventh 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 probability of violation 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"

TABLE 3. Comparing the different algorithms. The results with a * are for random scenario removal. Average over 5 runs.

n	S	k	[29] results [s]	noPnoD [s]	PnoD [s]	P&D [s]	P&D* [s]
20	2,500	18	15.1	11.15	2.56	0.49	0.04*
	5,000	76	138.0	117.16	13.63	2.36	0.18*
	10,000	220	875.3	772.93	47.03	6.99	0.54*
	20,000	582	5,412.4	5,315.1	176.28	23.44	1.73*
200	20,000	582	5,521*	-	-	3,486.5	44.3*
	40,000	1,164	26,307*	-	-	8,312.4	126.3*
	80,000	2,328	120,535*	-	-	19,521.7	241.7*

TABLE 4. The "quality" of the solutions produced by the P&D algorithm, $n = 30$, target $\epsilon = 0.01$, optimal objective $J^*_\epsilon = 1.0309$. Varying number of scenarios, single run of the algorithm.

S	k	RSO	RSPV	ODϵS	PVDϵS	ASO	NRS
500	5	1.0324	0.0291	1.0400	0.0466	–	–
1,000	10	1.0292	0.0183	1.0377	0.0368	–	–
2,000	20	1.0289	0.0120	1.0336	0.0220	–	–
5,000	50	1.0231	0.0041	1.0326	0.0160	1.0303	31
8,547	85	1.0179	0.0029	1.0318	0.0138	1.0291	53
20,000	200	1.0167	0.0014	1.0319	0.0128	1.0304	147
50,000	500	1.0140	0.0004	1.0310	0.0111	1.0305	463
100,000	1,000	1.0129	0.0003	1.0309	0.0102	1.0308	980
250,000	2,500	1.0101	0.0001	1.0308	0.0101	1.0308	2,487

RSO – "robust" solution optimal objective value (no scenarios removed),
RSPV – "robust" solution probability of violation,
ODϵS – optimal objective value after discarding $\lfloor \epsilon S \rfloor$ scenarios,
PVDϵS – probability of violation after discarding $\lfloor \epsilon S \rfloor$ scenarios,
ASO – admissible solution optimal objective value,
NRS – number of removed scenarios for admissible solution.

