

Solving Two-Stage Stochastic Programming problems via Machine Learning

Xiaochen Chou¹[0000-0003-1491-9336], Enza Messina¹[0000-0002-4062-0824], and
Stein W. Wallace²[0000-0001-5909-7909]

¹ University of Milano-Bicocca

Department of Informatics, Systems and Communication

Viale Sarca, 336, Milan, 20125, Italy

² NHH Norwegian School of Economics

Department of Business and Management Science

Helleveien 30, 5045 Bergen, Norway

Abstract. Decision-making under uncertainty addresses real-world problems that are often hard to solve, particularly when involving both discrete and binary decision variables. In two-stage stochastic programming problems, decisions are made before uncertain data is revealed, with adjustments optimized in a recourse problem after the revelation. The ideal solution is robust and resilient across all possible future scenarios. Classic solving techniques rely on selected sets of scenarios to approximate the problem. Accuracy improves with the inclusion of more scenarios, concurrently, the complexity of the problem increases, frequently resulting in instances that are computationally intractable.

With the rapid advancement in machine learning technologies, our study aims to train neural networks as surrogate models for the recourse problem. These networks aim to predict solution quality across diverse scenarios, thereby reducing the complexity of the original optimization model. Through empirical analysis, we present preliminary findings demonstrating the application of this predictive capacity and examine the practical challenges encountered.

Keywords: Stochastic Programming · Neural Network · Machine Learning.

1 Introduction

Stochastic programming (SP) [1] is a framework for modeling optimization problems that involve uncertainty, with a broad range of applications across multiple fields, such as finance [2], transportation [3] and energy optimization [4]. To efficiently solve the SP problems, solutions are typically derived from scenario sets sampled from the stochastic process. It limits the uncertainties to a finite set of potential realizations and identifies robust solutions that perform well across different scenarios, offering resilience decisions in complex problems.

The two-stage formulation is a fundamental model widely used in stochastic programming for various problems. Within a two-stage SP framework, usually a

first-stage decision $x \in \mathcal{X}$ has to be taken based on the information available at the time, while a stochastic process ξ exists and its value can only be observed in the future. Upon revelation of the precise value of the stochastic process, a second-stage decision $y \in \mathcal{Y}$ is made to refine the existing strategy. The feasibility sets $\mathcal{X}$ and $\mathcal{Y}$ are defined by their respective predefined constraints. The space of $\mathcal{Y}$ depends on both the first-stage decision x and the stochastic process ξ . In its general form, a two-stage SP problem can be represented as:

$$\min_{x \in \mathcal{X}} f(x) + \mathbb{E}_{\xi}[Q(x, \xi)] \quad (1)$$

with $f(x)$ being the cost for the first-stage decisions and $Q(x, \xi)$ being the optimal value of the second-stage problem, also known as the *recourse* problem:

$$Q(x, \xi) = \min_{y \in \mathcal{Y}} g(y, \xi) \quad (2)$$

In practice, the stochastic process ξ is commonly conceptualized as a random vector with a finite set of possible realizations. The problem can then be solved using techniques such as Benders Decomposition [5] or by using the Deterministic Equivalent. We denote a finite set of scenarios as $S = \{1, \dots, s\}$, with p_s being the probability of scenario s and ξ_s being the actual values of the stochastic process ξ in scenario s . The SP problem can be discretized as follows:

$$\min_{x \in \mathcal{X}} f(x) + \sum_{s=1}^S p_s Q(x, \xi_s) \quad (3)$$

The quality of the solution relies heavily on the chosen scenarios. To attain a robust and resilient optimal solution, it is imperative to select an appropriate scenario set. Certain approaches provide effective small scenario sets, but may not consistently give favorable scenarios as the set size increases [6]. Following statistical principles, large scenario sets lead to more precise solutions as they converge towards the true values of the underlying distribution. However, this may significantly increase the solving time. In certain instances, even a modest count, such as 20 scenarios, can make the problem challenging to solve. Moreover, when integer variables are involved, Benders Decomposition struggles to converge efficiently [7]. Hence, there arises a necessity to explore alternative solving techniques or to limit the number of scenarios. In this study, we investigate machine learning-enabled solutions.