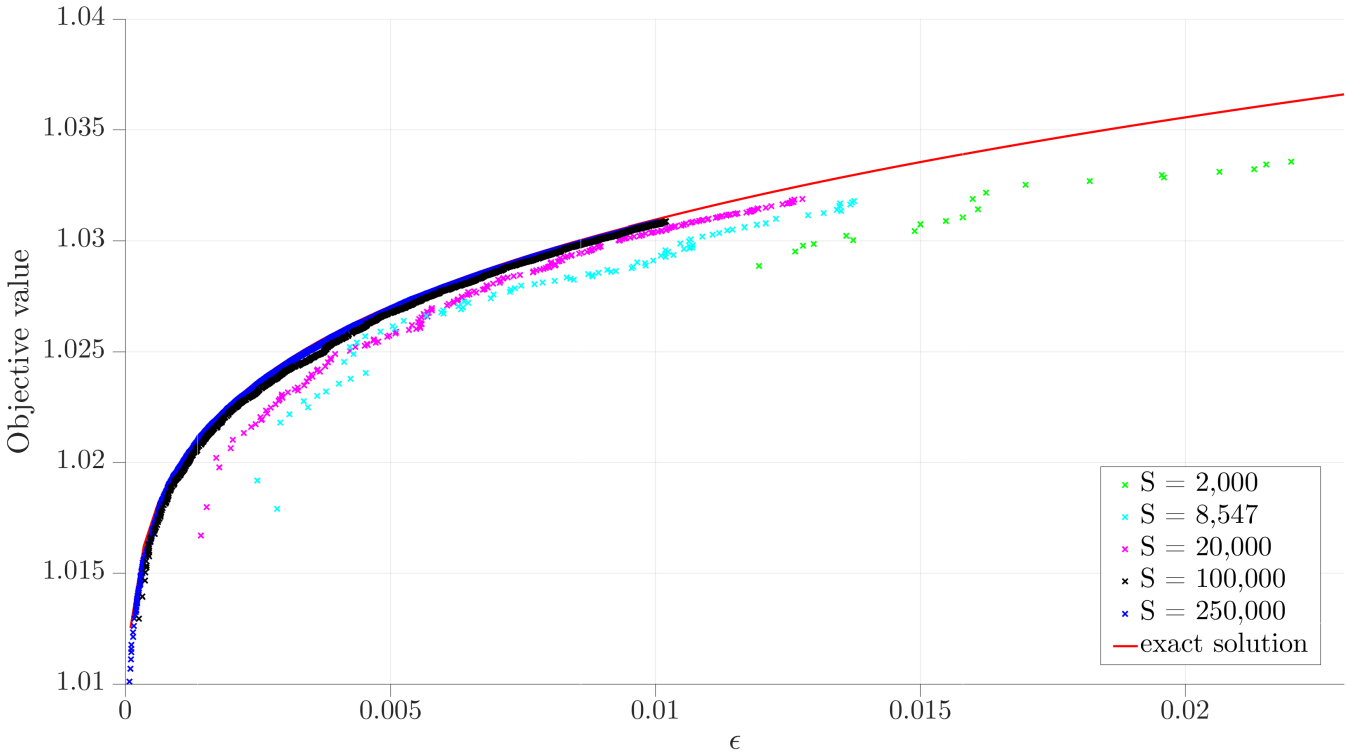

FIGURE 1. The "quality" of the solutions produced by the P&D algorithm, $n = 30$. Varying number of scenarios, single run of the algorithm.

solution objective (the solution after the Pooling part) and smaller corresponding probability of violation (both of these are rather intuitive). Secondly, when we increase the number of scenarios, the probability of violation 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 (6).

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 1, where we can see the progression of the