The rest of the paper is organized as follows. In Section 2, we conduct a literature review on employing machine learning in stochastic programming problems. In Section 3, the formulation of the problem studied in this work is presented. The proposed approach using neural network as a surrogate model is detailed in Section 4. Preliminary experimental results are presented in Section 5. Conclusions are drawn in Section 6.

2 Literature Review

In recent years, the application of machine learning (ML) approaches has notably increased in addressing diverse challenges associated with optimization problems.

In the domain of SP problems, ML predominantly serves in scenario generation, scenario reduction, and the development of surrogate models.

Various scenario generation methodologies have been developed over time due to the significant impact of scenarios on solution quality [8, 9]. Classic methodologies often rely on probability distributions of the stochastic process inferred from historical data or employ empirical approaches using sampled data as prior models for deriving posterior distributions through Bayesian updating procedures [10]. Recent studies show that considering problem structure in scenario generation leads to more concise scenario sets than distribution-based approaches. In [11], a problem-based scenario generation method is proposed that does not rely on the parameters of the stochastic distribution but utilizes the output distribution across various decisions to create informative scenarios. An innovative active learning approach employing Bayesian Optimization is introduced in [12]. The algorithm iteratively constructs a scenario set by incorporating promising new scenarios identified through efficient black-box global optimization, thus identifies the most informative set for the given problem. As the generation process is independent of the probability distribution of the stochastic process, this approach is classified as problem-driven scenario generation [13]. Another example of distribution-free Learning Enabled Optimization can be found in [14].

Parallel to the pursuit of informative scenarios, significant efforts have been dedicated to identifying representative scenarios for scenario reduction [11, 15–17]. In [18], a supervised learning approach is employed to predict representative scenarios. With a surrogate problem representing the deterministic version of the SP problem under various scenarios, a scenario is identified as *representative* if the optimal objective function value of the surrogate model matches with that of the SP problem. Through a regression process involving problem instances and their corresponding representative scenarios, the model minimizes the mean squared error between predicted and actual representative scenarios, enabling close-to-optimal solutions for the original SP problem.

Instead of seeking specific scenarios s or a scenario set S to derive a solution for the original SP problem, alternative surrogate models are constructed using machine learning techniques. A neural network is trained in [19] using first-stage solutions x and scenarios or scenario sets s/S to approximate the expected recourse function value $\mathbb{E}_\xi[Q(x, \xi)]$. The trained neural network can then serve as a surrogate model for the recourse function and be integrated into the original SP problem. Research on embedding a trained neural network with Rectified Linear Unit (ReLU) activation into Mixed Integer Linear Programming (MILP) can be found in [20, 21]. Further use of neural networks for universal approximation has been discussed in [22]. In line with the aforementioned approaches, our work similarly uses neural networks as surrogate models for the recourse problem. Different from training with certain scenarios, our neural network is trained with a theoretically large enough scenario set, enabling it to predict the performance of any given first-stage solution across all possible scenarios. Consequently, the original problem can be solved using the surrogate model, which

provides accurate evaluations based on a large set of scenarios, while introducing no additional computational complexity to the solving procedure.

3 Problem Formulation

The Facility Location Problem (FLP) is a classic optimization problem central to various fields, such as supply chain management, urban planning and healthcare. In this work, we address a variant of the problem, the single-source capacitated facility location problem (SSCFLP) with stochastic demands.

We denote $\mathcal{I}$ as a set of operatable facilities and $\mathcal{J}$ as a set of customers with stochastic demands ξ . The facilities have fixed operational costs f_i and capacities k_i , which may vary across facilities. The transportation cost is determined by the travel distance $t_{i,j}$ between each facility i and customer j , with α denoting the cost per unit distance. The objective is to decide which facilities to operate to fulfil customer demands while minimizing the total operational and expected transportation costs. To model the problem, we introduce two decision variables. Let x_i represent the operational state of facility i and y_{ij} indicate the assignment of customer j to facility i . Both decision variables are binary, taking the value of 1 when the respective condition is met and 0 otherwise. The objective function of the problem is defined as follows:

$$\min \sum_{i \in \mathcal{I}} f(x) + \mathbb{E}_{\xi}[Q(x, \xi)] \quad (4)$$

where $f(x)$ is the fixed first-stage cost, i.e., $f(x) = \sum_{i \in \mathcal{I}} f_i x_i$ and $Q(x, \xi)$ is the optimal value of the recourse problem, that optimizes the assignment of customers to the operatable facilities upon revelation of the actual demand values.

To model the recourse problem, additional costs are included, i.e., penalties, for scenarios wherein demands may exceed available capacities, ensuring feasibility for any first-stage solution. In the SSCFLP, each customer is assigned to a single facility, and that facility is responsible for meeting all the demands of the respective customer. If a customer's demand cannot be met, the loss is penalized in the objective function and no transportation costs are incurred for unfulfilled demands. We denote p as the penalty and introduce a novel binary variable u_j , indicating whether the demand of customer j is penalized. The recourse problem is formulated as follows:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \alpha \cdot t_{i,j} y_{ij} + \sum_{j \in \mathcal{J}} p \cdot u_j \quad (5)$$

s.t.

$$\sum_{i \in \mathcal{I}} y_{ij} + u_j = 1, \quad \forall j \in \mathcal{J} \quad (6)$$

$$\sum_{j \in \mathcal{J}} \xi_j \cdot y_{ij} \leq k_i x_i, \quad \forall i \in \mathcal{I} \quad (7)$$

$$x_i, y_{ij}, u_j \in \{0, 1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (8)$$

The SP problem can be solved on a scenario set $\mathcal{S}$ sampled or constructed from ξ . In each scenario $s \in \mathcal{S}$, the demand for each customer j is deterministic and independent, denoted as ξ_j^s . If we redefine the binary decision variables, with x being the same, as it only associates with the operational status of facilities, while y and u are extended to incorporate scenario dimensions, the Deterministic Equivalent of the SP problem can be rewritten as follows:

$$\min \sum_{i \in \mathcal{I}} f_i x_i + \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \alpha \cdot t_{i,j} y_{ijs} + \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{j \in \mathcal{J}} p \cdot u_{js} \quad (9)$$

s.t.

$$\sum_{i \in \mathcal{I}} y_{ijs} + u_{js} = 1, \quad \forall j \in \mathcal{J}, s \in \mathcal{S} \quad (10)$$

$$\sum_{j \in \mathcal{J}} \xi_j^s \cdot y_{ijs} \leq k_i x_i, \quad \forall i \in \mathcal{I}, s \in \mathcal{S} \quad (11)$$

$$x_i, y_{ijs}, u_{js} \in \{0, 1\}, \quad \forall i \in \mathcal{I}, j \in \mathcal{J}, s \in \mathcal{S} \quad (12)$$

The objective is to identify an optimal solution x that satisfies the constraints across all provided scenarios.

4 Using Neural Networks as surrogate models

4.1 Neural Networks

In this study, we employ neural networks as predictive mechanisms to estimate the recourse function value of the two-stage stochastic programming problem.

Given a first-stage solution x indicating the operational status of facilities, a second-stage problem $Q(x, \xi)$ can be solved to optimality, which entails assigning customer demands to the operational facilities. In the stochastic SSCFLP, as customer demands vary across scenarios, each scenario is independently solved to optimality, and the average objective function value is computed as the expected value for the recourse function associated with a given first-stage solution. According to the law of large numbers, when scenarios are sampled, a higher number of scenarios tends to lead to a more accurate evaluation value. We assume that as the number of scenarios reaches a sufficient threshold, the expectation value converges to stability. This stable value serves as the evaluation value for the performance of the given first-stage solution, thereby forming our training dataset.