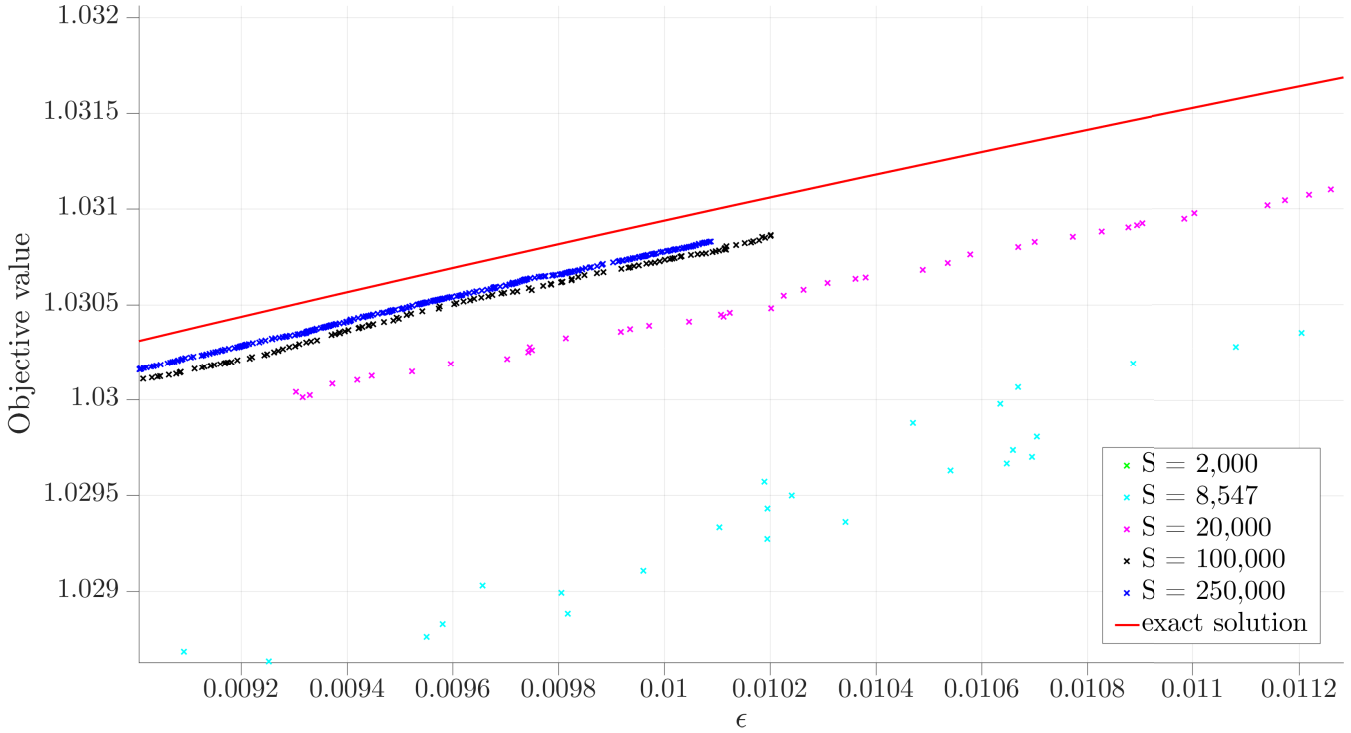

FIGURE 2. The "quality" of the solutions produced by the P&D algorithm, $n = 30$. Varying number of scenarios, single run of the algorithm. Close up on $\epsilon = 0.01$.

TABLE 5. Results of the computations, $n = 20$. Average over 10 runs.

	$\epsilon = 0.02$ $J^*_\epsilon = 1.0280$			$\epsilon = 0.05$ $J^*_\epsilon = 1.0364$			$\epsilon = 0.15$ $J^*_\epsilon = 1.0517$		
S	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]
100	1.0456	0.1465	<0.1	1.0497	0.1859	0.1	1.0601	0.2584	0.2
1,000	1.0322	0.0411	0.4	1.0387	0.0700	1.0	1.0528	0.1709	2.9
2,500	1.0299	0.0295	1.4	1.0374	0.0602	3.1	1.0512	0.1548	8.0
5,000	1.0286	0.0237	3.2	1.0363	0.0539	6.7	1.0509	0.1508	17.5
10,000	1.0284	0.0225	7.5	1.0361	0.0513	15.3	1.0510	0.1514	39.0
20,000	1.0280	0.0211	17.2	1.0363	0.0514	37.7	1.0510	0.1516	93.7
50,000	1.0279	0.0205	53.3	1.0360	0.0504	121.7	1.0508	0.1505	317.2
100,000	1.0278	0.0202	146.0	1.0359	0.0500	335.6	1.0507	0.1497	914.4
250,000	1.0278	0.0200	854.5	1.0360	0.0500	2,058.9	1.0508	0.1502	5,987.6

ODϵS – optimal objective value after discarding $\lfloor \epsilon S \rfloor$ scenarios,
PVDϵS – probability of violation after discarding $\lfloor \epsilon S \rfloor$ scenarios.

TABLE 6. Results of the computations, $n = 30$. Average over 10 runs.

	$\epsilon = 0.02$ $J^*_\epsilon = 1.0355$			$\epsilon = 0.05$ $J^*_\epsilon = 1.0433$			$\epsilon = 0.10$ $J^*_\epsilon = 1.0511$		
S	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]
100	1.0526	0.1785	<0.1	1.0585	0.2159	0.1	1.0636	0.2692	0.2
1,000	1.0406	0.0487	0.8	1.0457	0.0737	1.9	1.0527	0.1285	3.8
2,500	1.0375	0.0310	2.6	1.0441	0.0607	6.2	1.0514	0.1095	11.2
5,000	1.0369	0.0270	6.3	1.0434	0.0549	14.7	1.0512	0.1062	25.7
10,000	1.0357	0.0223	15.0	1.0434	0.0531	32.8	1.0508	0.1034	57.0
20,000	1.0356	0.0215	33.9	1.0430	0.0507	76.4	1.0508	0.1022	137.3
50,000	1.0354	0.0205	102.1	1.0430	0.0507	226.3	1.0506	0.1006	411.7

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 (6) for different values of ϵ (called "exact solution" in the legend of Figure 2).

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.

The relationship between computational times, number of scenarios S, number of variables n and chosen probability of violation ϵ is further investigated in Tables 5, 6 and 7. From these results we can see that when we increase ϵ, the computation times increase linearly. The same cannot be said for when increasing S – the number of scenario

TABLE 7. Results of the computations, $\epsilon = 0.02$. Average over 10 runs.

	$n = 50$ $J^*_\epsilon = 1.0442$			$n = 100$ $J^*_\epsilon = 1.0544$			$n = 200$ $J^*_\epsilon = 1.0630$		
S	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]	ODϵS	PVDϵS	t [s]
100	1.0624	0.2130	0.1	1.0749	0.3086	0.2	1.0808	0.3758	0.7
1,000	1.0494	0.0563	2.2	1.0608	0.0775	9.0	1.0702	0.1030	36.4
2,500	1.0466	0.0346	7.5	1.0576	0.0448	34.3	1.0670	0.0592	140.8
5,000	1.0456	0.0286	19.6	1.0566	0.0351	83.6	1.0652	0.0422	430.6
10,000	1.0450	0.0250	45.8	1.0557	0.0282	204.8	1.0642	0.0308	1,034.4
20,000	1.0444	0.0224	97.5	1.0551	0.0244	444.2	1.0639	0.0268	2,626.2
50,000	1.0442	0.0209	289.1	1.0545	0.0214	1,351.4	1.0633	0.0226	7,662.8

removals and computational times of the pooling steps grow simultaneously, resulting in a superlinear increase computational time. Similar (and more impactful) behaviour can be observed in Table 7, where the number of variables n changes – this results in a larger number of possible support scenarios and larger solution times for the successive optimization problems.

When we increase the number of scenarios S, the resulting solutions (after discarding $\lfloor \epsilon S \rfloor$ scenarios) get very close to the true optimum J^*_ϵ, although it was not guaranteed by any theory. In similar fashion, the probability of violation of the solutions get very close to ϵ.

VI. NONLINEAR JOINT CHANCE CONSTRAINED 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 [30] and [31]. 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 [30], the constraints need to be convex in x and the problem can be a joint chance constrained one. In the method described in [31], 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:

$$\begin{aligned} \underset{x \geq 0}{\text{minimize}} \; & -\sum_{j=1}^{n} x_j \\ \text{subject to } & \mathcal{P}\{\sum_{j=1}^{n} \xi_{ij}^2 x_j^2 - b \leq 0, \; i = 1, \ldots, m\} \\ & \geq 1 - \epsilon, \end{aligned} \tag{13}$$

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 [31], $m = 1$ (a single chance constraint).

Optimal solution x^* of the problem (13), derived in [30], is:

$$x_1^* = x_2^* = \cdots = x_n^* = \left[b / F^{-1}_{\chi^2_n}((1-\epsilon)^{\frac{1}{m}}) \right]^{\frac{1}{2}}, \tag{14}$$

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

Because of the nature of the problem (quadratic and convex), we were able to use the CPLEX solver, and utilize the problem modification feature again. We start the numerical examination with the same setting as [31]: $n = 10$, $m = 1$, $b = 10, \epsilon = 0.05$. The parameter γ that controls the selection of scenarios to discard, was set to 10^{-3}. Using the formula (14), 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 $(1 - \epsilon)$ of the obtained solution using 10^5 new scenarios. The results of the computations are summarized in Table 8.

TABLE 8. Results of the computation. $J^*_\epsilon = -7.390$. Average values over 10 runs.

S	ODϵS	reliability	P&D [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

Unsurprisingly, the more scenarios are taken into account, the better (closer to the theoretical optimum) the result. The computational time is quite good, considering [31] report around 500 s as the computational time for their algorithm (that uses 500 scenarios and reports the optimal objective value -7.627) and the big-M mixed-integer formulation does not converge in an hour [31] (again, using "just" 500 scenarios).

The second setting we investigate is from [30]: $n = 10$, $m = 10$, $b = 100$, $\epsilon = 0.1$. The optimal objective value, using (14), is -20.82. Note that in this setting we are dealing with a "proper" joint chance constraint problem, with nonlinear (but convex) constraint functions. In the Pooling part of the algorithm, when we find a scenario with a violated constraint, we have a choice of either adding all of the m constraints that correspond to this scenario to the problem, or to add only the violated ones. In our implementation we chose the former, since it corresponds more closely to the description of the P&D algorithm we gave in the previous sections, although additional computational savings could be gained by properly implementing the latter approach. 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 Table 9.