A neural network is then trained to capture the problem-specific features for the recourse problem in the stochastic SSCFLP and aimed at providing accurate predictions for the recourse function. i.e., for a given problem instance, the neural network Φ aims to capture the underlying function:

$$\Phi : x \rightarrow \lim_{|\mathcal{S}| \rightarrow +\infty} \mathbb{E}_{s \in \mathcal{S}} [Q(x, \xi_s)] \quad (13)$$

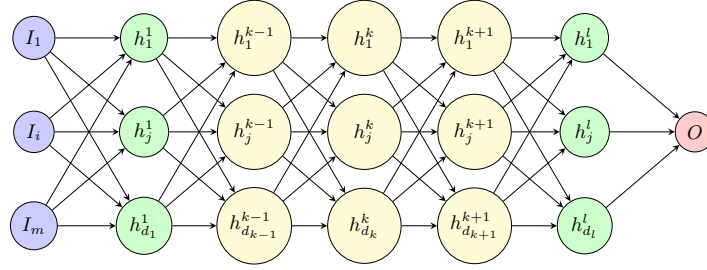


Fig. 1. Illustration of a fully connected neural network

In comparison to a network trained using both first-stage solutions and scenarios to derive the objective function value, the significance of this network lies in its ability not only to learn the problem features but also to comprehend performance variations relative to varying demands (scenarios). Once trained, the network can function as a predictive tool, offering an evaluative metric for any given first-stage solution x :

$$eval(x) = f(x) + \Phi(x) \quad (14)$$

One aspect of this evaluative metric serves as an alternative tool for comparative analysis of proposed first-stage solutions, facilitating the identification of near-optimal solutions through heuristic methods for hard-to-solve instances. Another interest lies in integrating the neural network into the optimization framework of the original SP problem. This use of machine learning models to replace complex processes is commonly referred to as surrogate models.

4.2 Surrogate Models

As discussed in Section 2, numerous studies have investigated embedding a trained neural network with ReLU activation into MILP. In this study, we adopt the formulations outlined in [19, 20]. For simple comprehension, we show an illustration of a fully connected neural network in Figure 1.

In a fully connected neural network with m inputs, l hidden layers and d_k units in each layer, the unit j in the k -th layer can be expressed as follows:

$$h_j^k = \sigma\left(\sum_{i=1}^{d_{k-1}} w_{ij}^{k-1} h_i^{k-1} + b_j^{k-1}\right) \quad (15)$$

with h being the value of a unit in the hidden layer, w being the weight associated with the connections between units, b being the bias and σ being the activation function. In this case, the activation function is given by $\text{ReLU}(a) = \max\{0, a\}$ for $a \in \mathbb{R}$.

To emulate the ReLU output for each unit, a binary variable z_j^k is introduced to impose logical constraints that constrain the output value of the network. The output value is then represented as $\hat{h}_j^k$ instead of h_j^k .

$$\sum_{i=1}^{d_{k-1}} w_{ij}^{k-1} \hat{h}_i^{k-1} + b_j^{k-1} = \hat{h}_j^k - \check{h}_j^k \quad (16)$$

$$z_j^k = 1 \Rightarrow \hat{h}_j^k \leq 0 \quad (17a)$$

$$z_j^k = 0 \Rightarrow \check{h}_j^k \leq 0 \quad (17b)$$

$$\hat{h}_j^k, \check{h}_j^k \geq 0 \quad (17c)$$

$$z_j^k \in \{0, 1\} \quad (17d)$$

The variable z_j^k ensures that at most one of $\hat{h}_j^k$ and $\check{h}_j^k$ is non-zero. Given that $\hat{h}_j^k$ and $\check{h}_j^k$ are both non-negative, constraint (16) ensures that if the sum of the weight plus bias from the previous layer is positive, then $\hat{h}_j^k - \check{h}_j^k \geq 0$, i.e., $\hat{h}_j^k > 0$ and $\check{h}_j^k = 0$; otherwise, $\hat{h}_j^k = 0$. This aligns with the features of ReLU function.

4.3 Solving the MIP with surrogate models

In our problem, the input variables for the neural network are the binary decision variables x_i indicating the operational status of a facility, i.e., $x = [x_1, x_2, \dots, x_m]$. The output value of the network is denoted as $\hat{y}$, representing the predicted value generated by the network Φ . As the neural network acts as a surrogate model for the recourse problem, we replace the entire second-stage problem component with $\hat{y}$, and add the constraints on its value through the use of a black-box function `ADD_PREDICTOR_CONSTR()` provided by Gurobi Machine Learning [23]. The optimization problem is thus reformulated as follows:

$$\min \sum_{i \in \mathcal{I}} f_i x_i + \hat{y} \quad (18)$$

s.t.

$$\text{ADD_PREDICTOR_CONSTR}(\Phi, x, \hat{y}) \quad (19)$$

$$x_i \in \{0, 1\} \quad \forall i \in \mathcal{I} \quad (20)$$

From the model we can observe there is no constraints on fulfilling the demands in the MIP and the constraints on demands depend entirely on the surrogate model. Therefore, it is important that the network is trained thoroughly, especially with the rare cases. Insufficient data on rare solutions with few or no open facilities in the training set, may introduce bias to the network, leading to inaccurate predictions in such instances, thereby affect the effectiveness of the proposed solving approach.

5 Experimental Results

In this section, we present preliminary experimental findings on the predictive and solving capabilities of the proposed methodology.

5.1 Dataset and Settings

The test instance is derived from a classic benchmark for the capacitated facility location problem [24]. For the stochastic SSCFLP, we assume customer demands follow a uniform distribution, fluctuating within $\pm 30\%$.

The training dataset is generated by solving the recourse problem with 100 scenarios for 1000 randomly generated first-stage solutions. The training-validation set split ratio is 0.8. With 2^m possible combinations for m facilities, identical sets of facilities are infrequently repeated. The number of scenarios for generating the training dataset can be increased, while for this test instance with 16 facilities and 50 customers, 100 scenarios are sufficient to achieve stable results. As noted in Section 4.3, the performance of the MIP model depends significantly on the quality of the trained neural network. Therefore, incorporating rare cases in the training set is essential. To ensure comprehensive coverage for the neural network, first-stage solutions with no operational facilities and those with only one opened facility are additionally included in the training set.

The experiments were implemented in Python 3.8 using Gurobi optimization solver version 11.0.0, conducted on a standard personal computer with an Intel Core i7-7567U CPU @ 3.50GHz and 16GB RAM. The maximum solving time was set to 1200 seconds.

5.2 Predictive and Solving Capabilities

To assess the performance of the trained network, we employ three indicators. The coefficient of determination r^2 evaluates the fit of the model's predictions to the actual values, where a score of 1 indicates a perfect fit and a score of 0 suggests that the predictions are no more accurate than simply using the mean of the target variable $\bar{y}$. Negative values can occur if the model performs worse than using the mean. The MAPE measures accuracy as the average absolute percentage difference between actual and predicted values. The two indicators are calculated as follows:

$$r^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \quad (21)$$

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \cdot 100 \quad (22)$$

Given that the optimal solution of the SP can be derived from a small set of scenarios, we use it as a baseline for comparison. The optimal solution x^* attained through SP is treated as unseen, since it is unlikely to appear in the randomly generated training set. An additional validation check is performed on this particular first-stage solution x^* . The prediction value $\hat{y}^*$ is compared to the objective function value y^* of x^* as the third indicator:

$$\text{CHECK} = \frac{y^* - \hat{y}^*}{y^*} \cdot 100 \quad (23)$$

Table 1. Performance of neural networks

NN Structure	r^2	MAPE(%)	CHECK(%)	c.var	constr.
10x5	0.68	4.57%	4.42%	254	50
10x10	0.46	5.91%	0.24%	354	100
10x15	-0.71	9.48%	3.45%	454	150
20x5	0.78	4.02%	4.56%	354	100
20x10	0.81	3.72%	3.53%	554	200
20x15	0.84	3.44%	1.49%	754	300
30x5	0.63	5.32%	4.36%	454	150
30x10	0.76	3.88%	3.07%	754	300
30x15	0.82	3.68%	3.99%	1054	450