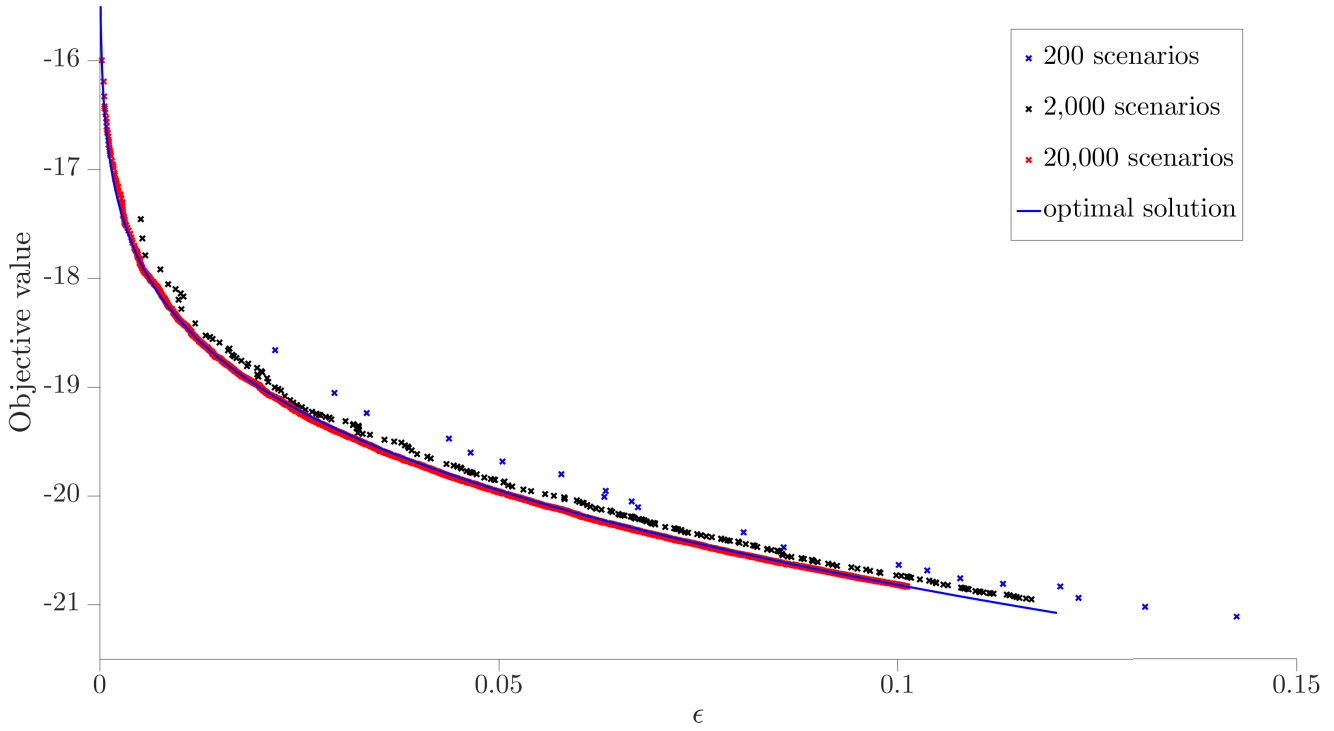

FIGURE 3. Approximation of trade-off between reliability and optimal objective value. Nonlinear joint chance constrained example.

TABLE 9. Results of the computation. $J^*_\epsilon = -20.82$. Average values over 10 runs.

S	ODϵS	reliability	P&D [s]
100	-21.46	0.8042	6.67
251	-21.42	0.8305	15.14
631	-21.03	0.8720	51.24
1,585	-20.96	0.8853	162.21
3,981	-20.84	0.8962	494.94
10,000	-20.83	0.8975	1,670.65

To compare the results with the ones achieved in [30], 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. It is important to emphasize again 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 (1-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 3 – it shows just one run of the P&D algorithm for different number of scenarios and the optimal values computed using the formula (14). 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 lower than the desired level $1 - \epsilon$ 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 6 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.