Table 2. Performance of the MIP using the neural networks as surrogate models

NN Structure	Best Objective	Best Bound	Gap(%)	Time (s)	Δ (%)
10x5	1.09E+06	1.09E+06	0.00%	0.52	1.53%
10x10	9.41E+05	8.81E+05	6.33%	1.69	2.16%
10x15	1.12E+06	1.01E+06	9.76%	18.65	23.62%
20x5	1.07E+06	1.06E+06	1.52%	0.62	0.45%
20x10	1.13E+06	1.08E+06	4.59%	57.23	1.44%
20x15	1.21E+06	1.11E+06	8.72%	133.1	5.48%
30x5	9.05E+05	8.89E+05	1.69%	4.98	3.75%
30x10	1.16E+06	1.11E+06	3.73%	110.61	1.96%
30x15	1.15E+06	9.11E+05	20.65%	1200	3.51%

To evaluate the quality of the optimal solutions $\hat{x}^*$ obtained by the MIP using the trained neural network as surrogate models, we report the objective function value, bound, gap(%) and the solving time. For a precise comparison with the known optimal solution of the original SP problem x^* , we assess performance $\mathcal{P}_S(x) = f(x) + \frac{1}{S} \sum_{s=1}^S Q(x, \xi_s)$ of the optimal solutions over 1000 scenarios and calculate their difference in percentage, denoted as Δ :

$$\Delta = \frac{\mathcal{P}_{1000}(x^*) - \mathcal{P}_{1000}(\hat{x}^*)}{\mathcal{P}_{1000}(x^*)} \cdot 100 \quad (24)$$

Six neural networks were trained with varying configurations of layers and neurons per layer, named by "number of neurons per layer x number of layers". In addition to the aforementioned indicators, we also provide the number of continuous variables (c.var) and general constraints (constr.) generated by each network when embedded into the MIP, for comparison. Table 1 shows the prediction capabilities of the networks and Table 2 presents the solving performance of the MIP using the trained neural networks as surrogate models.

From Table 1 we can observe that most networks offer favorable predictions of the recourse function value, apart from the 10x15 network. Networks with a small number of neurons per layer (i.e., 10) tend to get confused when additional layers are added and show worse performance. Conversely, networks with an adequate number of neurons per layer get higher r^2 scores as the number of

layers increases, but without gaining extra accuracy in MAPE. Larger networks generate more constraints when embedded into the MIP, resulting in increased gap and solving time in Table 2. However, when comparing the optimal solutions obtained by MIP using different neural networks as surrogate models with the known optimal solution of the original SP problem, the Δ value remains below 5% for all networks except 10x15, which already shows poor prediction performance in Table 1. Notably, even a small network such as 20x5 can provide a near-optimal solution to the original SP problem in 0.62 seconds, with a difference of 0.45% in objective function value. Overall, this shows the effectiveness of the approach.

6 Conclusions

In this study, we introduce a novel approach for training a neural network to predict the recourse function value of two-stage stochastic programming problems. By offline training the neural network to predict the recourse function value, it can be used as a surrogate model within the original problem-solving framework. This approach effectively avoids practical constraints imposed by scenario limitations, thereby reducing problem complexity and offering an alternative solution method for challenging problems.

Acknowledgments. This research was funded by the Italian Ministry of University and Research through PRIN2020 project grant number 20207C8T9M.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Infanger, G. (ed.) Stochastic Programming. The State of Art in Honor of George B. Dantzig. Springer, New York (2010)
2. Barro, D., Consigli, G., Varun, V.: A stochastic programming model for dynamic portfolio management with financial derivatives. *Journal of Banking & Finance*, **140**, 106445 (2022)
3. Gupta, S., Chaudhary, S., Chatterjee, P., Yazdani, M.: An efficient stochastic programming approach for solving integrated multi-objective transportation and inventory management problem using goodness of fit. *Kybernetes*, **51**(2), 768-803 (2022)
4. Bhattacharya, A., Kharoufeh, J.P., Zeng, B.: Managing energy storage in microgrids: A multistage stochastic programming approach. *IEEE Transactions on Smart Grid*, **9**(1), 483-496 (2018)
5. Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W.: The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, **259**(3), 801-817 (2017)
6. King, A., Wallace, S.W.: Modeling with stochastic programming. Springer Series in Operations Research and Financial Engineering, New York (2012)
7. Fakhri, A., Ghatee, M., Fragkogios, A., Saharidis, G.K.D.: Benders decomposition with integer subproblem. *Expert Systems with Applications*, **89**, 20-30 (2017)

8. Kaut, M., Wallace, S.W.: Evaluation of scenario-generation methods for stochastic programming. *Pacific Journal of Optimization*, **3**(2), 257-271 (2007)
9. Li, C., Grossmann, I.E.: A Review of Stochastic Programming Methods for Optimization of Process Systems Under Uncertainty. *Frontiers in Chemical Engineering, Sec. Computational Methods in Chemical Engineering*, **2**, 34 (2021)
10. Jaffray, J.-Y.: Bayesian updating and belief functions. *IEEE Transactions on Systems, Man, and Cybernetics*, **22**(5), 1144-1152 (1992)
11. Narum, B.S., Fairbrother, J., Wallace, S.W.: Problem-based scenario generation by decomposing output distributions. *European Journal of Operational Research*, <https://doi.org/10.1016/j.ejor.2024.04.006> (2024)
12. Candelieri, A., Chou, X., Archetti, F., Messina, E.: Generating Informative Scenarios Via Active Learning. In: Bruglieri, M., Festa, P., Macrina, G., Pisacane, O. (eds) *Optimization in Green Sustainability and Ecological Transition*. ODS 2023. AIRO Springer Series, vol 12, pp. 299-310, Springer, Cham (2024).
13. Chou, X., Messina, E.: Problem-Driven Scenario Generation for Stochastic Programming Problems: A Survey. *Algorithms*, **16**, 479 (2023)
14. Diao, S., Sen, S.: Distribution-free algorithms for learning enabled optimization with non-parametric estimation. *Management Science*, **66**(3), 1025-1044 (2020)
15. Dupačová, J., Gröwe-Kuska, N., Römisch, W.: Scenario reduction in stochastic programming. *Mathematical Programming*, **95**(3), 493-511 (2003)
16. Römisch, W.: Scenario Reduction Techniques in Stochastic Programming. In: Watanabe, O., Zeugmann, T. (eds) *Stochastic Algorithms: Foundations and Applications*. SAGA 2009. Lecture Notes in Computer Science, vol 5792, pp. 1-14, Springer, Berlin, Heidelberg (2009).
17. Prochazka, V., Wallace, S.W.: Scenario tree construction driven by heuristic solutions of the optimization problem. *Computational Management Science*, **17**(2), 277-307 (2020)
18. Bengio, Y., Frejinger, E., Lodi, A., Patel, R., Sankaranarayanan, S.: A learning-based algorithm to quickly compute good primal solutions for Stochastic Integer Programs. In: Hebrard, E., Musliu, N. (eds) *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. CPAIOR 2020. Lecture Notes in Computer Science, vol 12296, pp 99-111, Springer, Cham (2020).
19. Dumouchelle, J., Patel, R., Khalil, E.B., Bodur, M.: Neur2SP: Neural Two-Stage Stochastic Programming. *arXiv:2205.12006* (2022)
20. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. *Constraints*, **23**, 296-309 (2018).
21. Schweidtmann, A.M., Bongartz, D., Mitsos, A.: Optimization with Trained Machine Learning Models Embedded. In: Pardalos, P.M., Prokopyev, O.A. (eds) *Encyclopedia of Optimization*, pp 1-8, Springer, Cham (2022)
22. Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: a view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*, pp 6232-6240, Curran Associates Inc., Red Hook, NY, USA (2017)
23. <https://gurobi-machinelearning.readthedocs.io/en/stable/>, last accessed 21/05/24
24. Beasley, J.E.: An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research*, **33**, 314-325 (1988)