VII. CONCLUSION

The main advantage of the P&D algorithm lies in the exploitation of the structure of the scenario design problem, which is done on two levels:

- The Pooling part of the P&D algorithm utilizes the fact that the number of support scenarios is usually very small compared to the number of sampled scenarios. By iteratively solving much smaller problems we can get the solution faster and need less memory, than we would need to solve the scenario design problem with all scenarios at once (compare the columns CTC and CTPP in Table 1).
- The Discarding part of the algorithm fully utilizes the set $\mathcal{I}$ that contains the current support scenarios, to find the ones that will be discarded (either be the greedy or the randomized algorithm). Since it only solves comparatively much smaller problems (and, each scenario removal should terminate in just a few iterations), it brings additional computational savings (compare the columns PnoD and P&D in Table 2). The combined effect of the two parts of the algorithm is best seen

in the difference between columns noPnoD and P&D in Table 2.

The numerical examinations show that P&D algorithm provides a powerful framework for handling certain types of chance constrained optimization problems. When compared with conventional approaches [29] on a linear example (see Table 3), it was several hundred times more efficient in the largest instances. On the nonlinear examples, it was on par with the state-of-the-art methods [31] and [30].

Further investigation are possible – in the Pooling part of the algorithm for joint chance constrained problems, the choice between including all constraints for a violated scenario, or just the ones that are violated, could bring additional computational savings. Also, the use of P&D (or just the use of Pooling) could be applied in other classes of convex optimization problems (e.i., semi-definite problems in control) that need to be set in chance constrained (or robust) setting and require the consideration of large number of scenarios.

REFERENCES

[1] J. Kudela, "Advanced decomposition methods in stochastic convex optimization," Ph.D. dissertation, Inst. Math., Brno Univ. Technol., Brno, Czechia, 2019.

[2] M. C. Campi and S. Garatti, "A Sampling-and-Discarding approach to chance-constrained optimization: Feasibility and optimality," *J. Optim. Theory Appl.*, vol. 148, no. 2, pp. 257–280, Feb. 2011.

[3] A. Charnes, W. W. Cooper, and G. H. Symonds, "Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil," *Manage. Sci.*, vol. 4, no. 3, pp. 235–263, Apr. 1958.

[4] R. Jagannathan, "Chance-constrained programming with joint constraints," *Oper. Res.*, vol. 22, no. 2, pp. 358–372, Apr. 1974.

[5] A. Prekopa, "On probabilistic constrained programming," in *Proc. Princeton Symp. Math. Program.*, 1970, pp. 1–8.

[6] A. Prékopa, "Contributions to the theory of stochastic programming," *Math. Program.*, vol. 4, no. 1, pp. 202–221, Dec. 1973.

[7] A. Prékopa, *Stochastic Programming*, 1st ed. Norwell, MA, USA: Kluwer, 1995.

[8] A. Ruszczynski and A. Shapiro, *Stochastic Programming* (Handbooks in Operations Research and Management Science), 1st ed. Amsterdam, The Netherlands: Elsevier, 2003, vol. 10.

[9] R. Henrion and C. Strugarek, "Convexity of chance constraints with independent random variables," *Comput. Optim. Appl.*, vol. 41, no. 2, pp. 263–276, Nov. 2008.

[10] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust Optimization*, 1st ed. Princeton, NJ, USA: Princeton Univ. Press, 2009.

[11] A. Nemirovski, "On safe tractable approximations of chance constraints," *Eur. J. Oper. Res.*, vol. 219, no. 3, pp. 707–718, Jun. 2012.

[12] A. Nemirovski and A. Shapiro, "Convex approximations of chance constrained programs," *SIAM J. Optim.*, vol. 17, no. 4, pp. 969–996, Jan. 2007.

[13] R. Henrion and W. Römisch, "Metric regularity and quantitative stability in stochastic programs with probabilistic constraints," *Math. Program.*, vol. 84, no. 1, pp. 55–88, Jan. 1999.

[14] R. Henrion and W. Römisch, "HÃlder and Lipschitz stability of solution sets in programs with probabilistic constraints," *Math. Program.*, vol. 100, no. 3, pp. 589–611, 2004.

[15] B. R. Barmish and P. S. Shcherbakov, "On avoiding vertexization of robustness problems: The approximate feasibility concept," *IEEE Trans. Autom. Control*, vol. 47, no. 5, pp. 819–824, May 2002.

[16] S. P. Boyd and L. Vandenberghe, *Convex Optimization*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[17] G. Calafiore and M. C. Campi, "Uncertain convex programs: Randomized solutions and confidence levels," *Math. Program.*, vol. 102, no. 1, pp. 25–46, Jan. 2005.

[18] M. C. Campi and S. Garatti, "The exact feasibility of randomized solutions of uncertain convex programs," *SIAM J. Optim.*, vol. 19, no. 3, pp. 1211–1230, Jan. 2008.

[19] V. L. Levin, "Application of E. Helly's theorem to convex programming, problems of best approximation and related questions," *Math. USSR-Sbornik*, vol. 8, no. 2, pp. 235–247, Feb. 1969.

[20] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Rev.*, vol. 59, no. 2, pp. 295–320, Jan. 2017.

[21] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, Jan. 2017.

[22] *IBM ILOG CPLEX Optimization Studio: CPLEX User's Manual. Version 12, Release 7*. IBM Corp, New York, NY, USA, 2016. [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/pdf/usrcplex.pdf

[23] C. M. Lagoa, X. Li, and M. Sznaier, "Probabilistically constrained linear programs and risk-adjusted controller design," *SIAM J. Optim.*, vol. 15, no. 3, pp. 938–951, Jan. 2005.

[24] J. Luedtke and S. Ahmed, "A sample approximation approach for optimization with probabilistic constraints," *SIAM J. Optim.*, vol. 19, no. 2, pp. 674–699, Jan. 2008.

[25] J. Luedtke, S. Ahmed, and G. Nemhauser, "An integer programming approach for linear programs with probabilistic constraints," *Integer Program. Combinat. Optim.*, pp. 410–423, 2010.

[26] B. K. Pagnoncelli, S. Ahmed, and A. Shapiro, "Sample average approximation method for chance constrained programming: Theory and applications," *J. Optim. Theory Appl.*, vol. 142, no. 2, pp. 399–416, Aug. 2009.

[27] Y. Song, J. R. Luedtke, and S. Käkyavuz, "Chance-constrained binary packing problems," *INFORMS J. Comput.*, vol. 26, no. 4, pp. 735–747, Nov. 2014.

[28] S. Ahmed, J. Luedtke, Y. Song, and W. Xie, "Nonanticipative duality, relaxations, and formulations for chance-constrained stochastic programs," *Math. Program.*, vol. 162, nos. 1–2, pp. 51–81, Mar. 2017.

[29] 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," *J. Optim. Theory Appl.*, vol. 155, no. 2, pp. 707–722, Nov. 2012.

[30] F. Shan, L. Zhang, and X. Xiao, "A smoothing function approach to joint chance-constrained programs," *J. Optim. Theory Appl.*, vol. 163, no. 1, pp. 181–199, Oct. 2014.

[31] L. Adam and M. Branda, "Nonlinear chance constrained problems: Optimality conditions, regularization and solvers," *J. Optim. Theory Appl.*, vol. 170, no. 2, pp. 419–436, Aug. 2016.

JAKUB KŮDELA received the M.S. degree in mathematical engineering from the Brno University of Technology, in 2014, and the Ph.D. degree in applied mathematics from the Brno University of Technology in 2019.

Since 2018, he has been a Research Assistant with the Institute of Automation and Computer Science, Brno University of Technology. His research interests include the development of computational methods for various optimization problems and engineering applications.

PAVEL POPELA received the Ph.D. degree in econometrics from Charles University Prague, in 1998. Since 1986, he has been as a Senior Lecturer and a Researcher with the Brno University of Technology. His research interests include stochastic programming models and methods.